UNIVERSITY OF MIAMI

OPTIMIZATION-BASED ANIMATION

By

Harald Schmidl

A DISSERTATION

Submitted to the Faculty
of the University of Miami
in partial fulfillment of the requirements for
the degree of Doctor of Philosophy

Coral Gables, Florida
May 2002

UNIVERSITY OF MIAMI

A dissertation submitted in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

OPTIMIZATION-BASED ANIMATION

Harald Schmidl

Approved:

Victor J. Milenkovic, Ph.D. Chair of the Department of Computer Science	Steven G. Ullmann, Ph.D. Dean of the Graduate School
Shahriar Negahdaripour, Ph.D. Department of Engineering	Ronny Aboudi, Ph.D. Department of Management Science
Christian Duncan, Ph.D. Department of Computer Science	James Nearing, Ph.D. Department of Physics

SCHMIDL, HARALD (Ph.D., Interdepartmental Studies)
Optimization-Based Animation (May 2002)

Abstract of a doctoral dissertation at the University of Miami.

Dissertation supervised by Professor Victor J. Milenkovic.

No. of pages in text: 162

A new paradigm for rigid body simulation is presented and analyzed. 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. The extremely small integration steps inherent to traditional simulation techniques are avoided.

Contact points are synchronized at the end of each frame. Resolving contacts with friction is known to be a difficult problem. Analytic force calculation can have ambiguous or non-existing solutions. Purely impulsive techniques avoid these ambiguous cases, but still require an excessive and computationally expensive number of updates in the case of many simultaneous contacts. It is shown informally that even taking into account advances in stiff integration techniques, penalty force methods cannot overcome this issue of running time in highly crowded scenes. New algorithms are presented that calculate *simultaneous*

impulses to resolve collisions and static contacts under the Coulomb friction model. The simultaneous impulses are the solution to a QP.

In addition, the algorithms apply "bouncing at distance" and "freezing of bodies" to further speed up the simulation. These new QP algorithms are hybridized with a traditional priority queue momentum update scheme to allow sequential impulses when they are required for realism, such as in the office toy pendulum. When added to the implementation of OBA, these new algorithms increase the speed of the simulation by a factor of up to 30.

The position update has been hybridized with retroactive detection (RD) to prevent fast and thin bodies from passing through each other. Due to the modular design of the OBA simulator, the described techniques can be used as components in any existing simulator that follows a modular design of position update, finding contacts, and resolving contacts. Non-convex bodies are simulated as unions of convex bodies. Links and joints are simulated with bi-directional constraints. Analysis of the algorithm and discussion of example simulations are provided.

Dedication

In loving memory of my father.

Acknowledgments

I wish to thank my friends who have accompanied me during the last four and a half years, who have answered questions and helped out in tight spots.

My advisor, Victor J. Milenkovic, had the trust to take me as his student and funnelled his seemingly never-ending stream of ideas into me. I want to express my gratitude not only for his generous funding of my dissertation research and conference attendances, but also for his support in non-academic events over the past years. Thanks also to my other committee members who backed me up in the changes of the program as it took shape.

I owe gratefulness to my mother, who has a unique way of silent understanding and accepting. Needless to say she has been there since the very beginning.

My dear appreciation goes to my wife Jessica who knew the right words in times of stress. You did a good job in convincing me that not always "todo esta chimbo".

I also want to thank the people at CPLEX for discussion of problems with their QP solver, Brian Mirtich for sharing his experience with rigid body physics over dinner and through many emails, Adam McMahon for his expertise in Maya and rendering my simulations for the video, and Hüseyin Koçak for his explanation of stiff ODEs.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Graphics and Animation

Today, computer graphics (CG) is a mature discipline within computer science. It has a deep impact on modern everyday life. People may not be aware of theoretical or technical details, but everybody is familiar with computer generated images from movies, commercials, and applications. Scientists have achieved levels of amazing realism in rendering artificial scenes. We consider this an experimental proof that existing techniques work. However, many of the computations involved in generation of computer images and movies are expensive, and therefore research is continuing to devise more efficient algorithms.

In this treatise, we explore a branch of graphics called animation. Pure graphics is concerned with the qualities of an actual image. In contrast, animation is concerned with physical motion. Hand-created character cell animation had been made popular a long time ago and has accompanied our lives from early childhood on. With advances in computer technology and fast high quality PCs becoming available at affordable cost, people became more and more interested in letting the computer take over some of the

work in animation. It is sometimes hard for a human to hand-generate physically accurate motion. Instead, humans now devise general rules and goals, and the computer does the animation work.

We can think of animation as an application of the laws of physics. It is generally desirable to achieve a high level of physical accuracy, yet animations ought to be computationally efficient. Modeling true physics can be very expensive and often not even necessary. Depending on the particular task, a *plausible* simulation of physics may be sufficient. We call an animation plausible if its motion is believable. It has been shown by Chenney and Forsyth [24] that the concept of plausibility can be of great use for animation purposes. Artistic freedom is a general trait of graphics. While accuracy for computations in meteorology or manned space flight is enormously crucial, graphics applications with believable visual appeal are acceptable most times.

Much progress has been made in applied fluid dynamics lately [32]. Some recent feature films had giant water waves or scenes of storms at sea. The ocean water in the film *Titanic* was completely generated with CG techniques. Close-up scenes inside the hull of the ship were acted out in tanks. There was no real ocean water seen in any scene of the movie at all. Most movie goers have probably not realized this fact. Closely related to liquids, we find simulation of smoke for visual effects [31]. The underlying physics has been studied extensively for over a century. Computer scientists take these theories and modify them to an extent where they allow generation of nice efficient simulations for movies. Physics is altered, not only for reasons of computational efficiency, but also for reasons of visual appeal. If it looks better, then it is acceptable in the graphics world to change the math. Graphics at this point is not exclusively concerned with reality.

Following this philosophy, we examine a niche in animation called "rigid body

physics". Rigid body physics was studied centuries ago by the old masters of physics, such as Newton, Lagrange, and Hamilton [37]. Their great achievement was the application of mathematics to describe nature. Classical mechanics formulates differential equations to model physical systems. This unprecedented breakthrough in science allowed the prediction of future states of physical systems in motion when initial conditions were known. Standard examples are the equations of motion of a pendulum, a projectile, or the planets. Scientists had hoped for a long time that they would be able to describe all surrounding nature by mechanics at some point. However, those hopes were dashed with the advent of quantum physics. The deterministic nature of classical physics is no longer assured in the quantum world. For our purposes, classical mechanics is the proper vehicle, since we are simulating macroscopic effects.

1.2 Problem

We are concerned with simulation of interacting bodies in motion. A ball will bounce back when it hits a wall. The nature of the bounce will depend on material properties of ball and wall and the magnitude of velocities and forces. The benefits of research in this field become evident when we look at its versatility. Modern research even addresses topics of sound generation from the gathered knowledge of colliding bodies that are made of certain materials. Among others, O'Brien *et al.* [68] have devised methods that allow on the fly sound generation for the purposes of animation. Rather than a human animator meticulously synchronizing an animation with audio, the animation itself generates appropriate sounds for events, such as colliding or scratching of bodies against each other.

Physicists had developed various models for interactions between rigid bodies a long

time ago. Although some of these models are simplified to a frictionless world, models for friction also exist. It is vital for a realistic feel of any simulation to model body interactions with friction. Various simulation methods have been developed by researchers [5, 7, 12, 39, 53, 57, 59, 63]. They agree that the presence of friction makes simulation harder. Some existing simulation methods work well only for particular problems or are of limited practical value due to poor performance.

Despite the stunning quality of some current animations in movies, it remains a challenge to cope with computational problems that arise when the number of interactions between bodies grows greatly. Traditional techniques scale poorly, and for many applications this problem remains an important issue of research. Computer game developers are well aware of these problems and have a great interest in finding efficient solutions. Computer games are real-time applications and are therefore computationally demanding.

The movie industry is very interested in simulation techniques. CG techniques for special effects have become popular due to their relatively low production cost but high level of realism. Modeling explosions with CG is cheaper but also safer than filming real explosions. The same goes for the scenes of storms at sea in *The Perfect Storm*. Not only would it be hard to find the depicted situation in nature, but it would be also very dangerous to send a film crew and cast into it. Building physical models of ancient cities is laborious and can even not look as realistic as a computer model, especially for filling in the background. Even full feature length movies that are entirely animated with CG are common. An especially hard problem is the simulation of hair [1, 16]. One of the reasons why movies like to feature bugs and insects is that these animals have no fur or hair. They are therefore easier to animate. A physical correct simulation of hair would have to treat each hair as a tube-like flexible body that collides with its neighbors. With current

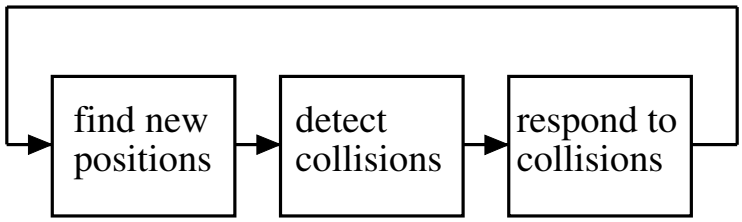


Figure 1.1: The simulation loop.

technology, this is an immensely challenging task and is mostly solved by approximate interpolating techniques. Simulation of fur and hair takes up a tremendous amount of computation time in new animation productions, *e.g.* the new movie *Monsters, Inc.*

From a practical viewpoint, simulating a system of interacting bodies in motion is equivalent to generating a movie with a computer. Our work is not concerned with the actual rendering, but the computation of physically realistic, or plausible body states. Baraff and Witkin give a comprehensive, practical, hands-on introduction into issues involved [6]. These notes are especially of interest for the novice in the field, since code and implementation tips are provided in addition to basic physics concepts. For more analytic background information and problem assessment, the reader might find several other papers to be of interest [4, 5, 60].

A movie consists of a series of frames. When played back in a consecutive fashion, the recorded motion becomes visible. The current standard of video is a frame rate of 30 frames/sec. Any simulation algorithm integrates the body trajectories and detects and resolves collisions if bodies interpenetrate. Non-penetration constraints are the "golden rule" of rigid body simulation, and all simulation systems enforce them in one way or another. When the system has reached the end of a full frame time, a redraw command is issued and a snapshot of the scene is displayed. This process loops continuously around. It has been coined the *simulation loop* (Fig. 1.1) by Mirtich [59].

Traditionally, and intuitively, many simulation algorithms have a basic idea in common. As bodies follow their paths they will interact and collide. We adopt the commonly used terms *collisions* and *static contacts* to classify body interactions. Each time a collision occurs, the motion is stopped, velocities are recalculated, and the process repeats. The problem with high numbers of collisions becomes evident for the example of a ping-pong paddle which is brought down over a bouncing ball on a table. As the paddle is lowered, the frequency of bounces becomes higher. Thus, the time interval that passes without any bounce becomes smaller. Figure B.1(a)[1] shows a stack of cubes. This seemingly simple scene is in fact very hard to simulate. Because the cubes are so tightly packed, virtually no time passes at all between collisions. Many *micro-steps* are taken to advance the system for just a frame time of 1/30 sec. Eventually, a simulation can be slowed to a crawl and can even appear to be completely frozen. Consider that for an admissible time-step of 10^{-6} sec., 33,333 micro-steps will be taken to reach the next frame time. Huge amounts of computations are used to simulate motion in-between frames that the human eye does not even get to see. It is true that the micro-steps in-between frames will influence the outcome of the final simulation. However, as for the case of many applications of simulation, plausible motion is perfectly sufficient. Dancing beverage cans and toothbrushes as we know them from TV commercials are different in this aspect from simulation of manned space flight or predicting hurricanes. For the latter, accuracy is much more crucial.

We address the issue of highly "crowded" physical systems by maintaining physical accuracy where it matters and trading it off for speed where it is admissible [58]. The OBA technique 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). Overlap

[1]Color plates appear at the end of the text.

is resolved by enforcing non-overlap constraints. To do so, we employ methods from the field of mathematical programming, in particular quadratic programming (QP). We use optimization to compute new body positions. For all resulting contact points, impulses and contact forces are computed with Coulomb friction. We also devise a purely impulsive collision and contact model, which facilitates calculations under the Coulomb friction model greatly. Several techniques can be used to reduce the number of contacts and achieve faster simulation speeds. Optimization techniques prove to be extremely powerful tools for our purposes.

The field of mathematical programming has been well researched and packages are readily available. These packages can be used in our research, and we directly benefit from the developments in optimization techniques. In a way, we devise the physical model and solve it with the most powerful black box math-module available. It seems a fruitful direction in graphics in general to combine disciplines. Most creative work in graphics is fueled by physics and mathematical techniques.

We have implemented a simulator, and the results are visually convincing. The algorithm is well suited to simulate large stacks of bodies with a high number of static contacts. The quality of a simulation algorithm can be measured with the following criteria. Any simulation algorithm has to be numerically stable, computationally efficient, yet it must be realistic enough to satisfy the particular application's needs. Numerical stability is an important issue in simulation. Stability becomes visible in simulations that exhibit no jumps or jerks. Probabilistic reasoning suggests that in a scene with many interacting bodies, almost all frames will have degeneracies in the contact geometry. One cannot ignore bad or hard to compute cases. After years of experimenting it is our experience that everything bad that can happen actually will happen if the animated sequence is

long enough and non-trivial. Existing methods do not always handle degeneracies in a satisfying manner. It is a common argument that degeneracies are rare and can therefore be neglected. This is an *ad hoc* and quite dangerous way to handle the problem if the simulated scene is complicated. A numerically stable algorithm will have to detect and handle bad cases.

OBA makes approximations regarding physical realism and generates plausible motion. One might argue that plausibility is a disadvantage of OBA. Existing techniques claim to be exact, but use heuristics, sometimes even arbitrary ones, to deal with degeneracies. For complicated simulations where almost all frames have degeneracies, heuristics will clearly also reduce realism. In contrast to heuristics, OBA poses a firm solution to the problem. It addresses problems found in rigid body simulation with a stable, reliable algorithm.

We mentioned that the movie and computer gaming industries are two major target groups for our kind of research. Both are extremely willing to sacrifice some realism for performance as long as visual credibility is preserved. In movie productions, meeting deadlines and not exceeding production cost are far more important than textbook physics behavior. Plausible animation is therefore not only a convenient idea for a researcher, but it has accepted and welcomed real life applications.

1.3 Overview

The remainder of this thesis is structured in the following way:

Chapter 2 introduces preliminaries. We explain rigid body concepts from physics and briefly state the important facts about mathematical programming, *i.e.* linear and

quadratic programming.

Chapter 3 then describes collision detection and presents algorithms used in distance calculation. Bounding box techniques and coherence exploitation are included in this discussion. The latter are important for efficiency reasons.

Chapter 4 steps further and presents the mechanics of collision resolution. We explain collision handling with impulses and static contact resolution with forces. Coulomb friction is introduced here.

Chapter 5 discusses related previous work and simulation concepts. Traditional simulation paradigms are analyzed, their advantages and weaknesses are explained.

Chapter 6 uses the gained knowledge to derive our new paradigm, optimization-based animation. We explain the big picture first and then details about non-overlap constraints, objective, and iterative update of the position algorithm. QP-based momentum and force update algorithms are given, as well as hybridization methods, explanation of purely impulsive contact resolution, handling non-convex and linked bodies, and possibilities for speed-up.

Chapter 7 presents experiments, which we conducted to examine the performance of our OBA algorithm and its implementation. We analyze the component algorithms and give insight into the expected running time of the simulations.

Chapter 8 investigates possible future research directions. We discuss next steps, as well as currently immature developments.

Chapter 9 finally presents possible applications of our techniques and rounds off this thesis by giving conclusions.

The **Appendix** explains some implementation details. A summary of quadratic programs used in this work is included, and all color plates are placed here also.

Chapter 2

Preliminary Techniques

We introduce important preliminary concepts in this chapter. It is important for the understanding of this work to be familiar with the physics of rigid bodies and techniques used in mathematical programming. We will introduce the notation used in this dissertation. We give a standard textbook introduction to classical rigid body mechanics, and we explain the concepts of *linear* and *quadratic programming* briefly.

2.1 Rigid Body Physics

Rigid body physics is studied in classical mechanics. Although any text on the subject gives a good introduction, the book by Goldstein [37] is seen as the standard in many places. Mechanics is an old branch in physics. Centuries ago, people like Newton have contributed to its understanding. The problem at the root of rigid body mechanics is the formulation of equations of motion for an arbitrarily shaped rigid body. The concept of rigidity assumes a perfectly non-deformable body. Rigidity is an idealized view and *per se* not found in real nature [22, 74]. On a microscopic level, even extremely hard bodies,

such as diamonds, deform slightly when they collide with each other. It is however for the purpose of didactics and for most applications acceptable to assume rigidity. The concepts of rigid body physics are implemented by any simulation program or simulator for short.

2.1.1 Mass and Center of Mass

We can think of a rigid body as a blob of particles with fixed relative pairwise distance. Physics texts sometimes use the image of particles that are connected with weightless rigid rods or stiff springs [35, 37]. The sum of all particle masses m_i yields the total body mass:

$$m = \sum_i m_i. \tag{2.1}$$

Furthermore, given the coordinate $\mathbf{x}_i$ for the i-th particle, we calculate the *center of mass* coordinate according to:

$$\mathbf{x}_{cm} = \frac{\sum_i \mathbf{x}_i m_i}{\sum_i m_i}. \tag{2.2}$$

For continuous and homogeneous bodies, *i.e.* bodies with uniform density, the sums in Equations 2.1 and 2.2 will have to be replaced by integrals:

$$m = \int dm, \tag{2.3}$$

$$\mathbf{x}_{cm} = \frac{\int \mathbf{x} dm}{\int dm}. \tag{2.4}$$

Mass is related to body volume by the material density, $dm = \rho \cdot dV$.

2.1.2 Position and Orientation

We maintain a world coordinate system (x, y, z) to measure absolute coordinates. It is convenient to express some body properties in a body-fixed coordinate system (x', y', z'). It is centered in the body's center of mass (Fig. 2.1). For a rigid body, all body points stay

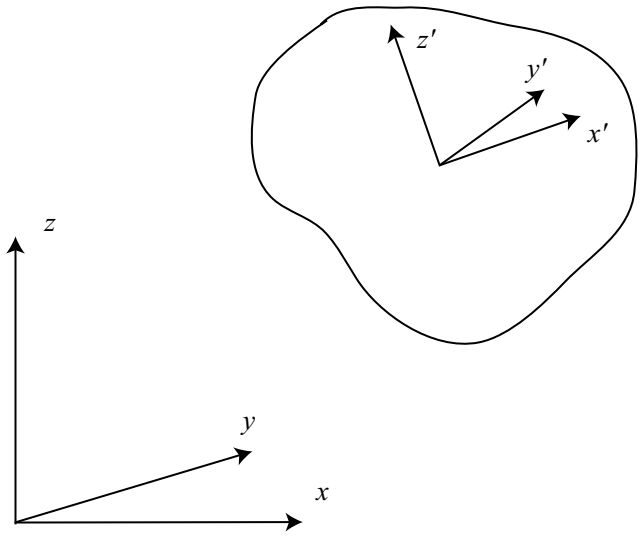


Figure 2.1: The world and body-fixed coordinate system.

invariable in the body-fixed system. A rigid, three-dimensional body has *six* degrees of freedom. We distinguish three *translational* degrees and three *rotational* degrees for center of mass position and orientation respectively. We denote translation of a body by the vector $\mathbf{x}$ and the orientation by the matrix $\mathbf{R}$. The columns of $\mathbf{R}$ have a physical meaning. The three column vectors of $\mathbf{R}$ give the directions in world coordinates into which the three principal body axes point. These three *nomalized* vectors are pairwise *orthogonal*. We can view the first column as the direction into which the body's x'-axis points and so forth.

A point $\mathbf{p}^{body}$ in body coordinates is easily translated into world coordinates by:

$$\mathbf{p}^{world} = \mathbf{R}\mathbf{p}^{body} + \mathbf{x}. \tag{2.5}$$

There are different ways of expressing the orientation of a body. Intuitively, orientation can be viewed as an axis and an angle around it. We can denote this with vector $\boldsymbol{\varphi}$ that has direction equal to the axis and magnitude equal to the rotation vector. This is a convenient representation for developing our equations. For the actual computations, however, it is more practical to use different ones. Appendix A.1 explains the basics of quaternions and

describes how we can convert between the three orientation representations used in our work, *i.e.* rotation matrices, rotation vectors, and quaternions. The three-dimensional rotation vector $\boldsymbol{\phi}$ points along the rotation axis and has magnitude $\tan\varphi$, where φ is the angle of rotation in radians counterclockwise about the rotation axis. If $\boldsymbol{\phi}$ is the rotation vector equivalent to the rotation matrix $\mathbf{R}$, we can use the small-angle approximation (φ is small) and write

$$\mathbf{R}\mathbf{w} \approx \mathbf{w} + \boldsymbol{\phi} \times \mathbf{w} \tag{2.6}$$

for any vector $\mathbf{w}$.

The use of rotation matrices has been criticized due to numerical issues involved. As a rotation matrix is updated while a body follows its path, numerical drift can cause introduction of errors, which destroy the orthogonality of the matrix. The visual effect would be a skewing of a body when the matrix is applied to it. In other words, the matrix is not a rotation matrix anymore. The use of *Euler angles* exhibits similar quirks since it can lead to *gimbal lock*. Gimbal lock occurs in the Euler angle representation of rotation, because it applies three successive rotations around three axes independently. This ignores the cross product interaction of rotations, can lead to alignment of axes, and therefore loss of one degree of rotational freedom. Shoemake describes *quaternions* for the purpose of animating rotation [77]. They allow natural and simple interpolation of motion in-between frames, and they produce stable, smooth motion. In fact, they have since long been used for navigation of spacecraft exactly for those reasons.

2.1.3 Inertia

After explaining orientation, we can now introduce the *moment of inertia*, the rotational counterpart of mass. Let's assume a body made of rigidly attached particles for simplicity

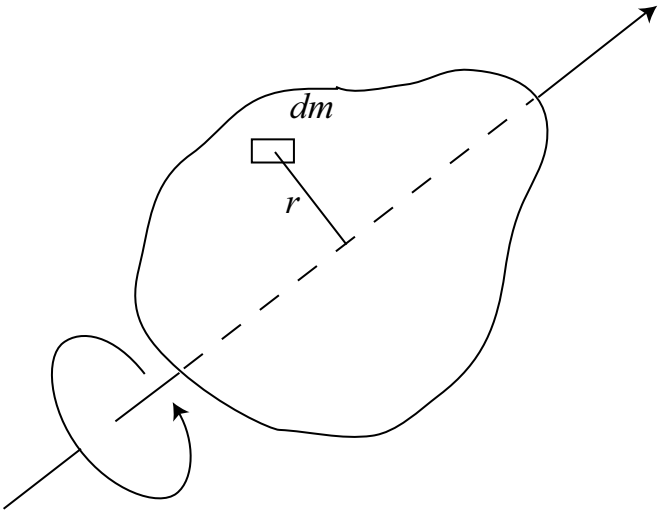


Figure 2.2: Moment of inertia: sum of mass elements times their squared distance from the rotation axis.

again. For an arbitrary rotation axis, the body's moment of inertia is denoted by the sum of particle masses times the squared particle's distance to the axis of rotation (Fig. 2.2)[1]:

$$J = \sum_i r_i^2 m_i. \tag{2.7}$$

An important concept of mechanics is the inertia *tensor* $\mathbf{I}$. It is a 3×3 matrix that represents the distribution of mass in the body relative to three axes of an arbitrary coordinate frame. The elements I_{ij} of $\mathbf{I}$ with $i, j \in \{x, y, z\}$ can be calculated in the following manner. Let (r_x, r_y, r_z) denote the displacement vector of the body's mass elements from the body's center of mass. The diagonal elements, $i = j$, of $\mathbf{I}$ are

$$I_{ii} = \int (r_x^2 + r_y^2 + r_z^2 - r_i^2)\rho dV, \tag{2.8}$$

and all others, $i \neq j$, are

$$I_{ij} = -\int (r_i r_j)\rho dV. \tag{2.9}$$

We initially calculate $\mathbf{I}_0$ [37, 39] in a body-fixed frame and transform it to the current orientation if needed. This avoids expensive recalculation of $\mathbf{I}$ from scratch whenever a body

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

rotates and its mass is redistributed relative to the coordinate frame at hand. The mass distribution in a body-fixed frame is constant, and so is $\mathbf{I}_0$. We can derive $\mathbf{I}$ at the current orientation $\mathbf{R}$:

$$\mathbf{I} = \mathbf{R}\mathbf{I}_0\mathbf{R}^T. \tag{2.10}$$

2.1.4 Velocities, Momenta, and Kinetic Energies

A body's position and orientation change with the body *velocities*. We distinguish *linear* and *angular*, or *rotational*, velocity. These properties denote the rate of change in position and orientation. For the linear velocity this is simply

$$\mathbf{v} = \dot{\mathbf{x}}. \tag{2.11}$$

Represent the body orientation with vector $\boldsymbol{\varphi}$ that has direction equal to the rotation axis and magnitude equal to the rotation angle. We yield then similarly for the angular velocity:

$$\boldsymbol{\omega} = \dot{\boldsymbol{\varphi}}. \tag{2.12}$$

Hand in hand with the velocities, we find linear and angular momentum. The linear momentum is:

$$\mathbf{p} = m\mathbf{v}. \tag{2.13}$$

Similarly, for the angular momentum we write:

$$\boldsymbol{\ell} = \mathbf{I}\boldsymbol{\omega}. \tag{2.14}$$

Another important concept is the notion of kinetic body energies. Linear kinetic energy is:

$$E_{lin} = \frac{1}{2}m\mathbf{v}^2 = \frac{\mathbf{p}^2}{2m}. \tag{2.15}$$

Similarly for the rotational kinetic energy:

$$E_{rot} = \frac{1}{2}\boldsymbol{\omega}^T\mathbf{I}\boldsymbol{\omega} = \frac{\boldsymbol{\ell}^2}{2\mathbf{I}}. \tag{2.16}$$

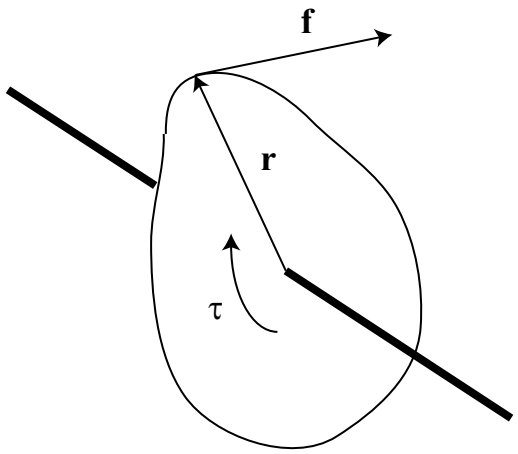


Figure 2.3: A torque $\boldsymbol{\tau} = \mathbf{r} \times \mathbf{f}$ is the result of a force $\mathbf{f}$ acting with lever $\mathbf{r}$.

2.1.5 Accelerations and Forces

The body velocities' rate of change is attributed to the body *accelerations*. We introduce the linear acceleration $\mathbf{a}$ and angular acceleration $\boldsymbol{\alpha}$. Over time, the accelerations modify the body velocities and thus the momenta:

$$\begin{aligned} \dot{\mathbf{v}} = \ddot{\mathbf{x}} = \mathbf{a} \quad &\Rightarrow \quad \dot{\mathbf{p}} = m\dot{\mathbf{v}} = m\mathbf{a}, \\ \dot{\boldsymbol{\omega}} = \ddot{\boldsymbol{\varphi}} = \boldsymbol{\alpha} \quad &\Rightarrow \quad \dot{\boldsymbol{\ell}} = \mathbf{I}\dot{\boldsymbol{\omega}} = \mathbf{I}\boldsymbol{\alpha}. \end{aligned} \tag{2.17}$$

The last equality is only valid in the body-fixed coordinate system in which $\mathbf{I}$ is constant. Otherwise, time derivatives of the inertia tensor $\mathbf{I}$ will have to be included.

Gravity is an acceleration. It exerts a force on a falling body, which adds momentum to the body. This is why a falling body becomes faster and faster, or in other words accelerates. Newton expressed exactly that in his famous axioms. The first axiom states that a force-free moving body continues to move in a straight line, *i.e.* with constant velocity. The second axiom states the familiar equality of total force on the body and

linear acceleration. Together with Equation 2.17 we write:

$$\mathbf{f} = m\mathbf{a} \Rightarrow \mathbf{f} = \dot{\mathbf{p}}. \tag{2.18}$$

Its rotational counterpart is called *torque* $\boldsymbol{\tau}$. We write analogously to Equation 2.18, again in a body-fixed coordinate system:

$$\boldsymbol{\tau} = \mathbf{I}\dot{\boldsymbol{\omega}} = \mathbf{I}\boldsymbol{\alpha} \Rightarrow \dot{\boldsymbol{\ell}} = \boldsymbol{\tau}. \tag{2.19}$$

Torque is the result of a force acting on a body with a lever, $\boldsymbol{\tau} = \mathbf{r} \times \mathbf{f}$ (Fig. 2.3).

2.1.6 Equations of Motion

With the knowledge of the above mentioned body properties, we can now introduce the *equations of motion*. Knowing the equations of motion means having the ability to predict future states of a body if the initial state is known. In their exact form, the equations of motion for a free falling body are as follows:

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{v}, \\ \dot{\mathbf{v}} &= \mathbf{a}, \\ \dot{\boldsymbol{\varphi}} &= \boldsymbol{\omega}, \\ \dot{\boldsymbol{\omega}} &= \boldsymbol{\alpha}. \end{aligned} \tag{2.20}$$

Equations of this form are called *ordinary differential equations* (ODEs). Solving them can be described as an *initial value problem*. If we know the initial state variable values, we can compute future values with several numerical methods. The body states change due to the body velocities, and the velocities change due to accelerations.

2.1.7 Numerical Integration

Integrating the equations of motion allows calculation of future body states. There are different techniques for numerical integration with varying degrees of accuracy and different applicability, most notably *explicit* and *implicit* methods [72].

Explicit Integration

Explicit methods are the simplest form of numerical integration. The well known Euler's method is very easily implemented and easy to understand. Given a function $\mathbf{x}(t)$ with initial value $\mathbf{x}(t_0)$ at time t_0, we calculate the function value $\mathbf{x}(t_0 + h)$ after a step h:

$$\mathbf{x}(t_0 + h) = \mathbf{x}(t_0) + h\dot{\mathbf{x}}(t_0). \tag{2.21}$$

The Euler method is not very accurate since it is basically a Taylor expansion that throws away terms with higher order than the first derivative. This is correct if $\mathbf{x}(t)$ is linear. Otherwise, in order to achieve sufficient accuracy, a small step size h may be required which slows down the integration. Better integration methods are known [6, 72], such as the midpoint method and Runge-Kutta integration. In general, these methods achieve better accuracy by including higher order derivatives in the calculation.

Determining a good integration step size is a problem. If the step size is too small, efficiency suffers. If it is too large, accuracy suffers. Adaptive step sizing varies the integration step size and is somewhat reminiscent of machine learning. The current error is determined and the step size is adjusted to accommodate an admissible error.

Implicit Integration

Implicit methods have been applied very successfully in the simulation and animation community lately [11, 62]. ODEs can become "stiff", and explicit methods are not good at

solving these. For stiff ODEs, it is usually best to use implicit solution methods. Consider the ODE

$$\dot{\mathbf{x}}(t) = -k\mathbf{x}(t). \tag{2.22}$$

It describes a simple spring with spring constant k. Perturbing the spring from its resting state at position zero and releasing it should make it go back to zero eventually. Starting at an arbitrary perturbation $\mathbf{x}(t_0) \neq 0$, apply an Euler step:

$$\mathbf{x}(t_0 + h) = \mathbf{x}(t_0) + h(-k\mathbf{x}(t_0)) = (1 - hk)\mathbf{x}(t_0). \tag{2.23}$$

For $|1 - hk| > 1$, the future value will be larger than the initial value $\mathbf{x}(t_0)$, *i.e.* the spring never reaches its resting state at zero again. Therefore, for large spring constant k, we would have to make the step size h enormously small if we wanted to avoid $|1 - hk| > 1$. However, for a small step size the spring will crawl back to zero excruciatingly slow. Hence, trying to simulate this simple system will be computationally very frustrating.

Implicit integration can be understood as a feedback process: find a future state $\mathbf{x}(t_0 + h)$ that would lead back to the desired initial state if we reversed the process:

$$\mathbf{x}(t_0 + h) = \mathbf{x}(t_0) + h\dot{\mathbf{x}}(t_0 + h). \tag{2.24}$$

Let $\mathbf{x}(t_0 + h) = \mathbf{x}_{new}$, $\mathbf{x}(t_0) = \mathbf{x}_0$, and $\frac{d}{dt}\mathbf{x}(t) = f(\mathbf{x}(t))$:

$$\mathbf{x}_{new} = \mathbf{x}_0 + hf(\mathbf{x}_{new}). \tag{2.25}$$

Introduce $\Delta\mathbf{x}$ with $\mathbf{x}_{new} = \mathbf{x}_0 + \Delta\mathbf{x}$. Following Equation 2.25, we write:

$$\begin{aligned} \mathbf{x}_0 + \Delta\mathbf{x} &= \mathbf{x}_0 + hf(\mathbf{x}_0 + \Delta\mathbf{x}) \Rightarrow \\ \Delta\mathbf{x} &= hf(\mathbf{x}_0 + \Delta\mathbf{x}). \end{aligned} \tag{2.26}$$

We can use the Taylor approximation $f(\mathbf{x}_0 + \Delta\mathbf{x}) = f(\mathbf{x}_0) + f'(\mathbf{x}_0)\Delta\mathbf{x}$. With the identity matrix $\mathbf{I}_{id}$, we write:

$$\begin{aligned} \Delta\mathbf{x} &= h(f(\mathbf{x}_0) + f'(\mathbf{x}_0)\Delta\mathbf{x}) \Rightarrow \\ \Delta\mathbf{x} - hf'(\mathbf{x}_0)\Delta\mathbf{x} &= hf(\mathbf{x}_0) \Rightarrow \\ \left(\frac{1}{h}\mathbf{I}_{id} - f'(\mathbf{x}_0)\right)\Delta\mathbf{x} &= f(\mathbf{x}_0) \Rightarrow \\ \Delta\mathbf{x} &= \left(\frac{1}{h}\mathbf{I}_{id} - f'(\mathbf{x}_0)\right)^{-1} f(\mathbf{x}_0). \end{aligned} \tag{2.27}$$

Note, since $f(\mathbf{x}_0)$ is generally a vector, the derivative $f'(\mathbf{x}_0)$ is a matrix. Using implicit methods involves solving a linear system to calculate $\Delta\mathbf{x}$. The computational cost is justified as will become clear after going back to the example of a stiff spring (Eq. 2.22). Calculate $\Delta\mathbf{x}$ for this example with Equation 2.27:

$$\Delta\mathbf{x} = -\frac{h}{1+kh}k\mathbf{x}_0. \tag{2.28}$$

The interesting feature of implicit integration becomes clear when we investigate the limit on the step size:

$$\lim_{h\to\infty}\Delta\mathbf{x} = -\mathbf{x}_0. \tag{2.29}$$

That means $\mathbf{x}_{new} = \mathbf{x}_0 + (-\mathbf{x}_0) = 0$, *i.e.* the spring goes back to the desired rest state even if we choose a very large step size h. Being able to choose large step sizes is desirable because it will make a simulation much faster.

2.1.8 OBA Integration

The OBA algorithm is completely modular and can use any integration technique, *i.e.* also the best technique available. It turns out that OBA is so robust that even a bad method like Euler's produces good results. This is also due to the particular simulations which we

ran. Simulating a fast spinning gyro or an ice skater might make implementation of a better numerical integration method necessary. We currently use the simplest approximation to the integral of the equations of motion. Our step size is equal to the frame time. We achieve sufficient accuracy at low computational cost and little implementation effort.

If we know a body's current position $\mathbf{x}_0$ and its velocity $\mathbf{v}_0$ at time t_0, then we can calculate its future position $\mathbf{x}_{new}$ at time $t_0 + \Delta t$:

$$\mathbf{x}_{new} = \mathbf{x}_0 + \mathbf{v}_0 \Delta t. \tag{2.30}$$

If the body is accelerated, we calculate the updated velocity according to:

$$\mathbf{v}_{new} = \mathbf{v}_0 + \mathbf{a} \Delta t. \tag{2.31}$$

By computing Equations 2.30 and 2.31, we yield future states for body position and velocity. Equation 2.31 assumes constant acceleration. This is approximately the case for a body that falls through the earth's field of gravity if the height change is not too drastic. However, if the total force on a body and therefore its acceleration changes, the computation must take that into account.

Collisions and contacts between bodies are called *discontinuities*. Practically, this means that the computation stops, forces and accelerations are re-calculated, and the bodies' forward motion can continue. This is a problem that simulators have to face when many bodies collide frequently. The re-evaluation of parameters in the simulation loop (Fig. 1.1) at a high frequency necessitates large numbers of computations, which slow down the simulation.

Predicting future states for the angular case needs somewhat more thought, since orientation is a matrix. As explained earlier, the columns of $\mathbf{R}$ form an ortho-normal coordinate system representing the three principal body axes' directions. We expect intuitively

$\dot{\mathbf{R}}$ to be a matrix whose columns represent the rates of change for the original matrix's columns. This is indeed the case, let $\mathbf{R}^i$ be the i-th column of $\mathbf{R}$:

$$\dot{\mathbf{R}}^i = \boldsymbol{\omega} \times \mathbf{R}^i. \tag{2.32}$$

The columns of $\mathbf{R}_{new}$ are calculated with:

$$\mathbf{R}^i_{new} = \mathbf{R}^i + \dot{\mathbf{R}}^i \Delta t. \tag{2.33}$$

Following Equation 2.31, we attribute the change of angular velocity to a torque acting on the body resulting in angular acceleration:

$$\boldsymbol{\omega}_{new} = \boldsymbol{\omega}_0 + \boldsymbol{\alpha}\Delta t. \tag{2.34}$$

Call the set

$$\mathbf{S} = \begin{pmatrix} \mathbf{x} \\ \mathbf{R} \\ \mathbf{v} \\ \boldsymbol{\omega} \end{pmatrix} \tag{2.35}$$

the *body state*. It completely describes the body position, orientation, and velocities at a given time. The rate of change of the state is denoted by:

$$\dot{\mathbf{S}} = \begin{pmatrix} \dot{\mathbf{x}} \\ \dot{\mathbf{R}} \\ \dot{\mathbf{v}} \\ \dot{\boldsymbol{\omega}} \end{pmatrix}, \tag{2.36}$$

and therefore

$$\mathbf{S}_{new} = \mathbf{S} + \dot{\mathbf{S}}\Delta t. \tag{2.37}$$

We implemented these equations in our simulator to generate the forward motion of bodies.

2.2 Mathematical Programming

Techniques of mathematical programming have been well researched [27, 64]. High quality packages for solving *linear* (LP), *quadratic* (QP), and *integer* (IP) programming problems are readily available. Probably most commonly known is the *simplex* algorithm for solving LPs. The classic problem of linear programming is the minimization of cost or maximization of profit. Operations research is concerned with these problems, and a large number of excellent techniques and theories have been developed. We can benefit from this research in a number of ways.

The computations involved in graphics and computational geometry can in some cases create great difficulties where numerical stability is important. Due to many degeneracies in the geometry of crowded systems of bodies, robustness is especially important in animation and simulation where visual stability is crucial.

By rewriting our physics problems as LPs or QPs, most of the computation work is done inside of the LP or QP solver. Due to the high quality of available solver packages and their highly optimized performance, we can concentrate more on the actual physical theories and models. The physics is separated from the numerics. We formulate the problem and use the best available algorithm to solve it. Nowadays, QPs are solved by fast interior point methods.

2.2.1 Linear Programming

Consider the example of profit maximization or equivalently cost minimization. The interesting property that we want to maximize or minimize is called the *objective*. Several *constraints* and *bounds* dictate conditions which have to be observed. For example, a corpo-

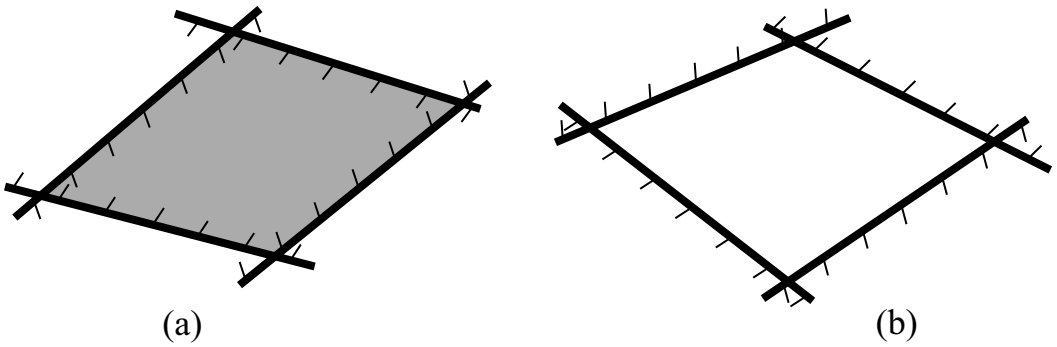


Figure 2.4: The LP is *feasible* if and only if the intersection of all half-spaces is non-empty (a). Otherwise, it is *infeasible* (b).

ration that produces several products might want to know what numbers of each it should produce in order to maximize profit. However, it has limitations regarding possible output of each, either through environmental regulations by the government, availability of raw materials, or availability of trained workers needed for production.

A linear program can be described as follows:

$$\begin{aligned} &Minimize && \sum_i c_i x_i \\ &Subject\ to && a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n \leq b_1 \\ & && a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n \leq b_2 \\ & && ... \\ & && a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n \leq b_m \end{aligned} \tag{2.38}$$

The coefficients a_{ij} and c_i, as well as the bounds b_i and variables x_i are real numbers. Except for the variables x_i, all others are constants. We can write the same LP with a different notation by using vectors $\mathbf{c} \in R^n$, $\mathbf{b} \in R^m$, $\mathbf{x} \in R^n$, and a coefficient matrix $\mathbf{A} \in R^{m \times n}$:

$$\begin{aligned} &min && \mathbf{c}^T\mathbf{x} \\ &s.\,t. && \mathbf{A}\mathbf{x} \leq \mathbf{b} \end{aligned} \tag{2.39}$$

The function to be minimized is called the *objective function*. It can be viewed as a direction

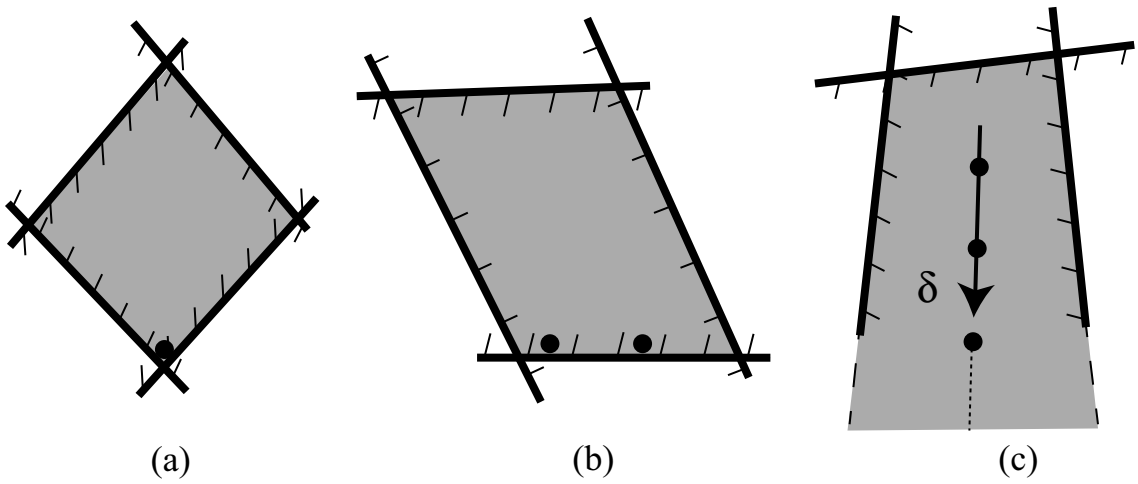


Figure 2.5: Assume the objective function's gradient points vertically "down": *unique optimal* solution (a), several equally good, *alternative optimal* solutions (b), and an *unbounded* LP (c).

in n-dimensional space. Likewise, the linear constraints can be viewed as n-dimensional half-spaces. Solving the above LP means finding values for the x_i which minimize the objective function under the condition that all constraints are "true". Figure 2.4 depicts the two-dimensional case where constraints are lines. The shaded side of a line is where the associated constraint is met.

The intersection of all half-spaces is the set of points which satisfy the constraints. It is called the *feasible* region (Fig. 2.4(a)). Points inside this region are called *feasible*. If the feasible region is empty, the LP is said to be *infeasible* (Fig. 2.4(b)). If there exists exactly one solution that minimizes the objective, we call the solution *unique optimal* (Fig. 2.5(a)). It could be that there are several equally good, and therefore *alternative optimal* solutions (Fig. 2.5(b)). Finally, we call the LP *unbounded* (Fig. 2.5(c)) if the feasible region is unbounded in a direction δ, such that the objective function takes on arbitrarily large values along δ.

2.2.2 Quadratic Programming

Finally, we formulate a QP with a quadratic objective function and linear constraints:

$$\begin{aligned} min \quad & \mathbf{c}^T\mathbf{x} + \tfrac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x} \\ s.\,t. \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b} \end{aligned} \tag{2.40}$$

Again, $\mathbf{c} \in R^n$, $\mathbf{b} \in R^m$, $\mathbf{x} \in R^n$, $\mathbf{A} \in R^{m \times n}$, and $\mathbf{Q} \in R^{n \times n}$ is symmetric. For $\mathbf{Q} = 0$, the quadratic program is reduced to a linear program. A significant number of applications in engineering and the physical sciences lead to a convex QP model. The latter means that the objective function is convex. The objective function is convex, if and only if $\mathbf{Q}$ is positive semi-definite (PSD). We say $\mathbf{Q}$ is PSD if $\mathbf{y}^T\mathbf{Q}\mathbf{y} \geq 0$ for all $\mathbf{y} \in R^n$ [64]. QPs with convex objective can be solved in polynomial time [33].

Chapter 3

Collision Detection

It is an important task during the simulation loop (Fig. 1.1) to detect collisions and to respond to them appropriately. As the body moves along its trajectory, integrating the equations of motion (Sec. 2.1.6) generates the necessary information for rendering the scene. Allowing the bodies to follow their trajectories will cause them to collide. We have to provide effective mechanisms to detect collisions. It seems quite natural that the distance between two bodies is a good indicator for this purpose. If the distance between two bodies at any given moment of the simulation becomes negative, we declare that a collision has happened. At this point, the simulation has to stop and collision has to be resolved. It has been also described as practical to simply find overlap without any distance measure [63]. If any vertex of body **A** is found to lie within body **B**, or vice versa, collision is declared.

In this section, we will examine methods for distance calculation. This topic is well researched due to its importance for applications in robotics. Computational geometry has devised practical algorithms for this purpose. Motion and path planning are important areas of study. Algorithms have been made practical for simulation purposes and libraries,

such as V-Clip [61], I-COLLIDE [26], and SWIFT [30] are available. In some cases it will be important to know the exact distance between bodies. Gilbert *et al.* [36] devised an algorithm, which was extended by Cameron [19] to *constant* time by using *hill climbing*. Lin and Canny devised a very interesting algorithm for closest feature tracking, which is a standard in many applications [50, 51]. Our work uses our own implementation of an algorithm, which follows the Lin-Canny distance finding algorithm somewhat loosely.

Collision detection is usually performed in stages. A cheaper mechanism prunes pairs of bodies and exact collision detection is used where it cannot be avoided. Bounding boxes [81], bounding spheres [41] and hierarchical hash tables [59] are used to quickly find pairs, which are definitely non-overlapping. Suri *et al.* [81] have actually proven that bounding boxes are extremely efficient and not, as assumed before, of poor performance in the worst case. They have demonstrated that algorithms that use bounding boxes, are often of linear time complexity in the number of actual overlaps between bodies if the compared bodies are of similar size and not very elongated. Furthermore, it has been shown extremely useful for purposes of efficiency to exploit *coherence* [3, 19, 50]. Coherence means that the relative positions of bodies will not change drastically between time-steps. Therefore, the current distance can be calculated using some knowledge from the previous time-step. This is usually described by the use of *witnesses*.

Due to its discrete sampling nature it is difficult in simulation to *always* detect collisions. It could happen that a very fast flying body passes completely through a paper-thin wall without being noticed by the simulator. This could be correctly avoided by similar collision detection algorithms [20, 63] which check the four-dimensional hyper-polyhedra, as generated by time-swept motion. For many applications this is not necessary and too expensive. However, simply ignoring bodies that pass through each other is not a good idea

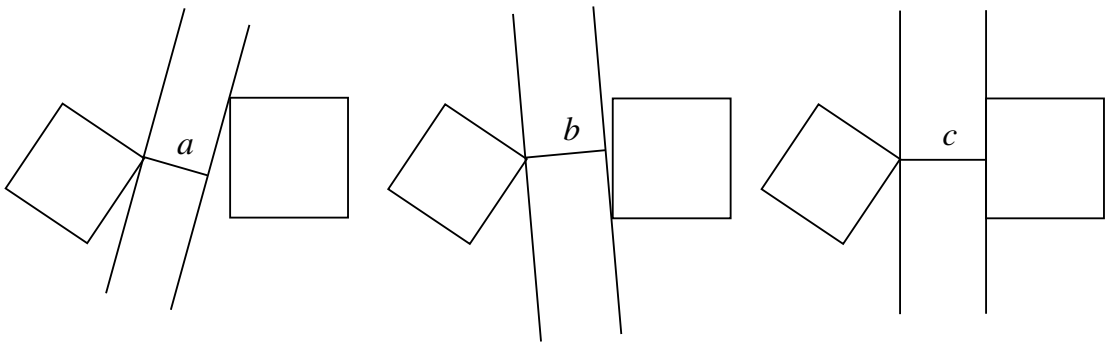


Figure 3.1: "Wiggle" the pair of planes that separate the bodies: from the three different pairs of planes, the correct pair maximizes the distance, $a < b < c$.

either. Mirtich introduces the idea of conservative advancement to address the problem [59].

3.1 Distance Calculation

All our reasoning here is based on the assumption of rigid and polyhedral convex bodies. Some of our simulations contain spheres. Spheres are certainly convex and can be approximated by multi-faceted polyhedra. However, it turns out that the distance calculation for exact spheres is much simpler than for polyhedra. A brief discussion can be found in Appendix A.2. We define the distance between two bodies $\mathbf{A}$ and $\mathbf{B}$ to be the maximum separation between the two. We will reference a feature on body $\mathbf{A}$ by subscript a and similarly for body $\mathbf{B}$ by subscript b. The closest segment connects closest points on the bodies $\mathbf{q}_a \in \mathbf{A}$ and $\mathbf{q}_b \in \mathbf{B}$. The body features which contain the closest points are called *closest features*. In particular, the two closest points will define a normal direction $\mathbf{n}$, which we call the *collision normal* or also *contact normal*. By definition and for reasons that will become clear later when we introduce the OBA algorithm, we assume that $\mathbf{n}$ always points from $\mathbf{A}$ into $\mathbf{B}$. If we knew the collision normal $\mathbf{n}$, we could derive the closest points and vice versa. Our approach is to iterate over a closed set of possible normals $\mathbf{n}$ and choose

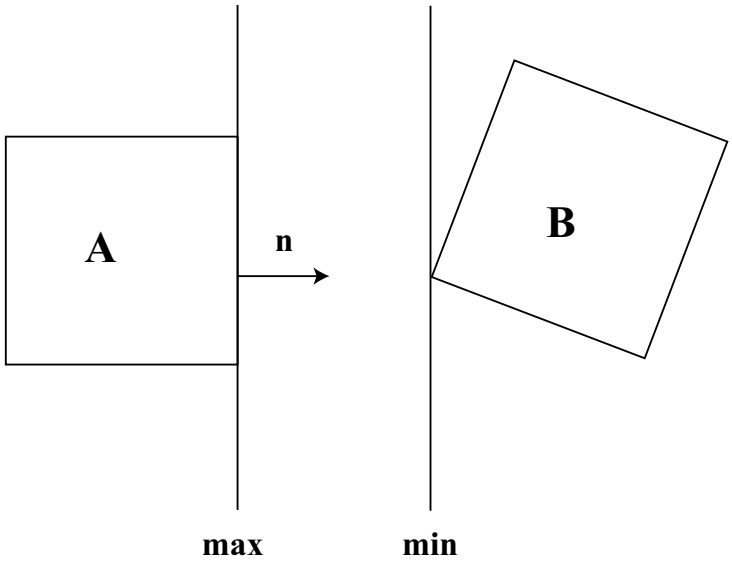


Figure 3.2: The normal **n** points from body **A** to body **B**: to calculate the distance as a difference of extreme features. In the direction of **n**, take the maximum and minimum vertex on **A** and **B** respectively.

the one that maximizes the distance between the bodies. It is somewhat counter-intuitive at first to maximize the distance. This will be clearer by looking at Figure 3.1.

For a given **n**, we calculate the distance D between two bodies **A** and **B** as follows (Fig. 3.2):

$$D = \left(\min_{\mathbf{q}_b \in B} \mathbf{n} \cdot \mathbf{q}_b - \max_{\mathbf{q}_a \in A} \mathbf{n} \cdot \mathbf{q}_a \right). \tag{3.1}$$

Of all possible normals **n**, we will then pick the one which maximizes the distance D of Equation 3.1. The question remains how we generate the set of possible vectors **n**. The answer is given by geometry and a closer look at the *features* of a polyhedral body, *i.e.* its vertices, edges, and faces. For the relative position of two bodies, we can distinguish four cases of opposing features. We have the cases *vertex-vertex*, *vertex-edge*, *vertex-face*, and *edge-edge*. Cases *edge-vertex* and *face-vertex* are included by symmetry, and cases *edge-face* and *face-face* are covered by *vertex-face*. We iterate over all possible combinations and can so derive the set of normals **n** from these cases by simple geometry. It is important to

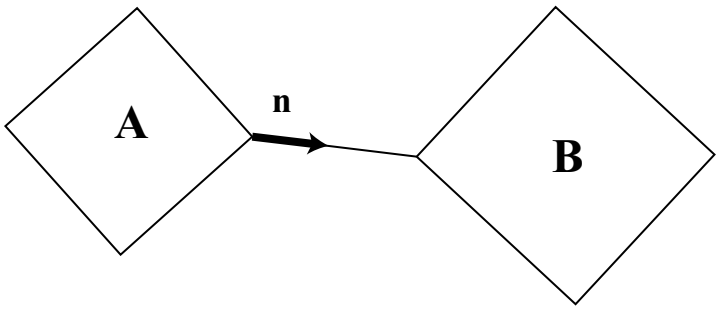


Figure 3.3: The simplest case: **n** is given by the connection of two vertices.

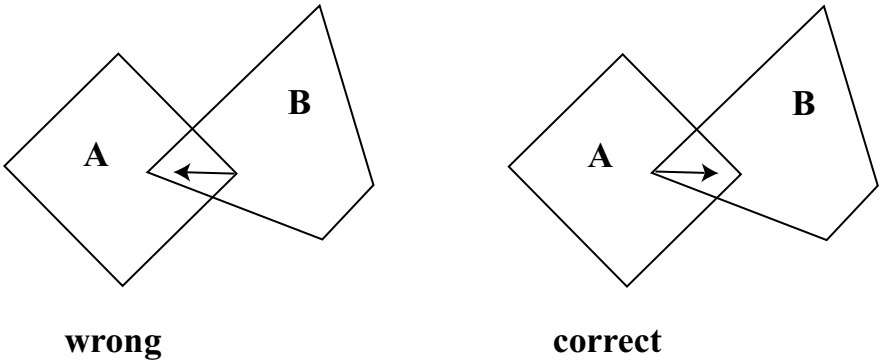


Figure 3.4: Only one direction for the normal is correct: pointing out of body **A**.

keep in mind that this normal has to point from body **A** to body **B** in order to maximize Equation 3.1. We will now discuss the four cases in detail.

3.1.1 Vertex-Vertex

In the case where the closest features between two bodies are vertices, we compute **n** simply by normalizing the connecting vector between the vertices. Normalization is denoted by the subscripted symbol "$\circ$". Assume bodies **A** and **B** with closest points $\mathbf{q}_a \in \mathbf{A}$ and $\mathbf{q}_b \in \mathbf{B}$ (Fig. 3.3):

$$\mathbf{n} = (\mathbf{q}_b - \mathbf{q}_a)^\circ. \tag{3.2}$$

We have to explicitly try both directions, **n** and $-\mathbf{n}$, to ensure we find the proper minimum of Equation 3.1. When two bodies become overlapping, having **n** point from $\mathbf{q}_a$ to $\mathbf{q}_b$ would

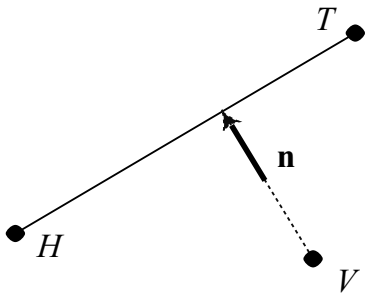


Figure 3.5: Deriving the normal **n** for the case *vertex-edge*.

not generate the desired normal **n** that points *out* of **A** (Fig. 3.4).

3.1.2 Vertex-Edge

We derive the normal **n** by projection of the vertex **V** onto the edge **e** with endpoints **H** and **T** (Fig. 3.5):

$$\mathbf{n} = \left(\overline{VH} + \left(\frac{\overline{HT} \cdot \overline{HV}}{\overline{HT} \cdot \overline{HT}} \right) \overline{HT} \right)^{\circ}. \tag{3.3}$$

We also need to try both directions of **n**.

3.1.3 Vertex-Face

The normal here is given simply by the face normal:

$$\mathbf{n} = \mathbf{n}_f. \tag{3.4}$$

If $\mathbf{n}_f \in \mathbf{A}$, we will actually pick **n** according to Equation 3.4, because we know it points out of body **A** as required. Otherwise, we need to negate and choose $-\mathbf{n}$.

3.1.4 Edge-Edge

We compute the normal **n** as the cross-product of the two edges:

$$\mathbf{n} = (\mathbf{e}_a \times \mathbf{e}_b)^{\circ}. \tag{3.5}$$

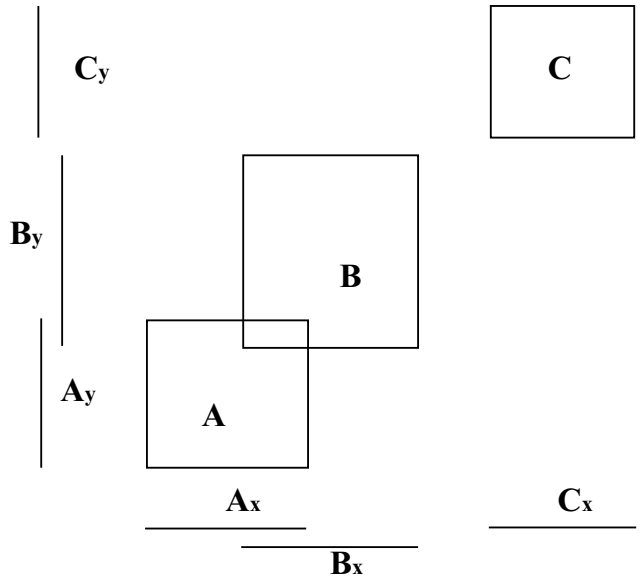


Figure 3.6: Comparing the lists immediately reveals the overlap status of the three boxes. The extent of each box along a coordinate axis is stored as an interval in a list.

In this last case, we also have to try both directions of **n** in order to find the one which properly minimizes Equation 3.1.

3.1.5 Pruning Pairs

Our approach as described above is to iterate over all possible normal vectors as derived from four cases and pick the one that maximizes Equation 3.1. For m features on each body, each distance calculation is $O(m^2)$. We do this for all possible pairs. For n bodies, we have to deal with n^2 pairs. This will work, but it clearly is an $O(m^2n^2)$ operation, and therefore expensive if we apply it at every time-step. It is therefore useful to employ filtering techniques that find definitely non-overlapping pairs before an exact algorithm has to be used on the ones where it cannot be avoided. A simple method for this is to employ bounding boxes or bounding spheres.

Bounding boxes are a versatile tool also for various other graphics applications, such

as visible surface determination and clipping. It is too expensive to check all n^2 possible pairs of bounding boxes. To do this check more efficiently, we sort the bounding boxes of all bodies based on the three coordinates axes into three lists. See Figure 3.6 for a 2-dimensional example. The dimension of each box is stored as three intervals. Each interval has entries for the begin and end of the box along the axis at hand. During the simulation, we re-sort the three lists each time the bodies have moved to their new positions.

During sorting, we determine intervals that become disjoint or overlapping. If a pair of bodies becomes disjoint in any list, we know this must be the case for the whole box. If a pair becomes overlapping in any list, we must check if this is so for the other two lists also. Two boxes overlap if and only if they overlap along all three coordinates.

It is convenient and efficient to maintain a hash table of box pairs in order to allocate particular interval pairs quickly when we need to make comparisons. To further speed up the process of bounding box comparisons, we set a flag to determine if a pair has already been compared. Say a pair of bounding boxes has already been checked as a result of comparison in x-coordinates, we need no longer do the same when we re-sort the y-list and z-list.

3.1.6 Coherence

We now have a method that allows a more efficient distance calculation and overlap detection after introducing bounding boxes for pruning of pairs. A further insight into the nature of animation can speed up the distance calculation even more drastically. Baraff described the concept of *coherence* [3]. Coherence means that the geometric relation between bodies will not change too drastically between frames due to the small size of the time-step. Baraff states that for a pair of bodies, a separating plane can be found based on

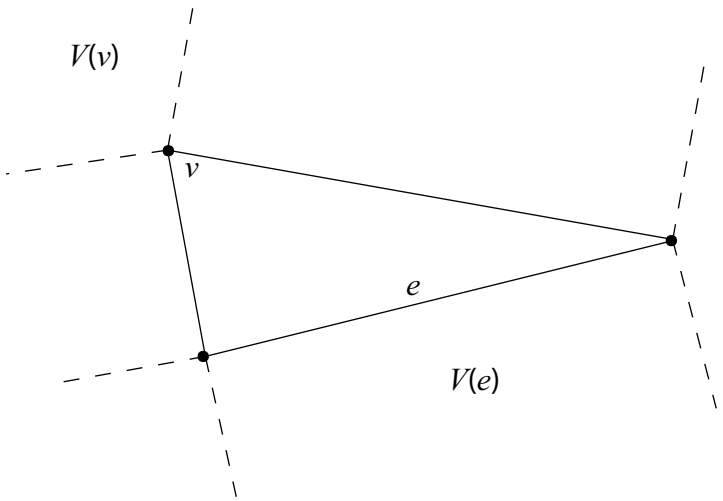


Figure 3.7: Voronoi regions: extend rays perpendicular to the edges at the polygon vertices.

the observation that the geometry of collision between polyhedra includes at least a face of one of them or an edge of each. The separating plane is then given by the face or the cross product of the two edges. If two bodies are interpenetrating, it is "almost always" [3] the case that a vertex of one body is inside the other or an edge has interpenetrated one of the other body's faces. In other words, a body's vertex or edge does not lie on the proper side of the separating plane. One can therefore cache the vertex or edge, and the separating plane as witnesses for subsequent time-steps and use them for checking quickly for interpenetration. This algorithm is somewhat *ad hoc* and it cannot be assured that it always terminates with the correct answer.

For a full and thorough exploitation of coherence, the Lin-Canny algorithm is much more exact and refined [50]. Our algorithm employs some of the concepts of Lin-Canny. It makes use of the observed smallness of change between frames. Its underlying concept is the use of *Voronoi* regions, which are well known from computational geometry. It is easiest to explain Voronoi regions with 2-dimensional polygons. See Figure 3.7 for a simple example with a triangle. The Voronoi regions are derived by drawing rays perpendicular

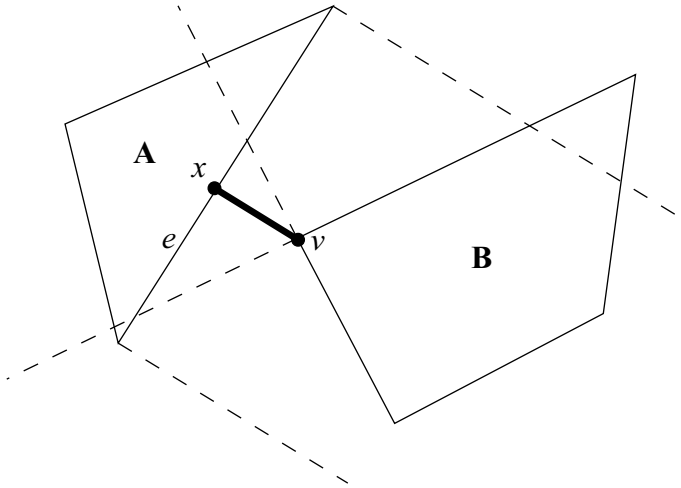


Figure 3.8: We have the case $v - e$: the closest points between two polygons **A** and **B** are $x \in e$ and v respectively. It must be that $x \in V(v)$ and $v \in V(e)$.

to the edges at their endpoints. The Voronoi region of an edge $V(e)$ is enclosed by the parallel rays emanating from the edge and the edge itself. Similarly, the Voronoi region of a vertex, $V(v)$ is enclosed by the wedge that is formed by the two rays that emanate from the vertex.

For three-dimensional bodies, the Voronoi regions V are volumes, which extend over the body features Ω. These regions are denoted by $V(\Omega)$. The interesting property of a Voronoi region is that any point $\mathbf{x} \in V(\Omega)$ is closer to Ω than to any other feature on the body. Assume bodies **A** and **B** with features Ω_a on **A** and Ω_b on **B**, and points $\mathbf{q}_a \in \Omega_a$ and $\mathbf{q}_b \in \Omega_b$. It can be shown [51, 60] that for non-intersecting polyhedra **A** and **B**, if $\mathbf{q}_a$ and $\mathbf{q}_b$ are the closest points between the features Ω_a and Ω_b respectively, $\mathbf{q}_a$ and $\mathbf{q}_b$ are closest points between **A** and **B**, if $\mathbf{q}_a \in V(\Omega_b)$ and $\mathbf{q}_b \in V(\Omega_a)$. See Figure 3.8 for an example.

This knowledge can be fully employed for finding the closest points between polyhedra under exploitation of coherence. We cache the features that form $\mathbf{n}$ and also the closest body features Ω_a and Ω_b in the current time-step as witnesses. Instead of recalculating $\mathbf{n}$

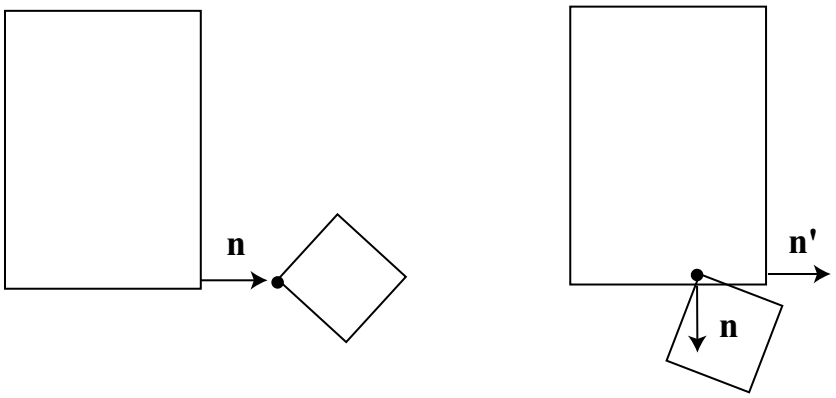


Figure 3.9: For overlapping bodies, we cannot simply use the cached witnesses. Using $\mathbf{n}'$ will report deep interpenetration while the actual value, when computed with the correct $\mathbf{n}$, is much smaller.

in the next step from scratch, we calculate a temporary new $\mathbf{n}'$ from the cached witness features at their new orientations. In the direction of $\mathbf{n}'$ we find closest points $\mathbf{q}_a$ and $\mathbf{q}_b$ on the previously closest features Ω_a and Ω_b at their current position and orientation. If $\mathbf{q}_a \in \Omega_a$ and $\mathbf{q}_b \in \Omega_b$, we are done. Otherwise, we currently use exhaustive search to find the new closest features and the new normal. Properly and efficiently, this has to be done by hill climbing, *i.e.* if the cached features are no longer the closest, we go to their neighbors and find the new pair of closest features in an efficient manner. The additional cost through starting all over if the cached witnesses need to be updated is negligible only for simple geometries as in our case, *i.e.* cubes and spheres. We also use exhaustive search if the distance between bodies becomes negative, *i.e.* bodies are overlapping.

The above assumption for Voronoi regions, as used in Lin-Canny, breaks when bodies overlap. It could be that the cached features produce closest points which lie in each other feature's Voronoi region as required, but they do not give the actual distance. See Figure 3.9 for an example where the cached witnesses will not give the right normal and distance. On the left, $\mathbf{n}$ is correctly found to be an outward pointing face normal. On the right,

the bodies have become overlapping. The interpenetrating vertex is still projected onto the cached face. However, if we accept $\mathbf{n}'$ as the new normal, the calculated distance will be wrong. We currently compute $\mathbf{n}$ and the distance from scratch whenever two bodies interpenetrate.

3.2 Determining Contacts

After finding the collision normal $\mathbf{n}$, it is easy to determine contacts for touching bodies. In contrast to Baraff's separating planes [3], our distance calculation finds the actual plane located at the geometric center between the bodies according to:

$$d = \frac{1}{2}\left(\max_{\mathbf{q}_a \in A} \mathbf{n} \cdot \mathbf{q}_a + \min_{\mathbf{q}_b \in B} \mathbf{n} \cdot \mathbf{q}_b\right). \tag{3.6}$$

We have now a separating plane with equation $\mathbf{n} \cdot \mathbf{x} = d$. Next, we find extreme vertices on the two bodies in the direction of $\mathbf{n}$, *i.e.* vertices which *touch* the separating plane. After finding all extreme vertices, we determine which feature they form on body $\mathbf{A}$ or $\mathbf{B}$, *i.e.* is it a vertex, edge or face. The bodies touch with these extreme features, and the contact geometry can be determined by examining the intersection. The intersection is a point, line segment, or convex polygon. That point, the endpoints of the line segment, or the vertices of the polygon are the contact points. Under this definition, two cubes can have up to eight contact points if one is rotated and stacked on top of each other (Fig. 3.10).

In practice, a tolerance must be used to determine if a point "touches" the separating plane. These points must be projected onto the separating plane, and the projections are then intersected to determine the contact geometry. Since $\mathbf{n}$ points from $\mathbf{A}$ to $\mathbf{B}$, all vertices that lie within a slab of the maximum on $\mathbf{A}$ ($\max_{\mathbf{q}_a \in A} \mathbf{n} \cdot \mathbf{q}_a$) or lie within a slab of the minimum on $\mathbf{B}$ ($\min_{\mathbf{q}_b \in B} \mathbf{n} \cdot \mathbf{q}_b$) can be viewed as touching. **Note**: to find the extreme

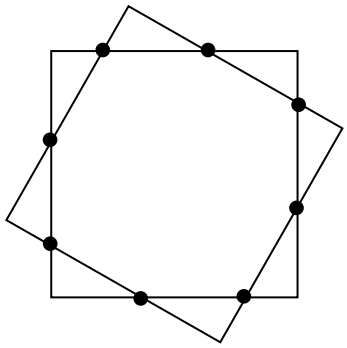


Figure 3.10: Two stacked cubes can generate an octagonal intersection, *i.e.* the cubes yield eight contact points, the vertices of the octagon.

point of a sphere, project the sphere's center onto the separating plane. This point is then treated as if it was a vertex.

Chapter 4

Collision and Contact Response

We have introduced how to determine collision and presented a method to find contacts. To respond to collisions and resolve them, methods to update body velocities and accelerations have to be devised. It is important to know how velocity and acceleration of contact points are determined. We start with a brief explanation of these properties. **Note**: the methods described here are valid for any convex body geometry, *i.e.* also spheres.

4.1 Contact Velocity and Acceleration

Intuitively, as introduced in Sections 2.1.4 and 2.1.5, we expect the velocity and acceleration of a point on a rigid body to be the first and second time derivative of its position respectively. We introduce an *infinitesimal* rotation (Fig. 4.1):

$$d\mathbf{w} = d\boldsymbol{\varphi} \times \mathbf{w}. \tag{4.1}$$

This is the linearized change of a particle's position, which rotates a small angle. Note that $\boldsymbol{\varphi}$ is a vector. Its direction gives the rotation axis and its magnitude the angle of rotation

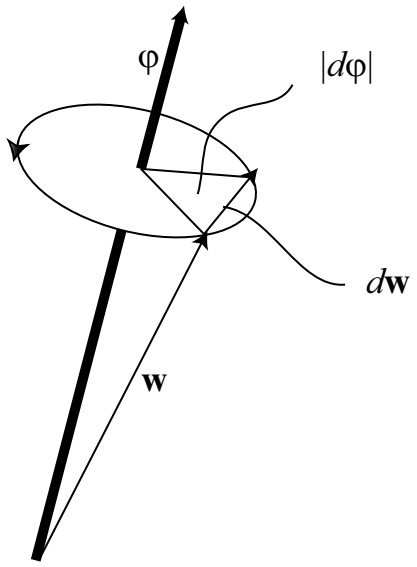


Figure 4.1: The linearized change $d\mathbf{w}$ of a vector $\mathbf{w}$ that is rotated a small amount $d\boldsymbol{\varphi}$ is $d\mathbf{w} = d\boldsymbol{\varphi} \times \mathbf{w}$.

around this axis. If the duration of the rotation is dt, we yield together with Equation 4.1:

$$\frac{d\mathbf{w}}{dt} = \frac{d\boldsymbol{\varphi}}{dt} \times \mathbf{w} = \boldsymbol{\omega} \times \mathbf{w}. \tag{4.2}$$

This is the velocity of the particle as a result of rotation. This rotating system can itself move with a linear velocity $\mathbf{v}_0$. The net particle velocity is therefore the sum:

$$\mathbf{v} = \mathbf{v}_0 + \boldsymbol{\omega} \times \mathbf{w}. \tag{4.3}$$

We arrive at the acceleration by differentiating Equation 4.3:

$$\mathbf{a} = \dot{\mathbf{v}}_0 + \dot{\boldsymbol{\omega}} \times \mathbf{w} + \boldsymbol{\omega} \times \dot{\mathbf{w}}. \tag{4.4}$$

The last term can be broken down further. The change of $\mathbf{w}$, that is $\dot{\mathbf{w}}$, is expressed by Equation 4.2, and the linear acceleration of the center of mass is denoted by $\mathbf{a}_0$:

$$\mathbf{a} = \mathbf{a}_0 + \dot{\boldsymbol{\omega}} \times \mathbf{w} + \boldsymbol{\omega} \times (\boldsymbol{\omega} \times \mathbf{w}). \tag{4.5}$$

We see that the point acceleration is the sum of the linear acceleration, a second term for the *tangential* point acceleration and the *centripetal* acceleration. The tangential accel-

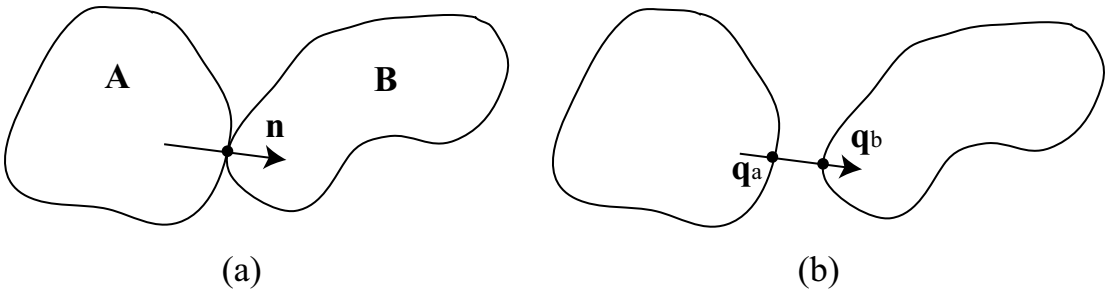


Figure 4.2: Separation distance $D = \mathbf{n} \cdot (\mathbf{q}_b - \mathbf{q}_a)$: $D = 0$ for contacting bodies.

eration is a result of the particle being angularly accelerated by $\dot{\boldsymbol{\omega}}$ with displacement $\mathbf{w}$. The centripetal acceleration stems from the fact that any rotation is actually accelerated. Although the velocity *magnitude* may be constant, it continually changes its *direction*. According to Newton's second axiom (Eq. 2.18) there is a centripetal force connected to the centripetal acceleration. The centripetal force constrains a rotating body to stay on its circular path by attracting it radially to the center of rotation. At the same time, according to the *actio = reactio* principle, there is an outward pulling force. We can experience this in many everyday situations, for example when going around a curve in a car.

Assume now two contacting bodies **A** and **B** (Fig. 4.2). Call the contact points $\mathbf{q}_a$ and $\mathbf{q}_b$ respectively, and call the contact normal $\mathbf{n}$. We can write for the distance or separation D between the bodies at the contact:

$$D = \mathbf{n} \cdot (\mathbf{q}_b - \mathbf{q}_a). \tag{4.6}$$

Of course, $D = 0$ for contacting bodies, since $\mathbf{q}_a = \mathbf{q}_b$ (Fig. 4.2(a)). However, we are interested in changing states. Hence, when separation happens, D gives the accurate separation distance (Fig. 4.2(b)). We will use this knowledge to derive expressions for relative velocity and acceleration at a contact.

4.1.1 Relative Contact Velocity

Analogous to our previous reasoning in Section 2.1.4, the relative contact normal velocity $v^{\perp}$ (scalar) is the first derivative of Equation 4.6 with respect to time:

$$v^{\perp} = \dot{\mathbf{n}} \cdot (\mathbf{q}_b - \mathbf{q}_a) + \mathbf{n} \cdot (\dot{\mathbf{q}}_b - \dot{\mathbf{q}}_a) = \mathbf{n} \cdot (\dot{\mathbf{q}}_b - \dot{\mathbf{q}}_a). \tag{4.7}$$

The latter equality is due to $\mathbf{q}_a = \mathbf{q}_b$ by definition for contacting bodies. The property $v^{\perp}$ gives the separation velocity at the contact point. If $v^{\perp} < 0$, we declare interpenetration, since the bodies are moving into each other at the contact. We always want to ensure $v^{\perp} \geq 0$.

4.1.2 Relative Contact Acceleration

We derive the relative contact normal acceleration $a^{\perp}$ by differentiating Equation 4.7 once again with respect to time:

$$\begin{aligned} a^{\perp} &= \ddot{\mathbf{n}} \cdot (\mathbf{q}_b - \mathbf{q}_a) + \dot{\mathbf{n}} \cdot (\dot{\mathbf{q}}_b - \dot{\mathbf{q}}_a) + \\ &\quad \dot{\mathbf{n}} \cdot (\dot{\mathbf{q}}_b - \dot{\mathbf{q}}_a) + \mathbf{n} \cdot (\ddot{\mathbf{q}}_b - \ddot{\mathbf{q}}_a) = \\ &= \mathbf{n} \cdot (\ddot{\mathbf{q}}_b - \ddot{\mathbf{q}}_a) + 2\dot{\mathbf{n}} \cdot (\dot{\mathbf{q}}_b - \dot{\mathbf{q}}_a). \end{aligned} \tag{4.8}$$

We use again the equality $\mathbf{q}_a = \mathbf{q}_b$ to do the final step in the above equation. The quantity $a^{\perp}$ is a measure for the relative contact normal acceleration. Assume for now that $v^{\perp} = 0$, *i.e.* there is no relative motion between the bodies. If $a^{\perp} < 0$, the bodies are accelerating into each other. This case must be avoided, because it means that the bodies will interpenetrate in the future. When $a^{\perp} > 0$, the bodies have an acceleration away from each other. We say the contact breaks. The final case where $a^{\perp} = 0$ indicates static contact, which is also called persistent or resting contact.

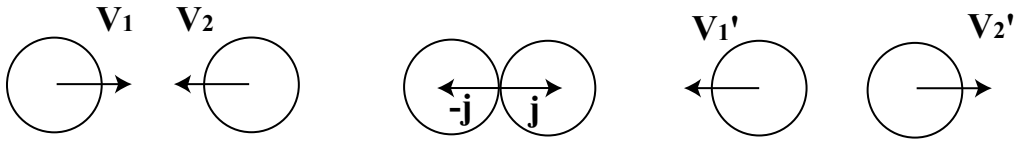


Figure 4.3: When two bodies collide, an impulse has to make them instantaneously receding.

4.2 Analytic Impulses and Forces

We will now introduce the tools for properly updating the body velocities and accelerations if there are collisions and contacts. For readability, we drop the $\perp$ symbol but introduce subscripts - and + to denote velocities and accelerations before and after impulses or forces were applied respectively.

We follow the literature [7] by defining a collision as a contact with $v^- < 0$. First, we will calculate impulses that resolve collisions. Impulses are applied to the bodies at a contact point in order to instantaneously make the bodies receding, $v^+ \geq 0$ (Fig. 4.3). Contacts with $v^+ = 0$ are static contacts. Static contacts have forces applied, which act over time and ensure $a^+ \geq 0$ (Fig. 4.4).

4.2.1 Impulses

There are two empirical collision models [80]. The *Poisson* model divides the impact of two bodies into a compression and decompression phase and relates them by a coefficient of restitution ϵ. We will use Newton's model in our work [17]. It establishes a relation between the relative contact normal velocities before and after the collision:

$$v^+ = -\epsilon v^-. \tag{4.9}$$

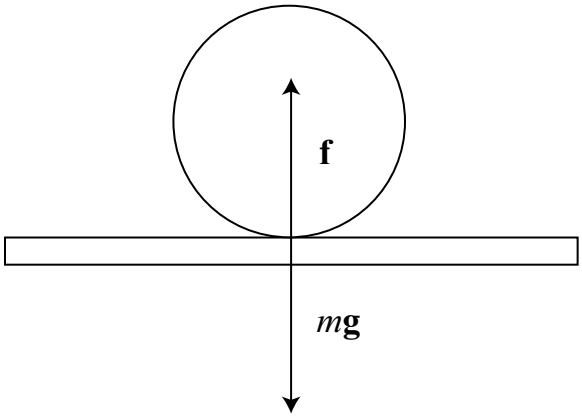


Figure 4.4: To prevent the ball from sinking into the table, a static contact force has to counteract gravity.

In principle, it states that a part of the colliding momenta has to be reflected. The coefficient of restitution $\epsilon \in [0..1]$ determines how elastic the collision is [55, 78]. For $\epsilon = 0$, we have a perfectly inelastic collision, *i.e.* all momentum is dissipated. On the other hand, $\epsilon = 1$ gives a perfectly elastic collision. We can view ϵ for our needs as a material property.

The impulse $\mathbf{j}$ is applied equally but opposite at a contact to both bodies. This is a direct result from the conservation of momenta, $-\mathbf{j} + \mathbf{j} = 0$. The direction of the impulse coincides with the collision normal, that is, the impulse acts perpendicular to the colliding bodies' surfaces. The only unknown is therefore the impulse magnitude. For simplicity, we will derive the mechanics of impulse without friction. Friction is an easy addition later after we have the framework of our equations worked out.

We start with a formal introduction of the impulse $\mathbf{j}$:

$$\mathbf{j} = \mathbf{f}\Delta t. \tag{4.10}$$

A force that acts for a time interval Δt changes the momentum of a body. We assume here $\Delta t \to 0$, *i.e.* the force acts for an infinitely small time. We call this an impulsive force or simply an impulse. Following Equation 2.18, we can see that the change of momentum is

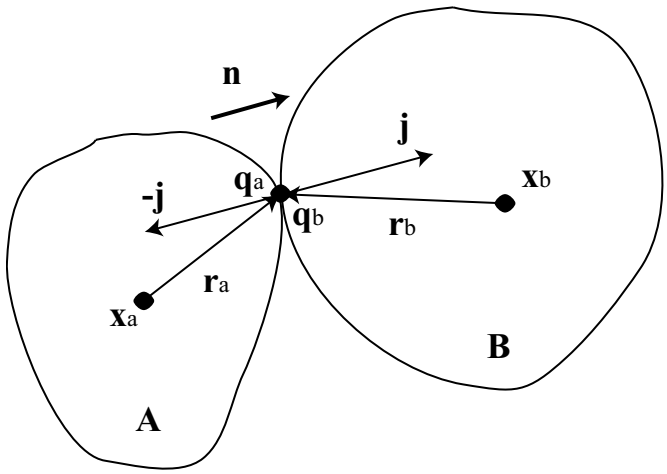


Figure 4.5: The contact geometry of two colliding bodies.

$\Delta\mathbf{p} = \mathbf{j}$, $\mathbf{p}_{new} = \mathbf{p} + \mathbf{j}$. We yield for the change of linear velocity for a body with mass m:

$$\Delta\mathbf{v} = \frac{\mathbf{j}}{m}. \tag{4.11}$$

If the impulse acts with lever $\mathbf{r}$ it produces an impulsive torque $\mathbf{r} \times \mathbf{j}$, yielding the changed angular momentum $\boldsymbol{\ell}_{new} = \boldsymbol{\ell} + \mathbf{r} \times \mathbf{j}$. We can calculate the change in angular velocity for a body with inertia $\mathbf{I}$:

$$\Delta\boldsymbol{\omega} = \mathbf{I}^{-1}(\mathbf{r} \times \mathbf{j}). \tag{4.12}$$

Consider the scenario of Figure 4.5. We have two bodies $\mathbf{A}$ and $\mathbf{B}$, and contacts $\mathbf{q}_a$ and $\mathbf{q}_b$ respectively. The levers are calculated by subtracting the center of mass coordinate from the contact, *i.e.* $\mathbf{r}_a = \mathbf{q}_a - \mathbf{x}_a$ and $\mathbf{r}_b = \mathbf{q}_b - \mathbf{x}_b$. An impulse acts in the direction of the collision normal. Since we know that the collision normal $\mathbf{n}$ points from body $\mathbf{A}$ to body $\mathbf{B}$, we also know that the impulse $\mathbf{j}$ acts positively on body $\mathbf{B}$ and negatively on body $\mathbf{A}$. We are looking for the impulse magnitude j. Therefore, let $-j\mathbf{n}$ be the impulse on body $\mathbf{A}$ and $j\mathbf{n}$ the impulse on body $\mathbf{B}$. If $\dot{\mathbf{q}}_a^-$ was the velocity of point $\mathbf{q}_a \in \mathbf{A}$ before

an impulse was applied, we can calculate the velocity after the impulse $-j\mathbf{n}$ according to:

$$\begin{aligned}
\dot{\mathbf{q}}_a^+ &= \mathbf{v}_a^- + \frac{-j\mathbf{n}}{m_a} + (\boldsymbol{\omega}_a^- + \mathbf{I}_a^{-1}(\mathbf{r}_a \times (-j\mathbf{n}))) \times \mathbf{r}_a = \\
&= \mathbf{v}_a^- + \boldsymbol{\omega}_a^- \times \mathbf{r}_a - j\left(\frac{\mathbf{n}}{m_a} + \mathbf{I}_a^{-1}(\mathbf{r}_a \times \mathbf{n}) \times \mathbf{r}_a\right) = \\
&= \dot{\mathbf{q}}_a^- - j\left(\frac{\mathbf{n}}{m_a} + \mathbf{I}_a^{-1}(\mathbf{r}_a \times \mathbf{n}) \times \mathbf{r}_a\right). \qquad (4.13)
\end{aligned}$$

Note that we have written down a relation for $\mathbf{q}_a$'s velocity before and after the impulse as a linear combination of known properties and our only unknown, the impulse magnitude j. For body $\mathbf{B}$, which has $j\mathbf{n}$ acting on it, we have symmetrically:

$$\dot{\mathbf{q}}_b^+ = \dot{\mathbf{q}}_b^- + j\left(\frac{\mathbf{n}}{m_b} + \mathbf{I}_b^{-1}(\mathbf{r}_b \times \mathbf{n}) \times \mathbf{r}_b\right). \qquad (4.14)$$

Following Equation 4.7 for the relative contact velocity, we write:

$$\begin{aligned}
v^+ &= \mathbf{n} \cdot (\dot{\mathbf{q}}_b^+ - \dot{\mathbf{q}}_a^+) = \\
&= \mathbf{n} \cdot (\dot{\mathbf{q}}_b^- - \dot{\mathbf{q}}_a^-) + \\
&\quad j\mathbf{n} \cdot \left(\frac{\mathbf{n}}{m_b} + \frac{\mathbf{n}}{m_a} + (\mathbf{I}_b^{-1}(\mathbf{r}_b \times \mathbf{n})) \times \mathbf{r}_b + (\mathbf{I}_a^{-1}(\mathbf{r}_a \times \mathbf{n})) \times \mathbf{r}_a\right) = \\
&= v^- + j\left(\frac{1}{m_b} + \frac{1}{m_a} + \mathbf{n} \cdot (\mathbf{I}_b^{-1}(\mathbf{r}_b \times \mathbf{n})) \times \mathbf{r}_b + \mathbf{n} \cdot (\mathbf{I}_a^{-1}(\mathbf{r}_a \times \mathbf{n})) \times \mathbf{r}_a\right) = \\
&= -\epsilon v^-. \qquad (4.15)
\end{aligned}$$

We apply Newton's collision model from Equation 4.9 when writing the last equality of above equation. In the final step, we solve for j:

$$j = \frac{-(1+\epsilon)v^-}{m_b^{-1} + m_a^{-1} + \mathbf{n} \cdot (\mathbf{I}_b^{-1}(\mathbf{r}_b \times \mathbf{n})) \times \mathbf{r}_b + \mathbf{n} \cdot (\mathbf{I}_a^{-1}(\mathbf{r}_a \times \mathbf{n})) \times \mathbf{r}_a}. \qquad (4.16)$$

All body properties in Equation 4.16 are known, and we can easily solve for the impulse magnitude j. We know in any case that $j \geq 0$. This means the impulse always pushes the bodies apart; it never pulls the bodies together.

4.2.2 Forces

We will again develop our mechanics without friction in this section. Assume that impulses have been applied at collisions, and we have therefore established $v^+ \geq 0$ at all collisions. The case where $v^+ > 0$ is not of any concern, because it signals separation at the contact. The interesting case is where $v^+ = 0$, *i.e.* static contact has formed. We need to ensure $a^+ \geq 0$ at all contacts by applying appropriate forces. In other words, the contacting bodies are to be prevented from accelerating into each other. Either the contact persists, $a^+ = 0$, or the bodies accelerate apart, $a^+ > 0$, *i.e.* they separate and the contact breaks.

Consider again bodies $\mathbf{A}$ and $\mathbf{B}$, contacts $\mathbf{q}_a$ and $\mathbf{q}_b$, and levers $\mathbf{r}_a = \mathbf{q}_a - \mathbf{x}_a$ and $\mathbf{r}_b = \mathbf{q}_b - \mathbf{x}_b$ respectively. The relative contact normal acceleration can be calculated following Equation 4.8, $a^\perp = \mathbf{n} \cdot (\ddot{\mathbf{q}}_b - \ddot{\mathbf{q}}_a) + 2\dot{\mathbf{n}} \cdot (\dot{\mathbf{q}}_b - \dot{\mathbf{q}}_a)$. Let $\mathbf{a}_a$ and $\mathbf{a}_b$ denote the linear accelerations of bodies $\mathbf{A}$ and $\mathbf{B}$ respectively. Together with Equation 4.5, the acceleration of a point, we can write:

$$\begin{aligned} a^\perp &= \mathbf{n} \cdot ((\mathbf{a}_b + \dot{\boldsymbol{\omega}}_b \times \mathbf{r}_b + \boldsymbol{\omega}_b \times (\boldsymbol{\omega}_b \times \mathbf{r}_b)) - \\ &\qquad (\mathbf{a}_a + \dot{\boldsymbol{\omega}}_a \times \mathbf{r}_a + \boldsymbol{\omega}_a \times (\boldsymbol{\omega}_a \times \mathbf{r}_a))) + \\ &\quad 2\dot{\mathbf{n}} \cdot (\dot{\mathbf{q}}_b - \dot{\mathbf{q}}_a) = \\ &= \mathbf{n} \cdot (\mathbf{a}_b + \dot{\boldsymbol{\omega}}_b \times \mathbf{r}_b - \mathbf{a}_a - \dot{\boldsymbol{\omega}}_a \times \mathbf{r}_a) + \\ &\quad \mathbf{n} \cdot (\boldsymbol{\omega}_b \times (\boldsymbol{\omega}_b \times \mathbf{r}_b) - \boldsymbol{\omega}_a \times (\boldsymbol{\omega}_a \times \mathbf{r}_a)) + \\ &\quad 2\dot{\mathbf{n}} \cdot (\dot{\mathbf{q}}_b - \dot{\mathbf{q}}_a). \end{aligned} \tag{4.17}$$

It depends on the linear and angular point accelerations and the velocities of the respective points on the contacting bodies. The body velocities are known after the impulses have been applied, and therefore we can calculate the second and third constant velocity-dependent terms immediately.

The body accelerations and point accelerations are a result of the sum of forces acting on the body. We assume external forces, which can include natural forces such as gravity, wind, electromagnetism, and buoyancy, and also internal forces, *i.e.* the actual contact forces.

For simplicity, we derive our equations for gravity as the only external force. Addition of other potentials is trivial. Denote the magnitude of the contact force at the i-th contact with f_i. Contact forces point again in normal direction $\mathbf{n}$ and they are conservative, *i.e.* $\mathbf{f}_i = -\mathbf{n}f_i$ for contacts on body $\mathbf{A}$ and $\mathbf{f}_i = \mathbf{n}f_i$ for contacts on body $\mathbf{B}$. We write for the total force on body $\mathbf{B}$:

$$\mathbf{F}_b = m_b\mathbf{g} + \sum_i (f_i\mathbf{n}). \tag{4.18}$$

The resulting linear acceleration of $\mathbf{B}$ is calculated with:

$$\mathbf{a}_b = \frac{\mathbf{F}_b}{m_b} = \mathbf{g} + \frac{\sum\limits_i (f_i\mathbf{n})}{m_b}. \tag{4.19}$$

Note that we can separate the external part from the contact force-dependent part. In the same fashion, we write the total torque on body $\mathbf{B}$ as a sum of external torque $\boldsymbol{\tau}_b$ and contact torques:

$$\mathbf{T}_b = \boldsymbol{\tau}_b + \sum_i \mathbf{r}_{bi} \times (f_i\mathbf{n}). \tag{4.20}$$

We calculate the angular acceleration of $\mathbf{B}$, keeping in mind that it can be also separated into an external part and a contact force-dependent part:

$$\boldsymbol{\alpha}_b \equiv \dot{\boldsymbol{\omega}}_b = \mathbf{I}_b^{-1}\mathbf{T}_b = \mathbf{I}_b^{-1}\boldsymbol{\tau}_b + \mathbf{I}_b^{-1}\sum_i \mathbf{r}_{bi} \times (f_i\mathbf{n}). \tag{4.21}$$

Analogously, we calculate the total force

$$\mathbf{F}_a = m_a\mathbf{g} - \sum_i (f_i\mathbf{n}), \tag{4.22}$$

and total torque

$$\mathbf{T}_a = \boldsymbol{\tau}_a - \sum_i \mathbf{r}_{ai} \times (f_i \mathbf{n}) \tag{4.23}$$

on body **A**, as well as the resulting linear acceleration

$$\mathbf{a}_a = \frac{\mathbf{F}_a}{m_a} = \mathbf{g} - \frac{\sum_i (f_i \mathbf{n})}{m_a}, \tag{4.24}$$

and angular acceleration

$$\boldsymbol{\alpha}_a \equiv \dot{\boldsymbol{\omega}}_a = \mathbf{I}_a^{-1} \mathbf{T}_a = \mathbf{I}_a^{-1} \boldsymbol{\tau}_a - \mathbf{I}_a^{-1} \sum_i \mathbf{r}_{ai} \times (f_i \mathbf{n}). \tag{4.25}$$

Intuitively, and similarly to our reasoning about impulses, $f_i \geq 0$, *i.e.* contact forces push the bodies apart, they never pull on them.

We have to find force magnitudes f_i which generate positive contact accelerations at all contacts. Denote with a_i^+ the relative contact normal acceleration at contact i after forces were applied at all contacts. We can setup a QP that implements constraints for the total force and torque on each body with Equations 4.18, 4.20, 4.22, and 4.23. The resulting body accelerations are provided by constraints according to Equations 4.19, 4.21, 4.24, and 4.25. Plugging the accelerations into Equation 4.17 yields a_i^+ for each contact as a result of external forces and internal contact forces f_i. The relative contact normal acceleration has to be positive, $a_i^+ \geq 0$. **Note**: we can separate the constant velocity-dependent terms from the contact force-dependent terms. Constant terms appear in the right-hand side bounds of all constraints.

The contact problem has been formulated as a *linear complementarity problem* (LCP) by Lötstedt [52]. Baraff has expressed this problem also as an LP [7]. The two formulations follow the same basic ideas. Let $\mathbf{a}$ be the vector of relative contact normal accelerations a_i^+ according to Equation 4.17, and $\mathbf{f}$ the vector of forces f_i for all contacts. Either the

acceleration or force at a contact has to be zero, and neither the acceleration, nor the force at a contact can be negative:

$$\begin{aligned} \mathbf{f}^T\mathbf{a} &= 0, \\ \mathbf{a} &\geq 0, \\ \mathbf{f} &\geq 0. \end{aligned} \tag{4.26}$$

Equation 4.26 expresses this by requiring the components of $\mathbf{f}$ to be positive, $f_i \geq 0$, and the same for the components of $\mathbf{a}$, $a_i^+ \geq 0$. Positive force is equivalent to non-adhesive force, and positive acceleration means that contacting bodies are accelerating relative to each other in a non-penetrating way. The equality in Equation 4.26 is the actual complementarity condition. It requires for each contact either a_i^+ or the contact force f_i to be zero (or both). If the bodies are accelerating apart at a contact, $a_i^+ > 0$, they are separating. We say the contact breaks, and the contact force f_i has to decrease instantaneously to zero. Otherwise, if actually a positive contact force acts, $f_i > 0$, the contact persists, and the acceleration a_i^+ must be zero. See Cottle *et al.* [27] for a thorough discussion of LCPs and solution methods.

The advantage of a QP-based formulation lies in the nature of its constraints. While Baraff writes down the contact acceleration directly as a result of internal contact forces and external forces [7], a QP derives the contact accelerations in a two-step process: calculate total force, torque, and the resulting accelerations for all bodies and use the latter to calculate the contact accelerations. Excess variables are automatically eliminated by the QP solver. Building the constraints is less cumbersome and less prone to error than in Baraff's work.

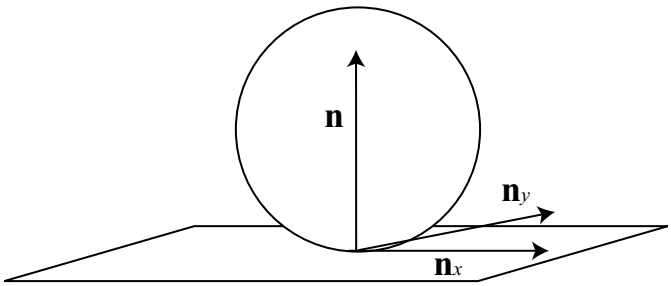


Figure 4.6: Collision coordinate frame with tangential vectors $\mathbf{n}_x$ and $\mathbf{n}_y$, and the normal $\mathbf{n}$ pointing along the z-axis.

4.3 Coulomb Friction

To model friction, we use the Coulomb friction model. This model exhibits some problems with unsolvable or ambiguous configurations. Nevertheless, it is widely used due to its simplicity and satisfactory performance for most applications [7, 13, 15, 21, 58, 59]. Coulomb friction simply relates the tangential, frictional force (or impulse [45]) to the force acting normal to the touching surfaces. It states that the tangential frictional component can be maximally a friction factor μ times the normal component, $|\mathbf{f}_t| \leq \mu|\mathbf{f}_n|$. Assume we have an orthonormal collision coordinate frame $(\mathbf{n}_x, \mathbf{n}_y, \mathbf{n})$ calculated with $\mathbf{n}$ pointing in the collision normal direction and the pair $(\mathbf{n}_x, \mathbf{n}_y)$ spanning the tangential collision plane (Fig. 4.6). The total contact force will be denoted by the vector $\mathbf{f} = (f_x, f_y, f) = f_x\mathbf{n}_x + f_y\mathbf{n}_y + f\mathbf{n}$. The Coulomb model requires for this force the following inequality:

$$f_x^2 + f_y^2 \leq \mu^2 f^2. \tag{4.27}$$

This inequality geometrically describes the inside of a cone and is commonly called the *friction cone*. Any force (or impulse) that lies within this cone satisfies the Coulomb friction model.

Typically [7, 59], when two bodies **A** and **B** collide with non-zero tangential velocity, the direction of the tangential component of the "bounce impulse" is chosen to oppose the

tangential velocity. The magnitude is set to μ times the magnitude of the normal impulse, as calculated in Section 4.2.1,

$$\mathbf{j}_t = -\mu|\mathbf{j}_n||\mathbf{v}_t|^{-1}\mathbf{v}_t, \tag{4.28}$$

where $\mathbf{v}_t = \mathbf{v}_{ab} - (\mathbf{n} \cdot \mathbf{v}_{ab})\mathbf{n}$ is the tangential part of the collision velocity $\mathbf{v}_{ab}$ between $\mathbf{A}$ and $\mathbf{B}$. Unfortunately, the assumption that friction always opposes the tangential velocity cannot always be supported [13, 45, 74, 80]. This assumption is an idealized view in the rigid body world, and closer examination on a microscopic scale reveals that the direction of frictional impulse can change during impact. Although this has no direct negative effect on graphic visualization, it can sometimes cause trouble in the actual implementation. Arising numerical difficulties can make it harder to find a good solution.

The non-colliding ($v^{\perp} = 0$) case is divided into the *sliding* case ($|\mathbf{v}_t| > 0$) and the *static* case ($|\mathbf{v}_t| = 0$). It sometimes makes sense to assume more friction and therefore larger μ for static friction. This can be viewed as initially larger friction at a contact. Once the contact slides, there is less opposition against the direction of sliding. In these cases, friction is a *force*, not an impulse. Contrary to popular belief, sliding friction is not velocity dependent. Regardless of how fast a block slides down a ramp, friction is the same. In the static case, the tangential friction force is exactly what is needed to prevent sliding: up until the magnitude required violates the Coulomb constraint. At that point, the contact "breaks free" and starts sliding. The transition between the static and sliding cases is very hard to model under the analytical method.

We attempted a solution to the problem with a new way of setting up a QP that calculates body accelerations at frame times. However, we also had to realize that the addition of friction is making finding of a solution hard. Ultimately, we employ an impulsive solution that always finds a solution, is efficient, and can be implemented easily.

Chapter 5

Simulation Techniques and Previous Work

Before we can introduce our new paradigm, optimization-based animation, we feel it is necessary to explain existing simulation techniques and their limitations. This chapter will give an overview of previous work. The high quality of applications of known techniques in commercials, computer games, and movies is a proof of the excellent work that has been done by many researchers. Yet there are drawbacks in that some of these techniques are specialized for only certain kinds of simulations or have performance limitations arising when the number of bodies in a simulation is scaled up to large numbers.

5.1 Analytical Methods

The most direct way of enforcing non-penetration is given by *analytical* methods. They accurately model the laws of Newtonian mechanics to derive exact contact forces that prevent interpenetration and were studied by various researchers, including Lötstedt [52,

53], Hahn [39], and Baraff [7]. As mentioned in Section 4.2.2, Lötstedt was the first to formulate a LCP which expresses the non-penetration conditions at each contact point. Intuitively, any force at a contact has to be non-adhesive, the contact acceleration has to be such that the contact separates, and if it actually separates, the force has to immediately assume magnitude *zero*:

$$\begin{aligned} \mathbf{f}^T\mathbf{a} &= 0, \\ \mathbf{a} &\geq 0, \\ \mathbf{f} &\geq 0. \end{aligned} \tag{5.1}$$

Solutions for the LCP can always be found if the modeled system is frictionless. Various sources, even as early as in the late 1800s [69], report that the presence of friction can cause problems when contact forces are to be calculated [8, 9, 14, 52, 53, 58, 60, 59, 79, 80]. There are contact configurations for which no force solution exists, or the solution can be ambiguous. In the first paradox case it can be observed that even an infinite force cannot prevent interpenetration at a contact. The second case arises if more than one force solution can be applied. Lötstedt gives conditions for solution existence and uniqueness [52]. More recent and comprehensive results have been published by Pang and Trinkle [70], and Trinkle *et al.* [86]. Baraff describes a method that allows impulsive forces according to the *principle of constraints* when forces alone cannot solve contact [8]. His method uses approximations to model friction. Dynamic friction is modeled correctly and static friction is approximated by it. He later generalized his work to flexible bodies [10].

Another problem arises with a different kind of force ambiguity. Analytical methods claim to be "exact" in the sense of exact application of the laws of physics. This is in principle true setting aside numerical round-off errors, and analytical methods will indeed

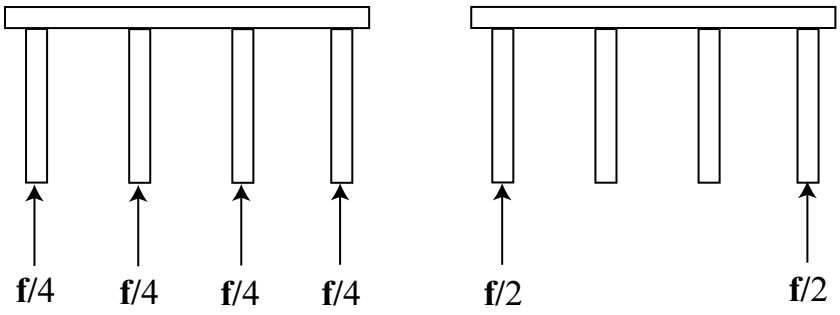


Figure 5.1: Both force combinations result in the same total force and thus the same motion.

yield the correct total body force or acceleration. It can be shown that the same total force on a body could be derived as a combination of different sets of contact forces. Figure 5.1 gives an example, which depicts a simple scenario where several sets of force solutions are possible to yield the same and correct total force. Although this insight is puzzling from a physical viewpoint, we have not found any visual problems induced by it. As long as the total force, or acceleration on a body is correct, the correct motion will result. We can justify and accept these ambiguities if we remember the main interest of graphics to produce images and animations that "look right" or "look realistic enough".

It has been stated that problems with friction arise due to the underlying models, which assume rigidity of bodies and apply the Coulomb friction law [60, 80] in a literal manner. Although these assumptions produce acceptable results for some simpler experimental physics, both are idealized viewpoints and therefore not accurately realistic. It has been noted by Stewart [80] that Coulomb friction is an acceptable approximation when friction between the bodies is moderate. Baraff gives a pivoting algorithm, which can calculate contact forces with static and dynamic friction [9]. Although it is a standard in many current animation packages, it is not clear whether this algorithm scales to arbitrary numbers of contacts without application of numerical tricks and heuristics.

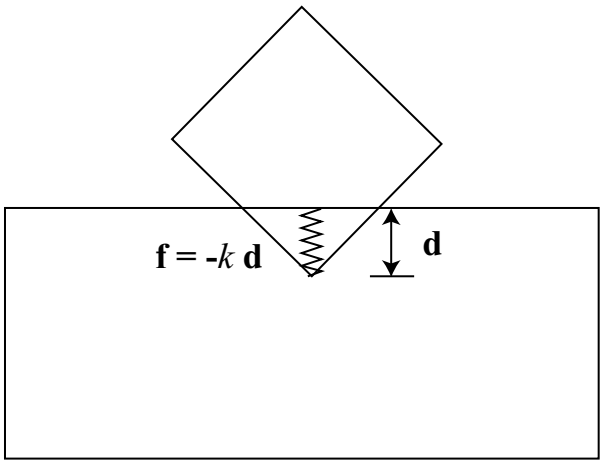


Figure 5.2: A symbolic spring counteracts interpenetration.

LCPs and related QPs have been applied to rigid body simulation by various researchers. Lötstedt did some early work in this area [53]. Baraff has a series of papers addressing the issue [7, 8, 9]. Stewart [80], Stewart and Trinkle [79], and similarly Sauer and Schömer [75] devise LCP-based methods that allow solving the complete dynamics problem, *i.e.* body positions and contact impulses and forces by solving one LCP. Although they alleviate the small time-step problem, these methods still have the notion of time-steps that are smaller than the frame time for numerical reasons.

5.2 Penalty Force Methods

Opposing the ubiquitous non-penetration constraints, penalty force methods allow interpenetration of bodies. As soon as interpenetration is detected, a symbolic spring with a high spring constant k is inserted between the bodies (Fig. 5.2). It exerts a repulsive force proportional to the interpenetration depth $\mathbf{d}$:

$$\mathbf{f} = -k\,\mathbf{d}. \tag{5.2}$$

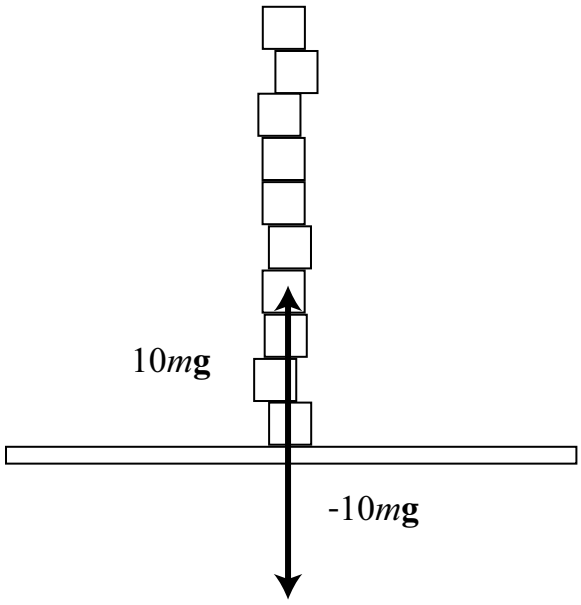


Figure 5.3: 10 cubes are stacked: the contact force between floor and the lowest cube has to compensate the weight of all 10 cubes.

This way, interpenetration is kept at a minimum. When bodies become receding, the spring is no longer needed and is therefore deleted. Penalty force methods are simple to implement, and they are a good choice for various applications. They are for these reasons commonly used for purposes of computer simulation [62, 63].

However, they exhibit some problems. Interpenetration is not prevented but "fixed" with penalty forces after it has happened. Hence, interpenetration can in some cases be deep, which generates a high repulsive spring force. Instabilities in the simulation are very likely introduced as a result. In addition, large spring constants could be necessary to avoid further interpenetration. We saw in Section 2.1.7 that large spring constants introduce stiff differential equations (ODEs). Integration of such equations has been reported to be hard, but recent developments use implicit integration methods [11], which make this problem tractable. Implicit methods have been known in the numerics community for a while, and algorithms are available for the solution of stiff ODEs [40].

A more severe drawback are singularities in the direction of friction. For tangential sliding velocities close to zero, the direction of friction becomes discontinuous. There are techniques for integration of stiff ODEs with discontinuities, but they generally have very high computational cost.

Furthermore, even "stiff solvers" cannot use an integration time-step larger than about 10% of the period of oscillation of the "penalty springs". Consider the scene in Figure 5.3 with a stack of 10 cubes with unit width and mass on a static floor. If we allow 1% interpenetration $\mathbf{x}$, a spring between the floor and bottom cube needs to have the spring constant k:

$$\begin{aligned} k\mathbf{x} &= m\mathbf{g} \Rightarrow \\ k\frac{1}{100} &= 10 \cdot 10 \Rightarrow \\ k &= 10000. \end{aligned} \tag{5.3}$$

We solve the equation of motion $m\ddot{\mathbf{x}} = -k\mathbf{x}$ with $\mathbf{x} = \cos(\omega t)$:

$$\begin{aligned} -m\omega^2 \cos(\omega t) &= -k\cos(\omega t) \Rightarrow \\ \omega &= \sqrt{\frac{k}{m}} \Rightarrow \\ T &= \frac{2\pi}{\omega} \Rightarrow \\ T &= 2\pi\sqrt{\frac{m}{k}} \Rightarrow \\ T &\approx 2\frac{1}{30}. \end{aligned} \tag{5.4}$$

Assuming the time-step for stiff integration is 1/10 the period of the oscillation, this result implies $1/10 \cdot 2/30 = 1/5 \cdot 1/30$, or 5 integration steps per frame. For the static force calculation, Equation 5.4 leads to the conclusion that we need to take five integration steps per (1/30 sec.) frame time. This result is actually not too pessimistic, and Mirtich

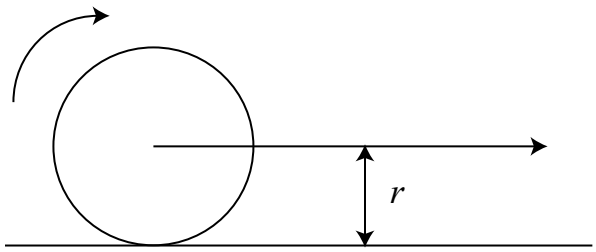


Figure 5.4: The ball is constrained by the table to roll with its center parallel to the surface.

obtained good results for static force calculation using stiff integration [62].

However, the same method would run much more slowly if we also applied it to the momentum update. Consider two colliding cubes with unit mass and width and relative collision velocity $\mathbf{v}_c$. We allow again 1% interpenetration and derive k:

$$\begin{aligned} \frac{1}{2}m\mathbf{v}^2 &= \frac{1}{2}k\mathbf{x}^2 \Rightarrow \\ k &= \mathbf{v}_c^2 \cdot 10000 \Rightarrow \\ T &= \frac{2\pi}{|\mathbf{v}_c|}\sqrt{\frac{m}{k}}. \end{aligned} \tag{5.5}$$

The resulting period T is therefore $|\mathbf{v}_c|$ times smaller than in Equation 5.4. Equation 5.5 shows that for bodies of unit mass and width, the period is inversely proportional to the collision velocity. To perform reliable integration, we must choose an integration time-step that is $v_{\max}$ times smaller than that of the static force calculation, where $v_{\max}$ can be 30 or larger. It is also not clear how this method scales with multiple simultaneous contacts, but even if that is not a problem, momentum update will be computationally expensive because of the extra factor of 30 or more.

5.3 Constraint-Based Simulation

Constraints are a familiar concept of mechanics [37]. The concept of *holonomic* constraints is widely used. Holonomic constraints are a class of constraints that can be expressed as equations on the body coordinates. Consider a mass on a string. The mass can swing back and forth, but it is constrained onto the surface of a sphere with radius equal to the length of the string. Similarly, a ball that rolls over a table is constrained to roll in a way that keeps its center of mass parallel to the table at a distance equal to the ball's radius (Fig. 5.4). This has been explored for simulation purposes [12, 88]. Constraint forces are calculated and enforce the constraint at hand. These constraint forces do not do any work on the constrained body.

It has been stated that problems can arise with solution non-existence [79]. Different types of possible constraints are known. It has been shown that more than one type of constraint can model rolling contact [48]. These ambiguous constraints can produce different simulation behavior. It is therefore not trivial to choose the "right" constraint for a given situation. During the course of a simulation, the nature of constraints will change. A ball may roll down a slope, hit another body, bounce off, continue to roll on a table, and may eventually fall off the table. Each event changes the constraints on the ball. Keeping track of the changed constraints can be a hard problem. Finally, it is not easy to include impulses resulting from collisions into a constraint-based simulation. Constraint forces alone do not realistically model bouncing bodies.

5.4 Impulse-Based Dynamic Simulation

The paradigm of impulse physics was introduced by Mirtich [59, 60]. He describes a simulator that handles collisions and contacts entirely with the application of impulses. Due to the problems with force calculation, impulses have been described as a way to solve the problems with solution ambiguity and non-existence also in other places [8, 80]. As opposed to forces, impulses act instantaneously. An advantage over constraint-based and penalty force methods is claimed as far as realism and stability are concerned. Mirtich describes a simulator, called *Impulse*, which generates simulations with impressive levels of realism.

Mirtich introduces the interesting concept of *conservative advancement* (CA). A lower bound is calculated on the next *time of impact* (TOI). The whole system is then allowed to advance this safe time-step. Compared to *retroactive detection* (RD) where a simulation has to be backed up every time a collision is detected, wasted work is reduced. The exact time of collision is not calculated after the simulation has gone too far, but it always advances a safe step only. However, the simulator still has to face the problems arising from small time-steps as a scene becomes "crowded". The simulations that were generated with *Impulse* are very impressive in their level of realism.

5.5 Position-Based Physics

Our work is very closely related to Milenkovic's *position-based physics* [57]. It can in fact be seen as an extension and generalization to position-based physics. Milenkovic was able to simulate 1000 spheres in a polyhedral hourglass with *zero*-order physics. That means the spheres have no notion of velocities or accelerations. Furthermore, they do not

bounce, and there is no friction modeled therefore. Pairwise non-overlap constraints are enforced by linear programming with the objective to bring all spheres as close as possible to "where they want to be". Milenkovic investigates the use of a variety of potential energy functions for this purpose. Bodies can only translate; they are not allowed to rotate. This is acceptable for spheres but imposes a severe limitation when arbitrary polyhedra are to be animated. The polyhedral hourglass is static and has therefore not to face this problem. Since bodies have no momenta, the algorithm is not suited for simulating free-flying unencumbered motion. Gravitational acceleration has to be "faked" with additional bounding box constraints. Although the algorithm can efficiently generate "clumps" and "piles" of spheres, its realism is limited.

An additional artifact is introduced by the nature of the simplex algorithm. The spheres are attracted to extreme points of the feasible region. Depending on the objective, this could make them "fall" at an angle. Milenkovic proposes a fix by modifying the objective with the addition of "conservative forces". However, in the final video, spheres still collect in a diamond shape as they hit the hourglass floor, as can be seen in Figure B.2(a). This is again a resulting artifact due to the underlying simplex algorithm.

5.6 Timewarp Rigid Body Simulation

Mirtich's timewarp algorithm [62] attacks the same problem as our proposed paradigm. It is based on Jefferson's work [43] and paves the way for parallel or distributed simulation. Mirtich gives a very comprehensive summary of problems with existing techniques. He states that efficiency suffers mostly due to work which is wasted when a simulator has to back up in simulations with RA and due to the known small time-step problem regardless

of whether RA or CA is employed. Conceptually, finding the exact TOI can be viewed as a root finding problem. Simple bisection, Newton's method, or *regula falsa* can be employed for this purpose. The benefits and disadvantages of each method have been discussed [3, 7, 63].

The achievement of timewarp is the de-synchronization of simulations by exploiting discrete properties of rigid body simulation. Mirtich observes that it is not necessary to maintain a global simulation clock in every case. Spatially separated groups of bodies do not have to be simulated in a synchronized fashion as long as they do not interact with each other. Contact groups are introduced, and groups are integrated individually according to local clocks. A global messaging and rollback mechanism takes care of collisions that occur between groups. Wasted work is minimized, and thus a tremendous speed-up can be achieved. Mirtich was able to simulate an avalanche of hundreds of rocks with his timewarp paradigm.

Although it is specifically designed to address the small time-step problem, timewarp can not avoid it when all bodies form one cluster as in a stack. In this case, all bodies form essentially one large contact group which has to be integrated as a unit. A stack cannot be de-synchronized and timewarp reduces to the underlying CA method. A performance gain can therefore no longer be achieved. Even worse, and even for the case when de-synchronization is possible, small time-steps cannot be ruled out due to the nature of the problem.

5.7 Animation with Neural Networks

For completeness and interest, we include the use of neural networks for the purpose of animation although it is not directly related to our work. Grzeszczuk *et al.* [38] have implemented a simulator based on machine learning. The authors observed the computational cost of numerical simulation of physics-based models. They propose *NeuroAnimator*, a simulator based on methods from neural networks. NeuroAnimator is trained off-line and learns how to emulate physical dynamics through the observation of physics-based models in action. They report a speed-up over traditional simulation methods by one or two orders of magnitude. Furthermore, NeuroAnimator has the capability of meeting certain desired animation goals, such as parking a car in a particular manner. NeuroAnimator can be trained not only to produce realistic motion, but a motion that meets the needs of a script. In this way, the method is truly interesting and innovative. Physical motion alone is for some applications not enough, since certain goals need to be met [24, 47, 71]. NeuroAnimator is shown to work well for specific examples. However, there is no evidence of its applicability to crowded systems of bodies, in which we are interested.

Chapter 6

OBA Simulation of Rigid Bodies

We have so far developed and introduced the mathematical and physical tools for rigid body simulation. With this knowledge, we can now explain how we employ mathematical programming to generate "crowded" animations efficiently. Our proposed paradigm, optimization-based animation (OBA) [58], addresses the small time-step problem that arises in simulation. Mirtich gave a very comprehensive summary of the problems involved in traditional rigid body simulation [62]. The two main approaches to rigid body simulation are retroactive detection (RD) and conservative advancement (CA).

Trajectories of bodies are integrated forward in time under non-overlap constraints. In RD, the integrator takes a step from t_0 to $t_0 + \Delta t$ and detects overlap at the newly calculated positions. If there are any, a root finding method is employed and an exact permissible time-step $\Delta t' < \Delta t$ is found that will not generate overlap. The initial advancement of all bodies to Δt has to be cancelled. Instead, trajectories are integrated up to $\Delta t'$. Computation work is first done, then cancelled and redone with different parameters. Hence, work is actually wasted.

CA recognizes this problem and establishes a conservative upper bound on the time-

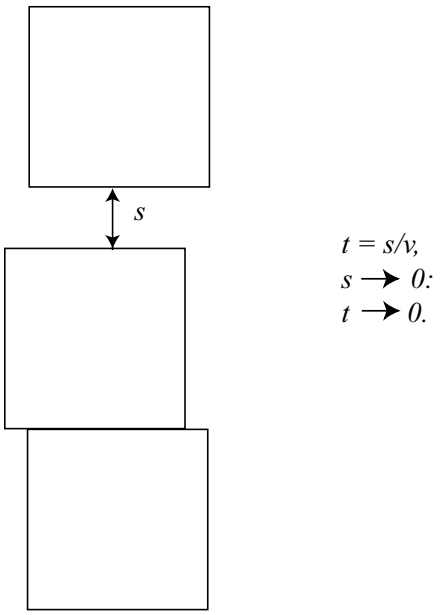


Figure 6.1: For shrinking space and constant velocity, the time without collisions becomes smaller.

step Δt that ensures the simulation will not have to be backed up and restarted. The amount of wasted work is minimized, but CA still has to face a different problem. If the bodies in a simulation are in very close proximity, the admissible time-step until the next collision happens will be small. Assuming constant velocities, the time $t = s/v$ goes to zero as the free space shrinks, *i.e.* $s \to 0$ (Fig. 6.1). This is observed in stacks of bodies where each body gets wedged between the body below and above it. The result is a rattling motion with high collision frequency. In fact, the number of collisions will tend towards infinity as the space shrinks. An infinite number of collisions makes simulation intractable. Note that the total number of bodies does not have to be excessively large. The example of a ping-pong paddle that is brought down over a jumping ball is a good illustration to the problem. Only three bodies are in the scene, yet the number of bounces grows audibly.

The small time-step problem can also be explained by statistical reasoning for large numbers of bodies. The "mean free path length" of atoms in a constant volume is indi-

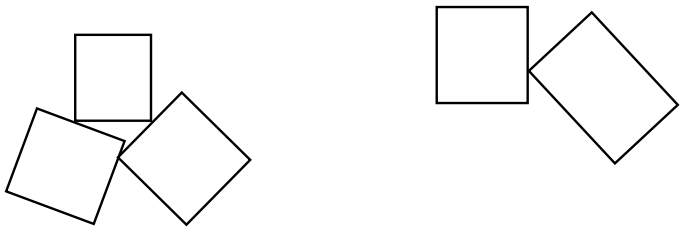


Figure 6.2: Timewarp: the two disjoint groups can be integrated independently.

rectly proportional to the density of atoms [35]. As the number of atoms and therefore their density grows, the mean free path length and hence the admissible time-step shrinks. Eventually, the admissible time-step becomes extremely small. Many micro-steps are necessary to advance the simulation for just a single frame time, and the simulation appears as if stopped. The number of collisions is a polynomial in the number of bodies. For n bodies, we have n^2 pairs. The mean free path length is proportional to n^{-1}, and thus the number of overall bounces is n^3.

Mirtich recognized that a part of the small time-step problem is caused by unnecessary synchronization in simulations [62] and devised a timewarp algorithm that integrates disjoint body groups independently (Fig. 6.2). A group should not have to be stopped due to a collision in another group if the two groups are spatially separated. A global messaging mechanism is implemented which re-synchronizes local group time to global simulation time. Efficiency is gained, because the amount of wasted work and the number of micro-steps are reduced. If all bodies form one group as in a stack, the algorithm loses its advantage. However, even if de-synchronization is possible, the rattling motion of confined, colliding bodies cannot be eliminated and, in essence, we are dealing with an intractable problem.

Our OBA algorithm also achieves de-synchronization and avoids the small time-step problem, but with different methods. OBA also implements the simulation loop from

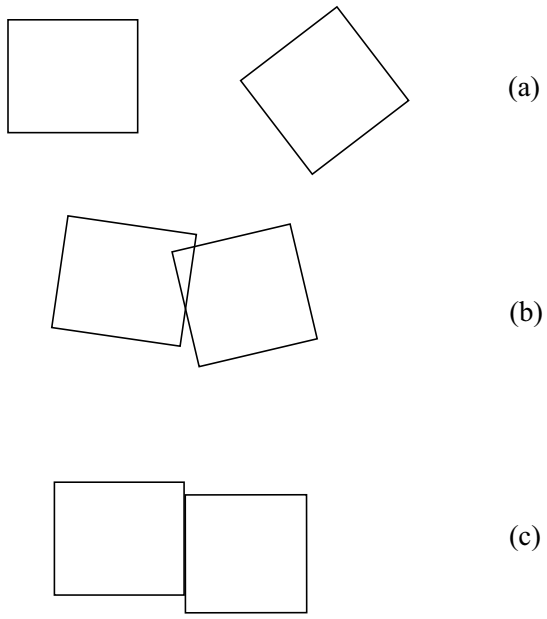


Figure 6.3: OBA uses iterative optimization to move bodies from non-overlapping *current* positions (a) as close as possible to their *targets* (b) without overlap (c).

Figure 1.1 but uses a fixed time-step Δt, which we set equal to the frame time (1/30 sec.). Therefore, OBA finds positions, momenta, and forces for all bodies strictly at frame times and does not have to execute micro-steps in-between as in traditional approaches.

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 (Fig. 6.3(b)). An iterative optimization algorithm moves the bodies *as close as possible* to their target positions and orientations under the constraint that bodies cannot overlap (Fig. 6.3(c)). If bodies overlap at their targets, the optimization translates and rotates them the minimal amount to get rid of the overlap. We insert separating plane constraints for each close pair of bodies that make sure each body stays to its assigned side of the separating plane (Fig. 6.4). If we can find a plane between two bodies then the bodies do not overlap. In this manner we avoid the micro-steps of traditional simulation algorithms. Our OBA algorithm can compute plausible visible states at frame times without executing all the steps in-between.

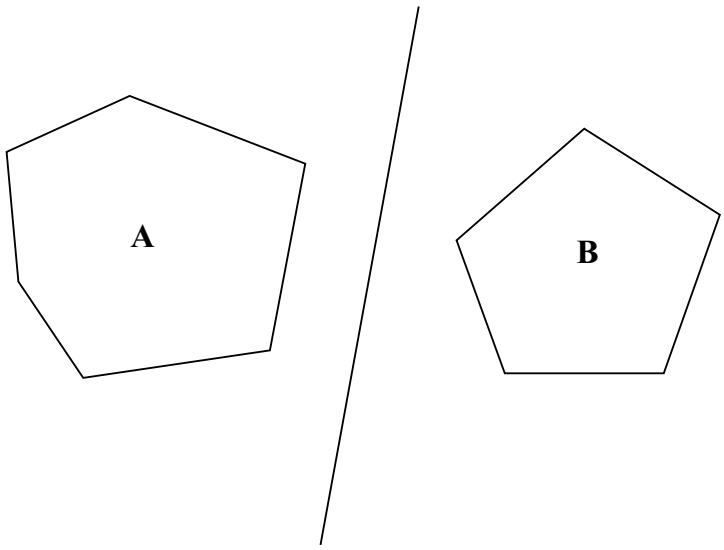


Figure 6.4: If we can find a plane between bodies **A** and **B**, the bodies are disjoint.

It tends to align neighboring bodies 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 has an approximating character due to the underlying optimization techniques. The overall motion of bodies is physical within limits where physical realism is actually visible. However, body positions and the times at which they collide are altered. Consequently, OBA is not suitable for applications that require microscopically accurate motion. It can be perfectly employed in situations where a plausible realistic motion is sufficient. Chenney and Forsyth [24] have shown plausible motion to be useful for animation. 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.

Our optimization-based algorithm will find non-overlapping positions for bodies. However, bodies may be contacting at these positions. Therefore, collisions and static contacts have to be resolved. To do so, 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 1.1. It finds new body positions but also provides the contact points at the new positions. Collisions and contacts are then resolved with Coulomb friction. Calculating contact forces under an analytical model is hard. We devise a purely impulsive scheme that solves collisions and static contacts, and produces very realsitic simulations.

Due to OBA's modular nature, code from existing simulation packages can be reused. The contact points, which were found after the position update can be used as input for any contact resolution algorithm [7, 9, 17, 45, 59, 63]. It is not necessary to implement OBA verbatim as described in this paper, but its modules can be employed to one's benefit when it is appropriate.

Friction is a very substantial part of everyday physical reality. Even simple tasks, such as walking, sitting, picking up objects, or shaking somebody's hand would not be possible without it. Unfortunately, modeling static contacts with friction is a non-trivial task. Baraff has shown [8] this problem to be NP-hard. It is a widely used heuristic to the problem to compute impulsive forces whenever static contact forces cannot be computed.

Each of the stages of our OBA algorithm is implemented as a quadratic programming problem. We devise hybrid approaches for the position update and for the momentum update. This is to circumvent potential but rare problems in the position update. Thin bodies with large velocities might pass through each other if not treated with care. For the momentum update, we introduce hybridization to allow specific simulations. For example,

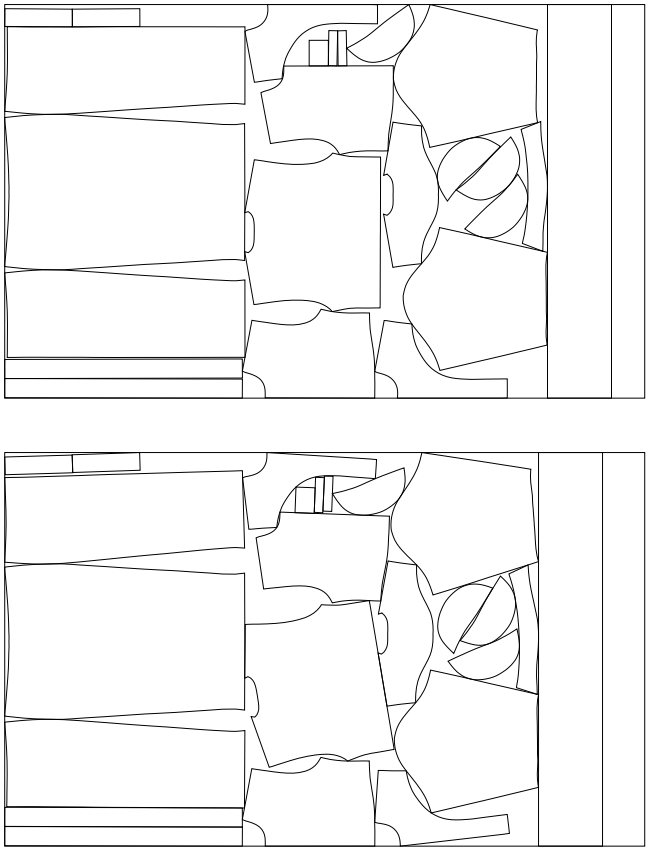


Figure 6.5: Production cost in the textile industry is minimized on the bottom: the strip length of used raw material is shorter when the pieces are arranged by compaction. Both strips have the same width.

the office toy pendulum cannot be generated by pure QP-based momentum calculation.

We will now develop each of the OBA component algorithms. For completeness and better overview, a detailed summary of QPs that are solved in each component is placed in Appendix A.3.

6.1 Position Update

OBA is closely related to Milenkovic's *position-based physics* [57]. Milenkovic used linear programming to animate a polyhedral hourglass with 1000 spheres (Fig. B.2(a)). He finds positions of bodies under non-overlap constraints and suggests several objective functions

that minimize a gravitational potential. Bodies move according to zero-order physics, *i.e.* they have no notion of bouncing and therefore friction, and they do not follow parabolic trajectories. This work was a generalization of an algorithm for translational *compaction* of two-dimensional part layouts [49]. Milenkovic's compaction work is used in algorithms for the clothing industry to facilitate part layout (Fig. 6.5). Clothing manufacturers are concerned with minimization of waste, *i.e.* arranging parts in a way that makes the best possible use of the raw material. Milenkovic has also generalized the compaction algorithm to allow rotations in two dimensions [56][1].

Although his work is laying out the basis for our animation algorithm, the position update algorithm we present here is not simply a three-dimensional generalization of the rotational compaction algorithm. In contrast to Milenkovic's reduced realism in his position-based physics approach, our bodies collide elastically (bounce), are subject to friction, and follow Newtonian (parabolic) trajectories. We therefore achieve a higher level of physical realism. In order to achieve this, we do not simply minimize gravitational energy in the position update, but instead we set up an *artificial* energy potential.

First, we calculate *target* positions (and orientations) for all bodies ignoring collisions. To do so, we allow all bodies to go to where they want to be in the next frame regardless of other bodies that are in their way. To enforce non-overlap, we only move the bodies as close as possible to their targets. The artificial energy potential attracts the bodies to their targets without allowing them to overlap. 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 $t_{\text{tgt}} = t_{\text{cur}} + \Delta t$, if it followed its Newtonian trajectory in the absence of the other bodies. We set Δt equal to the frame

[1]Figure courtesy of V. J. Milenkovic.

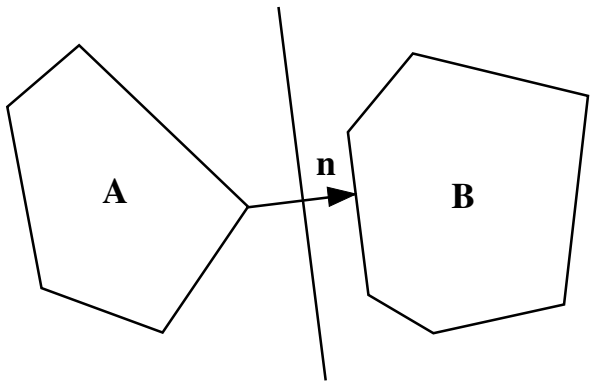


Figure 6.6: Separating plane between bodies **A** and **B** with normal vector **n**.

time. Consequently, two bodies **A** and **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 separating plane constraint to keep bodies non-overlapping. Section 6.1.2 describes possible positive-definite objectives that attract bodies to their target positions. Section 6.1.3 shows how to linearize the non-overlap constraint in order to write a quadratic programming problem. Section 6.1.4 gives an algorithm using iterated quadratic programming to minimize the objective under the non-linear separation constraint. Sections 6.1.5 and 6.1.6 describe ways how to keep the resulting QP small for efficiency and how to limit occurrence of infeasibilities.

6.1.1 Separating Plane Constraints

The main idea of our OBA position update is the concept of a separating plane. 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. 6.6). Specifically, convex bodies **A** and **B** do not overlap if and only if there exists a unit vector **n** and scalar d such that,

$$\forall \mathbf{q}_a \in \mathbf{A}, \mathbf{n} \cdot \mathbf{q}_a \leq d \quad \text{and} \quad \forall \mathbf{q}_b \in \mathbf{B}, \mathbf{n} \cdot \mathbf{q}_b \geq d. \tag{6.1}$$

Remember that the normal $\mathbf{n}$ always points from body $\mathbf{A}$ to body $\mathbf{B}$. We say body $\mathbf{A}$ is to the *left* and body $\mathbf{B}$ is to the *right* of the plane. Geometrically, $\mathbf{n}$ is the unit vector perpendicular to the separating plane, and d is the plane's distance from the origin. This gives rise to the plane equation $\mathbf{nx} = d$ for all points $\mathbf{x}$ on the plane.

Since the bodies are polyhedral, it suffices to check Equation 6.1 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. Note that spheres can be handled very similarly (Fig. B.1(b)). Assume spheres $\mathbf{A}$ and $\mathbf{B}$ with radii r_a and r_b, and center coordinates $\mathbf{x}_a$ and $\mathbf{x}_b$. Following Equation 6.1, we write for spheres:

$$\mathbf{n} \cdot \mathbf{x}_a \leq d - r_a \quad \text{and} \quad \mathbf{n} \cdot \mathbf{x}_b \geq d + r_b. \tag{6.2}$$

Each body starts at a current position $\mathbf{x}^{\text{cur}}$ and orientation $\mathbf{R}^{\text{cur}}$ in the current frame. The vector $\mathbf{x}^{\text{tgt}}$ and orientation matrix $\mathbf{R}^{\text{tgt}}$ denotes its target position: where its trajectory would take the body in the next frame if overlaps were ignored. If no pair of bodies overlaps at the targets, then there is no overlap to be resolved: 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, overlap has to be resolved by our QP position update algorithm. As described in Chapter 3, our collision detection algorithm efficiently finds all overlapping and potentially overlapping pairs. A pair is potentially overlapping if the bounding boxes overlap. If the distance for pairs with overlapping bounding boxes is below a threshold, we declare overlap and insert the pair into a list of close pairs.

6.1.2 Objective

We will now define the objective used in the OBA position update. We wish to move all bodies as close as possible to their target positions. This is achieved by defining a "distance to target" measure for the body positions and orientations, which we then minimize. Furthermore, we want an objective that is a positive-definite quadratic function of the distance to target measures, because this assures polynomial solution time for the QP (Sec. 2.2.2). We will first introduce the necessary variables.

As mentioned above, we have a current position $\mathbf{x}^{\text{cur}}$ and orientation $\mathbf{R}^{\text{cur}}$ for each body. There is no overlap between any pair of bodies at the current positions. The target position $\mathbf{x}^{\text{tgt}}$ and orientation $\mathbf{R}^{\text{tgt}}$ is a constant per time-step, as it represents the state where each body would go at the end of the time-step without collisions. Overlap at the targets must be resolved. Figuratively speaking, we need to find positions which are in-between the current and target positions and which show no overlap. Starting at the current positions, we perturb the bodies as close as possible to the targets and enforce non-overlap. Call the variable for body position and orientation $\mathbf{x}$ and $\mathbf{R}$ respectively. Initially, each body is at its current position, and we set $\mathbf{x} = \mathbf{x}^{\text{cur}}$ and $\mathbf{R} = \mathbf{R}^{\text{cur}}$. Since Equation 6.1 is non-linear, we will calculate a perturbed position $\mathbf{x}^{\text{per}}$ and orientation $\mathbf{R}^{\text{per}}$. 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

$$\Delta\mathbf{x} = \mathbf{x}^{\text{per}} - \mathbf{x} \quad \text{and} \quad \Delta\mathbf{R} = \mathbf{R}^{\text{per}}\mathbf{R}^{-1}. \tag{6.3}$$

We also define *target perturbations*. These are the perturbations which would move the body directly from the current position to its target:

$$\Delta\mathbf{x}^{\text{tgt}} = \mathbf{x}^{\text{tgt}} - \mathbf{x} \quad \text{and} \quad \Delta\mathbf{R}^{\text{tgt}} = \mathbf{R}^{\text{tgt}}\mathbf{R}^{-1}. \tag{6.4}$$

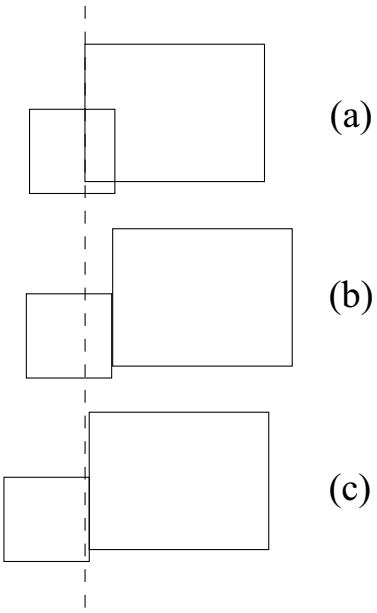


Figure 6.7: For overlapping bodies (a), we do not want the lighter body to push aside the heavier body (b), but the heavier body should push the lighter body (c).

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

$$\Delta\Delta\mathbf{x} = \Delta\mathbf{x} - \Delta\mathbf{x}^{\text{tgt}} \quad \text{and} \quad \Delta\Delta\boldsymbol{\phi} = \Delta\boldsymbol{\phi} - \Delta\boldsymbol{\phi}^{\text{tgt}}. \tag{6.5}$$

These represent the distance from the current perturbations to the target perturbations. When the target deficits 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. This is equivalent to minimizing the target deficits. We introduce as a natural measure for the distance to the targets the sum of weighted squares of the target deficits. For this purpose, we plug the target deficits into the formula for kinetic energy. The subscript i refers to that property of the i-th body:

$$\sum_{i=1}^{k} \frac{1}{2} m_i \mathbf{v}_i \cdot \mathbf{v}_i + \frac{1}{2} \boldsymbol{\omega}_i^T \mathbf{I}_i \boldsymbol{\omega}_i, \tag{6.6}$$

Equation 6.6 is the sum of kinetic energies for k bodies. We drop the constant factor $\frac{1}{2}$,

and in place of the velocities we plug in the linear and angular displacements:

$$\sum_{i=1}^{k} m_i \Delta\Delta\mathbf{x}_i \cdot \Delta\Delta\mathbf{x}_i + \Delta\Delta\boldsymbol{\phi}_i^T \mathbf{I}_i \Delta\Delta\boldsymbol{\phi}_i. \tag{6.7}$$

The linear and angular parts are weighted by the body mass and inertia respectively. The physical motivation behind this objective is that a heavier body pushes a lighter body out of the way. This makes sense because it follows an intuitive approach, which is verified by empirical observation in real life.

We also tested the following objective:

$$\sum_{i=1}^{k} \Delta\Delta\mathbf{x}_i \cdot \Delta\Delta\mathbf{x}_i + D_i^2 \Delta\Delta\boldsymbol{\phi}_i \cdot \Delta\Delta\boldsymbol{\phi}_i. \tag{6.8}$$

The diameter D of a body is the largest distance between two vertices on the body. We have to weight the rotational part by D so that it has the same units as the translational part. A rotation $\Delta\boldsymbol{\phi}$ moves a point $\mathbf{q}$ on the body by at most a distance $|\Delta\boldsymbol{\phi}|\,|\mathbf{q}|$, and D is an upper bound on $|\mathbf{q}|$. One might argue the objective from Equation 6.7 makes more sense physically. Both methods work well in practice with slight differences in simulation stability in favor of Equation 6.8. If the bodies in a scene vary greatly in size, shape, and density, Equation 6.7 might be a better choice, because it considers individual body inertias and masses.

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 $x = 0$ (Fig. 6.8(a)). 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 (Fig. 6.8(d)). If we use a non-quadratic objective such as $|x_1| + |x_2|$, then $(x_1, x_2) = (0, 1)$ (Fig. 6.8(b)) or $(x_1, x_2) = (-1, 0)$ (Fig. 6.8(c)) are both equally valid solutions which

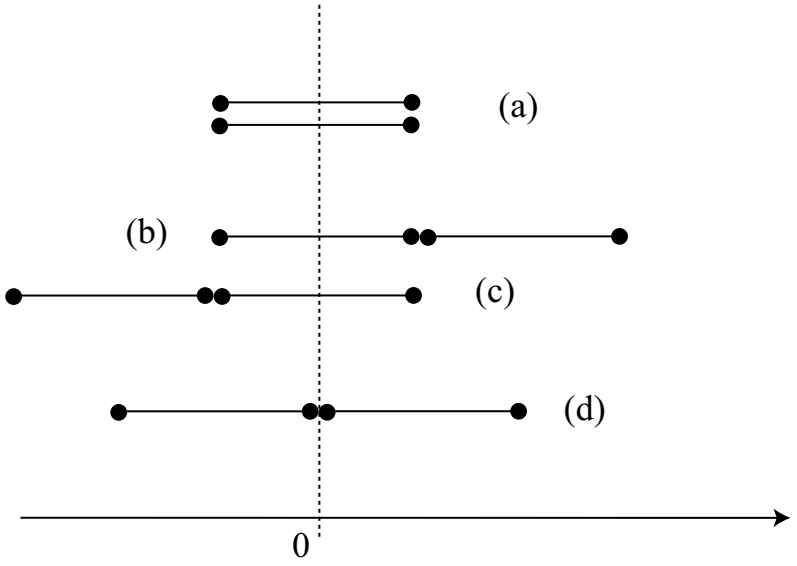


Figure 6.8: A simple one-dimensional example to explain why we use a quadratic objective.

minimize the objective. While $(x_1, x_2) = (-0.5, 0.5)$ is also a valid solution for this linear objective, the simplex algorithm for solving linear programs (LP) will *not* generate it, because the simplex algorithm always chooses an extreme (vertex) solution of the feasible solution space. Instead, the simplex algorithm 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.

6.1.3 Linearizing the Separation Constraints

The original separating plane constraints from Section 6.1.1 are non-linear and non-convex. Solving an optimization problem with such constraints is NP-hard in general. It is a standard trick to linearize constraints and yield a QP that can be solved in expected polynomial time. Solving the linearized QP will take a step towards a local minimum of

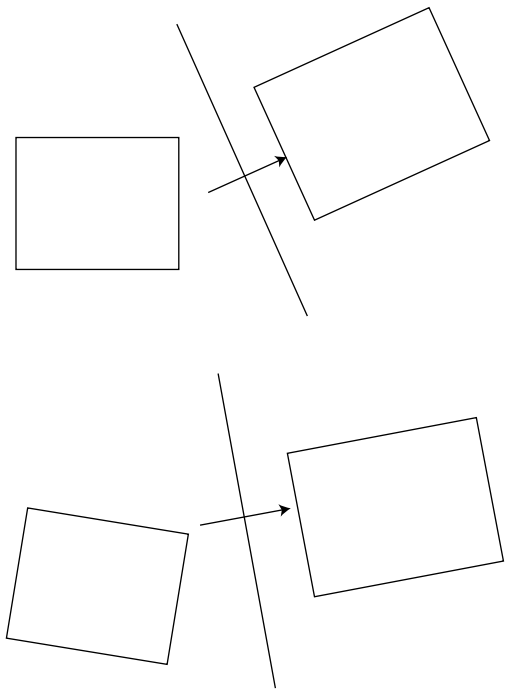


Figure 6.9: When the bodies in a pair are perturbed, also the separating plane changes.

the original problem. Finding the overall local minimum is achieved by iterating.

Finding a global minimum corresponds to the packing problem and is NP-complete. Rotational packing is at least NP-hard. However, it is perfectly physical to find a local minimum. Unloading a bag into a closet and pushing on the doors will not automatically make the objects jump to a global minimum. Instead, the objects will arrange themselves locally into an arrangement that minimizes internal pressure from the doors being pushed. It has been shown that iterative LP is good in practice for the two-dimensional rotational compaction problem [56].

We will now explain how to linearize the separation constraints in Equation 6.1. Let $\mathbf{n}$ and d denote a separating plane for body $\mathbf{A}$ at $\mathbf{x}_a, \mathbf{R}_a$ and $\mathbf{B}$ at $\mathbf{x}_b, \mathbf{R}_b$. As bodies are perturbed, also the separating plane between them will change its position and orientation, *i.e.* $\mathbf{n}$ and d (Fig. 6.9). Scalar $d^{\text{per}} = d + \Delta d$ is the new perturbed position of the separating plane. This is a linear equation, and we can therefore just have a single variable d per pair of

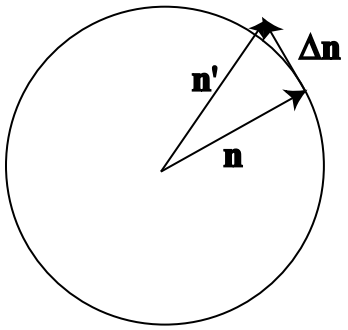


Figure 6.10: The linearized change in the separating plane normal $\mathbf{n}$.

bodies due to symmetry. A *collision coordinate frame* $(\mathbf{n}_x, \mathbf{n}_y, \mathbf{n})$ according to Section 4.3 has been computed. The vector $\mathbf{n}$ points in the collision normal direction, and $\mathbf{n}_x$ and $\mathbf{n}_y$ span the tangential collision plane. Vector $\mathbf{n}^{\text{per}}$ is the perturbed version of $\mathbf{n}$ with perturbation:

$$\mathbf{n}^{\text{per}} = \mathbf{n} + \Delta\mathbf{n}. \tag{6.9}$$

The perturbation of $\mathbf{n}$ lies really on a sphere with unit radius. We linearize and approximate it for a small perturbation where $\Delta\mathbf{n}$ is perpendicular to $\mathbf{n}$ (Fig. 6.10). We introduce scalar variables δ_x and δ_y

$$\Delta\mathbf{n} \approx \delta_x\mathbf{n}_x + \delta_y\mathbf{n}_y, \tag{6.10}$$

and yield the final linearized perturbed $\mathbf{n}$:

$$\mathbf{n}^{\text{per}} \approx \mathbf{n} + \delta_x\mathbf{n}_x + \delta_y\mathbf{n}_y. \tag{6.11}$$

Without loss of generality, we first derive the linearized separating plane constraint for body $\mathbf{A}$, *i.e.* the left half of Equation 6.1. If $\mathbf{q}_a^{\text{body}}$ are the body coordinates of a vertex $\mathbf{q}_a$ on body $\mathbf{A}$, then its unperturbed and perturbed world coordinates are:

$$\mathbf{q}_a = \mathbf{R}_a\mathbf{q}_a^{\text{body}} + \mathbf{x}_a \quad \text{and} \quad \mathbf{q}_a^{\text{per}} = \mathbf{R}_a^{\text{per}}\mathbf{q}_a^{\text{body}} + \mathbf{x}_a^{\text{per}}. \tag{6.12}$$

Combining these with Equation 6.3 ($\mathbf{R}_a^{per} = \Delta\mathbf{R}_a\mathbf{R}_a$) yields:

$$\begin{aligned}\mathbf{q}_a^{\text{per}} &= \Delta\mathbf{R}_a\mathbf{R}_a\mathbf{q}_a^{\text{body}} + \mathbf{x}_a^{\text{per}} = \\ &= \Delta\mathbf{R}_a(\mathbf{R}_a\mathbf{q}_a^{\text{body}}) + \mathbf{x}_a^{\text{per}} = \\ &= \Delta\mathbf{R}_a(\mathbf{q}_a - \mathbf{x}_a) + \mathbf{x}_a^{\text{per}}. \end{aligned} \tag{6.13}$$

Equation 6.13 is non-linear. We linearize this formula using the small-angle approximation $\Delta\mathbf{R}\mathbf{w} \approx \mathbf{w} + \Delta\boldsymbol{\phi} \times \mathbf{w}$ and write with Equation 6.3:

$$\begin{aligned}\mathbf{q}_a^{\text{per}} &\approx \mathbf{q}_a - \mathbf{x}_a + \Delta\boldsymbol{\phi}_a \times (\mathbf{q}_a - \mathbf{x}_a) + \mathbf{x}_a^{\text{per}} = \\ &= \mathbf{q}_a + (\mathbf{x}_a^{\text{per}} - \mathbf{x}_a) + \Delta\boldsymbol{\phi}_a \times (\mathbf{q}_a - \mathbf{x}_a) = \\ &= \mathbf{q}_a + \Delta\mathbf{x}_a + \Delta\boldsymbol{\phi}_a \times (\mathbf{q}_a - \mathbf{x}_a). \end{aligned} \tag{6.14}$$

Now plug Equations 6.11 and 6.14 into Equation 6.1 ($\mathbf{n}^{\text{per}} \cdot \mathbf{q}^{\text{per}} \leq d$) to write the separating plane constraint for a vertex $\mathbf{q}_a$ on body $\mathbf{A}$ in its perturbed, linearized version:

$$(\mathbf{n} + \delta_x\mathbf{n}_x + \delta_y\mathbf{n}_y) \cdot (\mathbf{q}_a + \Delta\mathbf{x}_a + \Delta\boldsymbol{\phi}_a \times (\mathbf{q}_a - \mathbf{x}_a)) \leq d. \tag{6.15}$$

In this inequality, we have vector variables $\Delta\mathbf{x}_a$ and $\Delta\boldsymbol{\phi}_a$, and scalar variables δ_x, δ_y, and d. We carry out the multiplication, but throw away quadratic terms:

$$\mathbf{n} \cdot \mathbf{q}_a + \mathbf{n} \cdot \Delta\mathbf{x}_a + \mathbf{n} \cdot (\Delta\boldsymbol{\phi}_a \times (\mathbf{q}_a - \mathbf{x}_a)) + (\delta_x\mathbf{n}_x) \cdot \mathbf{q}_a + (\delta_y\mathbf{n}_y) \cdot \mathbf{q}_a \leq d. \tag{6.16}$$

Now, move all terms with variables to the left and terms with constants to the right and pull out the scalar variables from the last two terms:

$$\mathbf{n} \cdot \Delta\mathbf{x}_a + \mathbf{n} \cdot (\Delta\boldsymbol{\phi}_a \times (\mathbf{q}_a - \mathbf{x}_a)) + (\mathbf{n}_x \cdot \mathbf{q}_a)\delta_x + (\mathbf{n}_y \cdot \mathbf{q}_a)\delta_y - d \leq -\mathbf{n} \cdot \mathbf{q}_a. \tag{6.17}$$

In the final step, we apply the identity $\mathbf{a} \cdot (\mathbf{b} \times \mathbf{c}) = -(\mathbf{a} \times \mathbf{c}) \cdot \mathbf{b}$. This separates the vector variable $\Delta\boldsymbol{\phi}_a$ and yields the linearized constraint:

$$\mathbf{n} \cdot \Delta\mathbf{x}_a - (\mathbf{n} \times (\mathbf{q}_a - \mathbf{x}_a)) \cdot \Delta\boldsymbol{\phi}_a + (\mathbf{n}_x \cdot \mathbf{q}_a)\delta_x + (\mathbf{n}_y \cdot \mathbf{q}_a)\delta_y - d \leq -\mathbf{n} \cdot \mathbf{q}_a. \tag{6.18}$$

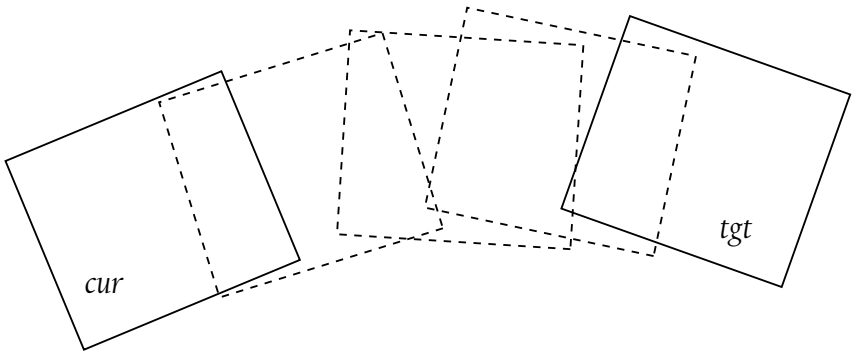


Figure 6.11: Left to right: as OBA iterates, the bodies move closer to their targets.

Equation 6.18 can be implemented as a constraint in a quadratic program. All variables have been separated, the constant coefficients and the right-hand side bound can be calculated. For symmetry reasons, we similarly write for a vertex $\mathbf{q}_b$ on body $\mathbf{B}$ from the right part of Equation 6.1:

$$\mathbf{n} \cdot \Delta\mathbf{x}_b - (\mathbf{n} \times (\mathbf{q}_b - \mathbf{x}_b)) \cdot \Delta\boldsymbol{\phi}_b + (\mathbf{n}_x \cdot \mathbf{q}_b)\delta_x + (\mathbf{n}_y \cdot \mathbf{q}_b)\delta_y - d \geq -\mathbf{n} \cdot \mathbf{q}_b. \tag{6.19}$$

The linear non-overlap constraints are the heart of our position update QP. We can immediately write a QP with constraints according to Equations 6.18 and 6.19, and an objective with Equation 6.7 or 6.8. However, due to the linearization in the constraints, we need to introduce an iterative update mechanism as following in the next section.

6.1.4 Single QP vs. Iterative Update

The original non-overlap constraints are non-linear and non-convex. Solving this particular optimization is NP-hard in general. The linearized version forms a QP with constraints as in Equations 6.18 and 6.19 can be solved in polynomial time. This solution will not minimize the original non-linear problem, but it will make a step towards its minimum. We devise therefore an iterated quadratic programming algorithm which solves a series of QPs and converges to a local minimum of the original problem.

At each simulation time-step, we solve a series of QPs to update the body positions. At the beginning of a time-step, each body starts out at its current position and orientation. We initialize the values of $\mathbf{x}$ and $\mathbf{R}$ with the current position $\mathbf{x}^{\text{cur}}$ and orientation $\mathbf{R}^{\text{cur}}$. After each iteration, we apply the calculated perturbation by setting $\mathbf{x} = \mathbf{x} + \Delta\mathbf{x}$ and $\mathbf{R} = \Delta\mathbf{R}\,\mathbf{R}$ (Fig. 6.11). Here, $\Delta\mathbf{R}$ is the matrix format of the orientation vector $\Delta\boldsymbol{\phi}$. The unknown $\Delta\mathbf{x}$ and $\Delta\boldsymbol{\phi}$ are the solutions of a single QP. An individual linearized QP is made up as follows. Each body has twelve variables, the components of the perturbation vectors $\Delta\mathbf{x}$ and $\Delta\boldsymbol{\phi}$, and the target deficit vectors $\Delta\Delta\mathbf{x}$ and $\Delta\Delta\boldsymbol{\phi}$. Each close pair has three variables which express the separating plane perturbation, δ_x, δ_y, and d. Each body requires six linear equality constraints to enforce Equation 6.5. Each close pair has the linear inequality constraints of Equations 6.18 and 6.19. The objective is given by Equation 6.7 or 6.8. **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 perturbed positions and orientations of the bodies.

The objective is quadratic and positive-definite. The linearized constraints generate a series of QPs. When solved, they produce 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. Therefore, we know the system is non-overlapping at a local minimum.

6.1.5 Reckless Dynamic Update

The brute force approach to generate each single QP would be to add separating plane constraints for all vertices. Each constraint in the QP refers to a vertex on a body and a separating plane with respect to another body. This would certainly make sure that all vertices are on the proper side of the separating plane, and therefore the bodies do not overlap. However, this is not a good idea for efficiency reasons. The solution time for a QP is in general polynomial in the number of constraints in the QP, and therefore we have to keep the size of each QP as small as possible in order to do the position update efficiently. We cannot simply add constraints for every vertex with respect to every separating plane. Instead, we employ Milenkovic's reckless dynamic update approach [56]. It finds an initially minimal set of constraints and adds more in a dynamic fashion as needed. For each close pair of bodies, we first find *critical vertices* and add constraints (Eq. 6.18 or 6.19) only for these. Critical vertices are those which lie within a slab of the separating plane at the current positions. In Figure 6.12, only one true contact is present, but four vertices (**a**, **b**, **c** and **d**) lie within a critical slab. Note that a vertex is also added if it is deeply penetrating into the other body. Such vertices are *per se* not in the slab, but certainly critical.

The approach is indeed reckless because a currently non-critical vertex might move past the separating plane and result in an overlap after the computed perturbation is applied. We mend this oversight by adding this vertex as a critical vertex for the next iteration. In a way, the algorithm "learns" from its mistakes. Usually, any introduced 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 becomes stuck in an overlapping configuration which it cannot resolve by adding constraints, it will produce an *infeasible*

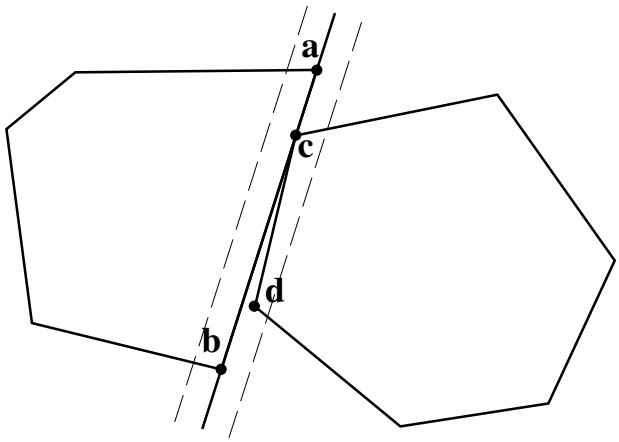


Figure 6.12: Explanation of critical vertices.

QP. A QP is infeasible if there is no possible solution that can satisfy all constraints. If this happens, we roll back a perturbation for *all* the bodies. Each body maintains a stack of previous perturbed positions in the current frame for this purpose. 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). The pseudo-code in Table 6.1 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. This means in particular that 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.

For spheres, the situation is slightly different, but even easier. Since we use an exact and not a polyhedral approximation representation for spheres, we consider each sphere to have only one critical "vertex" (calling it critical point would be more concise, but we stay with the terminology for consistency). This is the sphere's center, for which we add just one constraint in the form of Equation 6.2.

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 6.1: Position update algorithm.

6.1.6 Bounds

The linearization of our QPs can have a negative effect if the perturbation of a body's orientation is large. For very small change in angle, a linearized rotation becomes acceptably accurate. For arbitrary change in angle, the solution to an individual QP may result in positions and orientations that slightly overlap the bodies, even if all close pairs and critical vertices are known. Each subsequent QP re-asserts the non-overlap constraints and tries to fix the overlap. If the iterated QPs converge, then they must therefore 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,

$$d = \frac{1}{2}\left(\max_{\mathbf{q}_a \in A} \mathbf{n} \cdot \mathbf{q}_a + \min_{\mathbf{q}_b \in B} \mathbf{n} \cdot \mathbf{q}_b\right), \tag{6.20}$$

works in all three cases: separated, touching, overlapping. Due to the linearized constraints, *i.e.* the linearized, approximative expression of rotation, 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 $\Delta\boldsymbol{\phi}$, the perturbation of the orientation, to lie within the interval $[-0.1, +0.1]$ in radians. This allows only perturbations in orientation that are acceptable and can be calculated with linearized rotation. Bounding the components of $\Delta\boldsymbol{\phi}$ this way has so far prevented an infeasible configuration and has ensured convergence to a local minimum of the objective. We summarize our QP-model for the position update in Appendix A.3.1.

6.2 Momentum Update

We have introduced the position update algorithm, and we have seen how quadratic programming can be used to find non-overlapping body positions. However, bodies will be in contact. The position update deliberately puts bodies in an aligned, contacting arrangement. As body movement is de-synchronized, contacts that would have happened during a frame time are synchronized at the end of the frame, *i.e.* there will be many simultaneous collisions. We determine contacts as described in Section 3.2.

It is an advantage of our OBA technique that it is designed in a highly modular fashion. Tested components from existing simulation software can be reused. Although it is true that any analytic algorithm for momentum calculation would work, we devise a

new QP-based collision resolution algorithm [76], which we found to be very well suited for our needs and our simulations. This algorithm implements the concepts of Chapter 4, in particular Newton's impact model and the Coulomb friction model.

6.2.1 Pure QP-based Momentum Update

Traditionally, momentum update is done intuitively with a priority queue algorithm (PQ). Contacts are processed sequentially in the order of most negative relative velocity first. Say the minimum relative velocity is found for a collision c between body i and j. When c is resolved, all contacts that involve either body i or j have their relative velocity recalculated and the new minimum c must be remembered. Then, the new minimum c is bounced and so on. This process continues until no more collisions have negative relative velocity, *i.e.* they are separating. Collisions are handled sequentially. This process is very simple, but can take up more than 90% of the running time of a simulation if the number of collisions becomes large.

We argued in the introduction to this chapter that the number of collisions can go to infinity even for relatively simple scenes. The small time-step problem makes traditional simulation intractable. To alleviate the problem, people apply techniques and tricks, which are usually not published. A seemingly halted simulation due to rattling between bodies can be avoided by application of little bounces that create just enough separation between bodies to keep them moving. Although analytical methods claim to be exact, they often make use of such plausibility techniques in order to overcome problems. Even worse, this modified analytical method does not scale arbitrarily to large stacks. It allowed us to simulate about 10 cubes without visual artifacts. For larger numbers of bodies, the bounces need to be of greater magnitude and can introduce visible simulation instabilities. However,

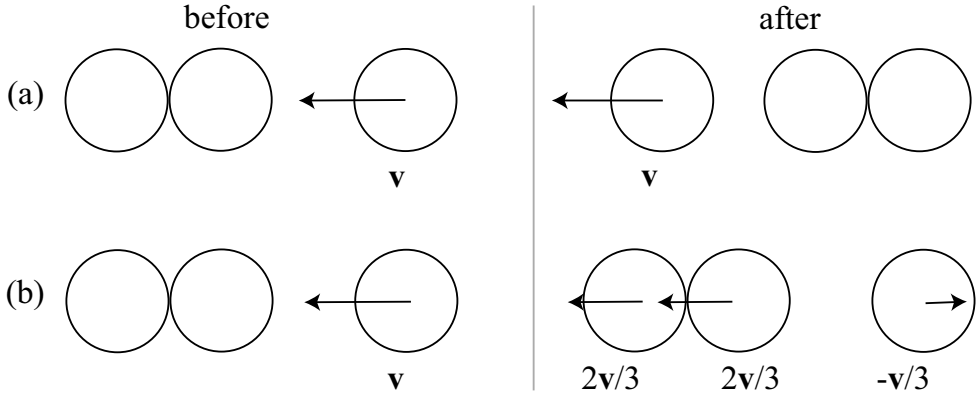


Figure 6.13: (a) Intuitive momentum conservation, (b) conservation of momentum in a simultaneous model.

even for the 10 cube stack, OBA performed almost two orders of magnitude faster than an analytical algorithm with the plausibility technique added. At any rate, application of bounces will not work for certain simulations. If slopes are involved, application of bounces will prevent bodies from resting on a slope under friction. Instead, bodies will slowly slide down a slope in a hopping motion.

Our initial implementation used a traditional sequential model to handle collisions. Although our experiments with this model were quite satisfactory when the number of collisions is moderate, we also experienced that the computational effort for sequential models could be over 90% of the running time when large numbers of collisions are at hand. This is mentioned in the literature [7]. Although simultaneous models exhibit some quirks in their behavior regarding how they conserve momentum (Fig. 6.13), we opted for simultaneous impulses to avoid the high computational effort for dealing with large numbers of collisions in a sequential model. The impulses are the solutions to a QP.

Collisions with Coulomb friction have been well researched [15, 17, 45, 74]. The two parameters for our impulses are the friction coefficient μ and the coefficient of restitution ϵ. Both can be understood as material constants. To derive the constraints of our QP, let

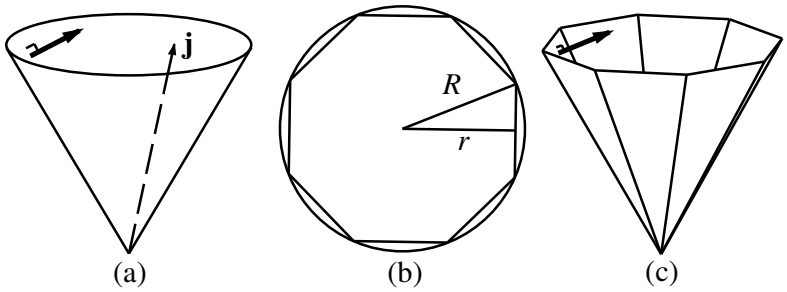


Figure 6.14: Friction cone and regular octagonal approximation. Ratio r/R of inner to outer radius is $\cos \pi/8$.

q be a contact point for bodies **A** and **B**, and let **n** be the normal to the separating plane. The vectors **n**, $\mathbf{n}_x$, and $\mathbf{n}_y$ (Sec. 6.1.3) form the right-handed *collision frame*. We re-state the Coulomb model from Section 4.3:

$$(\mathbf{n}_x \cdot \mathbf{j})^2 + (\mathbf{n}_y \cdot \mathbf{j})^2 \leq \mu^2 (\mathbf{n} \cdot \mathbf{j})^2 \quad \text{and} \quad \mathbf{n} \cdot \mathbf{j} \geq 0. \tag{6.21}$$

It states that any collision impulse **j** must lie within a cone, *i.e.* the tangential frictional impulse at a contact point is at most μ times the non-negative (no "stickiness") normal component. The normal **n** points up along the cone axis in Figure 6.14a, and the coordinate origin is at the cone tip. This constraint is convex but non-linear. To formulate a quadratic programming problem, we have to linearize the cone. We can achieve this easily by approximating the cone with a polygonal pyramid. Currently, we use an eight-sided pyramid. To do so, we inscribe a polygon with eight equal edges into the base of the pyramid as seen in Figure 6.14b. We yield an eight-sided pyramid as in Figure 6.14c. The collision frame has its origin centered at the cone tip, and the collision normal **n** points up in Figure 6.14c. The original non-linearized cone (Fig. 6.14a) has slope $1/\mu$.[2] The linearized cone's sides have slope $(\mu \cos \pi/8)^{-1} > \mu^{-1}$. To confine the impulse **j** to the inside of the linearized friction cone, we calculate inward pointing normal vectors to the

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

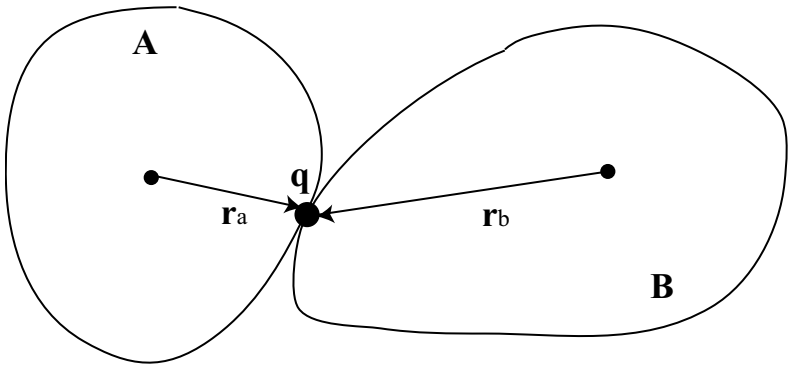


Figure 6.15: Contact levers and derivation of relative contact normal velocity.

cone sides

$$\mathbf{u}_h = \mu \cos\frac{\pi}{8}\mathbf{n} + \cos\frac{\pi h}{4}\mathbf{n}_x + \sin\frac{\pi h}{4}\mathbf{n}_y, \tag{6.22}$$

for $h = 0, 1, 2, \ldots, 7$.[3] The impulse $\mathbf{j}$ lies inside the linearized cone if and only if it lies inside all these sides:

$$\mathbf{u}_h \cdot \mathbf{j} \geq 0, \qquad \text{for} \qquad h = 0, 1, 2, \ldots, 7. \tag{6.23}$$

We have now eight linear inequalities per collision, which can be implemented in a QP to represent the Coulomb friction model.

We also have to address restitution by bouncing. We follow Newton's impact model from Section 4.2.1. This model relates contact velocities before and after impact. It states that a part ϵ of the colliding momenta has to be reflected. We introduce the variables $\mathbf{r}_a = \mathbf{q} - \mathbf{x}_a$ and $\mathbf{r}_b = \mathbf{q} - \mathbf{x}_b$ for the contact levers on bodies $\mathbf{A}$ and $\mathbf{B}$ respectively (Fig. 6.15), and calculate the (scalar) relative contact normal velocity $v(\mathbf{q})$ between bodies $\mathbf{A}$ and $\mathbf{B}$ at contact $\mathbf{q}$ following Equation 4.7:

$$v(\mathbf{q}) = \mathbf{n} \cdot ((\mathbf{v}_b + \boldsymbol{\omega}_b \times \mathbf{r}_b) - (\mathbf{v}_a + \boldsymbol{\omega}_a \times \mathbf{r}_a)). \tag{6.24}$$

By convention, the impulse $\mathbf{j}$ is applied positively to $\mathbf{B}$ and negatively to $\mathbf{A}$. It acts on

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

the bodies linearly by adding it to the bodies' linear momenta. In the same fashion, the resulting torsional impulse 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 to $\mathbf{v}_b$ and $\boldsymbol{\omega}_b$

$$\mathbf{v}_b^{new} = \mathbf{v}_b + m_b^{-1}\mathbf{j} \qquad \text{and} \qquad \boldsymbol{\omega}_b^{new} = \boldsymbol{\omega}_b + \mathbf{I}_b^{-1}(\mathbf{r}_b \times \mathbf{j}), \tag{6.25}$$

and subtracts from $\mathbf{v}_a$ and $\boldsymbol{\omega}_a$

$$\mathbf{v}_a^{new} = \mathbf{v}_a - m_a^{-1}\mathbf{j} \qquad \text{and} \qquad \boldsymbol{\omega}_a^{new} = \boldsymbol{\omega}_a - \mathbf{I}_a^{-1}(\mathbf{r}_a \times \mathbf{j}). \tag{6.26}$$

The relative contact velocity before impact is a constant and can be easily calculated with Equation 6.24. The relative contact velocity after impact can be calculated depending on $\mathbf{j}$ with Equations 6.24, 6.25, and 6.26. Just express the velocities after impact with Equations 6.25 and 6.26, and plug these into Equation 6.24. Let v^- denote the relative contact normal velocity $v(\mathbf{q})$ (Eq. 6.24) before any impulses are applied and let v^+ denote the relative contact normal velocity $v(\mathbf{q})$ after all the impulses are applied and $\mathbf{v}_a$, $\boldsymbol{\omega}_a$, $\mathbf{v}_b$, and $\boldsymbol{\omega}_b$ have been updated. In the following, we adopt some of Baraff's ideas [7]. If the bodies were not really colliding ($v^- \geq 0$) before the collision, then they must still not be colliding after the collision, else they must semi-elastically "bounce":

$$\text{if } \; v^- \geq 0 \; \text{ then } \; v^+ \geq 0 \; \text{ else } \; v^+ \geq -\epsilon \cdot v^-. \tag{6.27}$$

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 $\boldsymbol{\omega}$. 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 $\boldsymbol{\omega}$ change as a result of impulses according to Equations 6.25 and 6.26. It has one equality constraint per contact to relate v^+ to the updated body velocities according to Equation 6.24. It has

eight inequalities per contact to constrain $\mathbf{j}$ to the linearized friction cone according to Equation 6.23. Finally, it has one more linear inequality per contact point to implement the bounce according to Equation 6.27.

The objective of the QP is the total kinetic energy after application of impulses, which is a positive definite quadratic function of the $\mathbf{v}$'s and $\boldsymbol{\omega}$'s. For i, the index over all bodies, we have the objective:

$$\sum_i \frac{1}{2} m_i \mathbf{v}_i^2 + \frac{1}{2} \boldsymbol{\omega}_i^T \mathbf{I}_i \boldsymbol{\omega}_i. \tag{6.28}$$

In a way, we ask the system to diminish a maximum amount of kinetic energy as a result from friction. Without this objective, the QP's linear constraints permit energy to *increase*. Increasing energy through friction is entirely non-physical. Minimizing the kinetic energy complies with the laws of thermodynamics, which prohibit the existence of a perpetual motion machine. We assume that the physical system satisfies all the constraints and also converts as much kinetic energy to heat as possible. A block that slides down a slope will not pick up momentum, but eventually comes to a rest this way. Minimizing the kinetic energy has additionally a nice effect on the stability of our simulations. It moves a slowly moving system to rest and thus results in a very stable simulation. The momentum update QP is summarized in Appendix A.3.2.

6.2.2 Hybrid Momentum Update

The QP-based algorithm described in the previous section avoids the problem with inefficiency that was observed for traditional PQ momentum update [7]. It generates nice looking simulations of stacks and other scenes. However, in some cases it can be unrealistic, such as for the office toy pendulum where sequential bouncing has to me modeled.

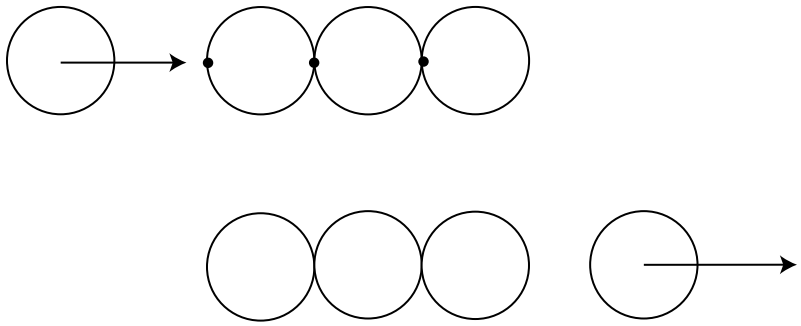


Figure 6.16: The office toy with 4 spheres: modeling 3 bounces (black dots) with the PQ will produce realsitic simulation.

The QP algorithm alone will not generate the text book physics behavior of the pendulum, because it does not conserve momentum in an intuitive way (Fig. 6.13). To address this, we implemented a hybrid of our QP algorithm with the standard PQ.

We have implemented a hybrid, which runs the PQ "for a while" until the QP-based algorithm finishes off. We can specify a number n of collisions to be resolved with PQ that ensures that the office toy is modeled properly. For m spheres in the office toy, $n = m - 1$ bounces with PQ will produce the desired outcome (Fig. 6.16). Coulomb friction in the PQ is implemented in the traditional way with Equation 4.28.

Specifying n has another advantage. Solving a QP has a certain overhead if only a few true collisions ($v^- < 0$) are at hand. Invoking the QP solver, building the constraints, and solving the QP comes at a cost. For only a few collisions, the PQ is much faster. If the number of actual collisions is less than n, PQ alone will finish the bouncing very cheaply. Therefore, we call the QP momentum update only if necessary. Currently, we ask the user to specify the value of n depending on the scene. It usually works well to set n equal to the number of bodies for scenes with fewer than 50 bodies and equal to 10% of the number of contacts for scenes with more than 50 bodies.

There is a hidden danger in handling friction in the traditional fashion following

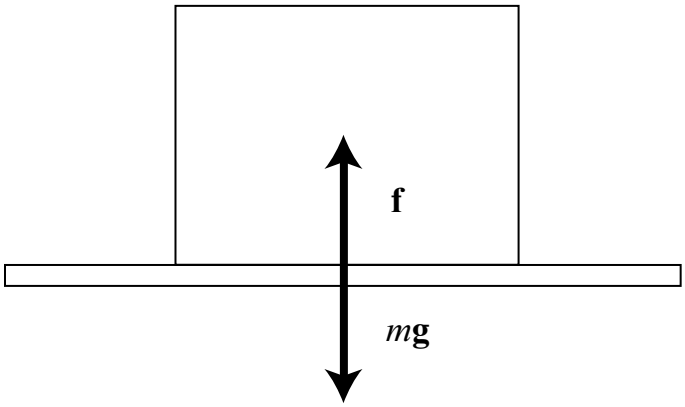


Figure 6.17: The supporting surface has to counteract gravity to keep the body at equilibrium, $\mathbf{f} = -m\mathbf{g}$.

Equation 4.28. It can increase kinetic energy when bodies have a large normal velocity and small tangential velocity. This could be addressed by using the QP momentum update even for individual pairs in the PQ. Unfortunately, due to computational overhead in solving QPs, this is too expensive. It would be an elegant solution to devise special code that solves small QPs. Each small QP represents one collision only. We could then treat individual bounces in the PQ efficiently without increasing kinetic energy.

6.3 Force or Acceleration Calculation

After collisions are resolved with bounces, resulting static contacts ($v^+ = 0$) must have contact forces computed to prevent bodies from sinking into each other. Consider a weight sitting on a surface. The supporting surface has to press up against gravity just strong enough to keep the weight at an equilibrium (Fig. 6.17). As with the momentum update, any algorithm for contact force calculation can be employed, *e.g.* Baraff's pivoting scheme [9]. It has been stated [52] 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. We base our reasoning here on Sections 4.2.2 and 4.3.

The goal of our work is to simulate crowded scenes of rigid bodies. Directly linked to large numbers of bodies are degeneracies in the contact geometry. If a scene is complicated, most frames will have degeneracies. It is not true that degeneracies are rare enough to be neglected. Baraff states that pivoting [9] can take a number of steps with size zero if degeneracies are present. It is therefore not a good idea to hope for the number of degeneracies to be negligible if, in fact, this will not be the case. A simulator has to be able to handle degeneracies in a reliable way.

By using QP, we separate the physics from the numerics. Our work is to formulate the problem. We then use the best available QP solver. Mathematical programming has an immense history of research. Very stable and reliable solution algorithms are known that have the ability to cope with degeneracies reliably. That way, we do not need to worry about explicit handling of degeneracies as it is taken care of in the *black box* QP solver module.

We made several attempts at the contact problem. After refreshing some background, we will introduce an algorithm for the frictionless case. We suggest an extension to friction, which unfortunately exhibits some efficiency problems. Therefore, we finally describe a purely impulsive algorithm which does not have the same problems as physically correct contact force calculation, but allows an approximate, visually convincing solution to the problem.

6.3.1 Force QP

Because solving the contact problem without friction is so much simpler, we first build a QP-based algorithm for the frictionless case. We reuse the collision frame from Section 6.1.3. Section 4.2.2 developed all necessary equations.

Frictionless Case

Contact forces have to keep bodies from accelerating into each other, *i.e.* the relative normal acceleration at all contacts must be positive. Contact forces and resulting torques change a body's linear acceleration $\mathbf{a}$ and its angular acceleration $\boldsymbol{\alpha}$. If we know these accelerations, we can calculate the relative contact normal acceleration $a^{\perp}$ (scalar). Recall that the relative contact normal acceleration of contacting points $\mathbf{q}_a$ and $\mathbf{q}_b$ between bodies $\mathbf{A}$ and $\mathbf{B}$ was calculated with Equation 4.8:

$$a^{\perp} = \mathbf{n} \cdot (\ddot{\mathbf{q}}_b - \ddot{\mathbf{q}}_a) + 2\dot{\mathbf{n}} \cdot (\dot{\mathbf{q}}_b - \dot{\mathbf{q}}_a). \tag{6.29}$$

The body velocities are known after the momentum update. Hence, the constant, velocity-dependent part in Equation 6.29 can be immediately calculated. We assume that the two bodies are at positions $\mathbf{x}_a$ and $\mathbf{x}_b$. Therefore, we have contact levers $\mathbf{r}_a = \mathbf{q}_a - \mathbf{x}_a$ and $\mathbf{r}_b = \mathbf{q}_b - \mathbf{x}_b$ for body $\mathbf{A}$ and $\mathbf{B}$ respectively. We rewrite Equation 4.17 and arrive at the following expression for $a^{\perp}$:

$$\begin{aligned} a^{\perp} \;=\; & \mathbf{n} \cdot (\mathbf{a}_b + \boldsymbol{\alpha}_b \times \mathbf{r}_b - \mathbf{a}_a - \boldsymbol{\alpha}_a \times \mathbf{r}_a) + \\ & \mathbf{n} \cdot (\boldsymbol{\omega}_b \times (\boldsymbol{\omega}_b \times \mathbf{r}_b) - \boldsymbol{\omega}_a \times (\boldsymbol{\omega}_a \times \mathbf{r}_a)) + \\ & 2\dot{\mathbf{n}} \cdot (\dot{\mathbf{q}}_b - \dot{\mathbf{q}}_a). \end{aligned} \tag{6.30}$$

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

$$a^{\perp} \geq 0. \tag{6.31}$$

This simple and intuitive constraint can be easily implemented in a QP together with constraints according to Equations 4.19, 4.21, 4.24, and 4.25 to express the body accelerations in terms of the contact forces. To complete the QP, a suitable objective must be devised.

Our first experiments were conducted with a formulation that calculates forces explicitly and uses the sum of squared forces for the objective. This seemingly natural objective turns out to have a flaw in that the complementarity condition $a^{\perp} \cdot f = 0$ is not necessarily always fulfilled for all contacts. Here, f is the force magnitude for the contact at hand. We came up with the idea to model implicit contact forces by formulating a QP, which is entirely based on accelerations. This QP will not calculate body accelerations according to Equations 4.19 through 4.25 but derives them directly without ever generating a solution for the contact forces. Furthermore, the non-interpenetration constraint from the combined Equations 6.30 and 6.31 will be approximated by a zero velocity assumption. To introduce the ideas, we start with a thought experiment.

First, due to Newton's axiom (Eq. 2.18), calculating the total force and torque acting on a body is equivalent to calculating the body's accelerations. The total body force and torque are a result of the sum of external forces and internal contact forces. We have currently only gravity as external potential but could easily add others, such as magnetism or wind. Devising an algorithm that finds the total force, torque, and associated accelerations on a body is equivalent to calculating contact forces and summing them up.

Following Newton's axiom, instead of computing contact forces we can therefore attach imaginary springs to each body and find the equivalent total force on the body as

exerted by the springs. To account for force and torque, we attach a linear and a torsional spring to the body. The force and torque exerted by the springs are proportional to the linear displacement $\mathbf{s}$ and the angular displacement $\boldsymbol{\sigma}$. The proportionality factor is given by linear and rotational spring constants. For reasons that will become clear, we set the linear spring constant equal to m, the mass of the body, and the angular spring constant equal to $\mathbf{I}$, the body's inertia tensor:

$$\mathbf{f} = -m\mathbf{s} \quad \text{and} \quad \boldsymbol{\tau} = -\mathbf{I}\boldsymbol{\sigma}. \tag{6.32}$$

Imagine a contacting system of bodies which is held at rest by springs. In other words, we attach springs that bring all body motion to a halt. The property of interest is the total energy of this system under gravity (or other external potentials). Subscript i indicates a particular property of the i-th body. For k bodies, the objective is:

$$\sum_{i=1}^{k} \frac{1}{2} m\mathbf{s}_i \cdot \mathbf{s}_i + \frac{1}{2} \boldsymbol{\sigma}_i^T \mathbf{I}_i \boldsymbol{\sigma}_i - m_i \mathbf{g} \cdot \mathbf{s}_i. \tag{6.33}$$

Equation 6.33 is derived as follows. We measure the energy stored in a spring by the absolute value of the work it takes to dilate the spring. The latter is the necessary force (torque) integrated over the path it acts. We subscript linear quantities by l and rotational (also angular or torsional) quantities by r:

$$W_l = \mathbf{f} d\mathbf{s} \quad \text{and} \quad W_r = \boldsymbol{\tau} d\boldsymbol{\sigma}. \tag{6.34}$$

In addition, we need to account for gravity. The work to dilate a weight on a spring against gravity is equal to $m\mathbf{g} \cdot \mathbf{s}$. The minus sign in Equation 6.33 appears because gravity points "down", and hence we need to do work against the direction of gravity. Equation 6.33 describes an artificial energy potential introduced by attaching springs to the bodies. It represents the total energy stored in the springs. To yield a static system, the springs have to keep the balance against the external potential, *i.e.* gravity.

We replace the original non-interpenetration constraint (Eq. 6.30 and 6.31) by the assumption that bodies may not *displace* into each other. Assume bodies **A** and **B** in contact:

$$\mathbf{n} \cdot (\mathbf{s}_b - \mathbf{s}_a + \boldsymbol{\sigma}_b \times \mathbf{r}_b - \boldsymbol{\sigma}_a \times \mathbf{r}_a) \geq 0. \tag{6.35}$$

This is a measure for interpenetration of a static system suspended by springs as a result from spring displacements. The system is static because we minimize the energy objective and therefore it is at minimum energy. Since the system is static, the springs must apply forces $\mathbf{F}$ and torques $\mathbf{T}$ which yield zero total body force and torque. We use index i to denote the force $\mathbf{f}_i$ and torque $\boldsymbol{\tau}_i$ on a body at a particular contact i:

$$\mathbf{F} + m\mathbf{g} + \sum_i \mathbf{f}_i = 0 \quad \text{and} \quad \mathbf{T} + \sum_i \boldsymbol{\tau}_i = 0. \tag{6.36}$$

Solving a system with objective as in Equation 6.33 and constraints according to Equation 6.35 will yield a non-interpenetrating, static system due to the above reasoning.

We have calculated forces $\mathbf{F}$ and torques $\mathbf{T}$ that will make the original system without springs static, hence bringing its motion to a halt. Now, take this reasoning a step further. Apply to the static system forces $-\mathbf{F}$ and torques $-\mathbf{T}$. First, the resulting accelerations are $-\mathbf{F}/m$ and $-\mathbf{T}/\boldsymbol{I}$ because the system was initially at rest. Furthermore, forces $\mathbf{F}$ and $-\mathbf{F}$ cancel out, and the same goes for torques $\mathbf{T}$ and $-\mathbf{T}$. Compared to the original system without springs, this system does therefore not have any additional forces and torques applied. We are back at the original system without any springs. However, we know that bodies comply with the non-interpenetration condition of Equation 6.35. We have thus generated solutions to the contact problem.

A closer look reveals that according to Newton's axiom $\mathbf{f} = m\mathbf{a}$ and Equation 6.32, force $-\mathbf{F} = m\mathbf{s}$ generates acceleration $\mathbf{a} = \mathbf{s}$. Similarly, the angular acceleration from a

torque $-\mathbf{T} = \mathbf{I}\boldsymbol{\sigma}$ is $\boldsymbol{\alpha} = \boldsymbol{\sigma}$. Things fall into place nicely and we realize we can formulate the whole problem by substituting accelerations $\mathbf{a}$ and $\boldsymbol{\alpha}$ for $\mathbf{s}$ and $\boldsymbol{\sigma}$ in Equations 6.33 and 6.35. However, compare with Equation 6.30 and note that this system makes the assumption of zero body velocities. It turns out though that we can approximate the original system by it and still yield a visually convincing simulation.

We formulate a QP that implements this system. The QP has six variables per body: the components of $\mathbf{a}$ and $\boldsymbol{\alpha}$. It has one variable per contact point for $a^{\perp}$. We add one linear inequality constraint per contact point according to Equation 6.35. Assume contacting bodies $\mathbf{A}$ and $\mathbf{B}$:

$$\mathbf{n} \cdot (\mathbf{a}_b - \mathbf{a}_a + \boldsymbol{\alpha}_b \times \mathbf{r}_b - \boldsymbol{\alpha}_a \times \mathbf{r}_a) = a^{\perp} \quad \text{and} \quad a^{\perp} \geq 0. \tag{6.37}$$

Subscript i indicates a particular property of the i-th body. For k bodies, the objective becomes:

$$\sum_{i=1}^{k} -m_i \mathbf{g} \cdot \mathbf{a}_i + \frac{1}{2} m_i \mathbf{a}_i \cdot \mathbf{a}_i + \frac{1}{2} \boldsymbol{\alpha}_i^T \mathbf{I}_i \boldsymbol{\alpha}_i. \tag{6.38}$$

We verified empirically that this algorithm gives the exact force solutions for the frictionless case. Note the interesting fact that the entire QP is formulated based on accelerations. We immediately yield body accelerations that ensure non-interpenetration. The contact forces do not appear: they are implicit. We summarize the frictionless QP in Appendix A.3.3.

Friction Case

Given our success with the frictionless case, we tried to follow up with an artificial frictionless system which modeled the system with friction. We wanted a solution with implicit forces that satisfied Coulomb's law. To develop the ideas, we start again with a thought

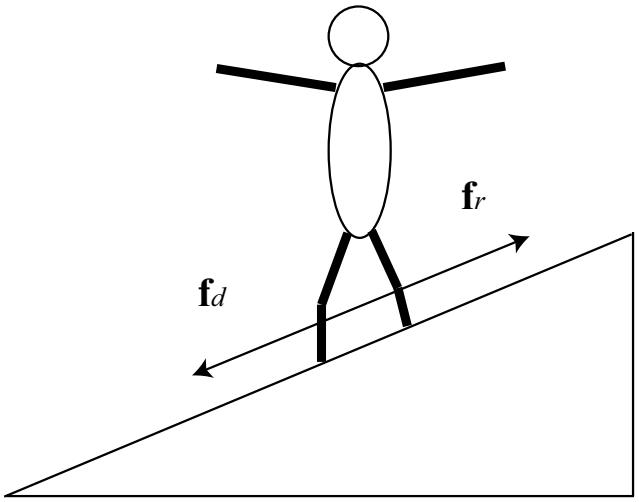


Figure 6.18: The downward force $\mathbf{f}_d$ is cancelled out by friction: $\mathbf{f}_d = -\mathbf{f}_r$.

experiment. Imagine standing on the floor. A force must point straight up to counteract gravity. When standing on a slope, a frictional force must prevent slipping, unless the slope's angle becomes very large (Fig. 6.18). The sum of forces has to lie within the friction cone (Sec. 4.3), *i.e.* the Coulomb law must be satisfied.

We can express the net contact force by introducing a frictionless *acceleration cone* as depicted in Figure 6.19. Each contact is modeled by a cup and a cone with sides perpendicular to the actual friction cone. The axes of cup and cone coincide with the friction cone's axis and therefore with the collision normal. Since this system is frictionless, all contact forces between the cup and cone are perpendicular to the acceleration cone sides. The net force is a combination of these forces and will therefore lie within the friction cone, *i.e.* it will comply with the Coulomb law.

Under the traditional Coulomb friction model, the force from friction is equal to the normal force times the friction coefficient, $\mu m\mathbf{g}\cos\alpha$. Therefore, standing on a slope without slipping means the frictional force $\mu m\mathbf{g}\cos\alpha$ is at least equal to or larger than the

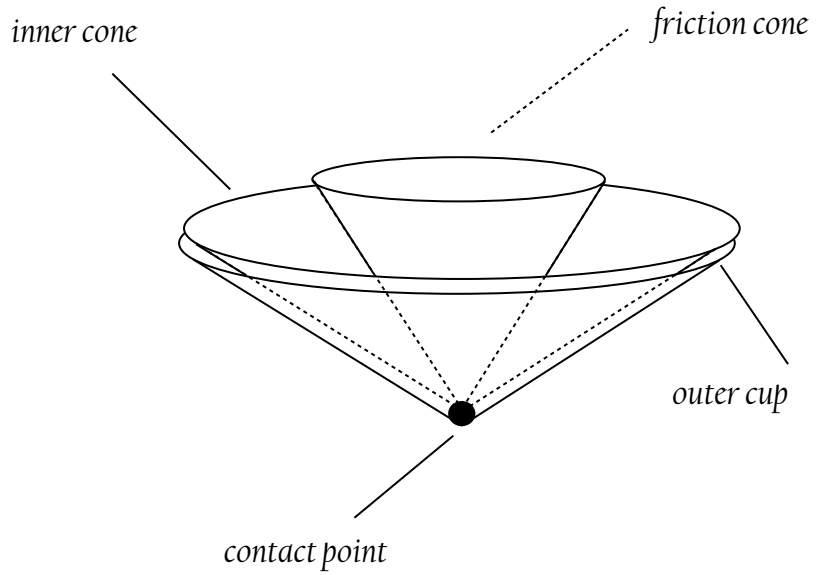


Figure 6.19: The *acceleration cone*: we model each contact by a cup and a cone. They are perpendicular to the familiar force cone.

force component parallel to the slope, $m\mathbf{g}\sin\alpha$ (Fig. 6.20):

$$\begin{aligned} \mu m\mathbf{g}\cos\alpha &\geq m\mathbf{g}\sin\alpha \Rightarrow \\ \mu &\geq \tan\alpha. \end{aligned} \tag{6.39}$$

With the new concept of an acceleration cone and a cup, standing on a slope without slipping becomes equivalent to the lower acceleration cone side being not below the horizontal. We distinguish three cases: above, on, and below the horizontal (Fig. 6.21). Above means the cone is firmly pressed into the cup and the contact is strong, $\mu > \tan\alpha$. On corresponds to a contact for which friction is just strong enough to not break, $\mu = \tan\alpha$. The final case, where the lower side ends up below the horizontal, the contact immediately breaks and $\mu < \tan\alpha$. Note as the slope angle becomes steeper, the cone orientation becomes more tilted in order for the cone axis to stay aligned with the collision (slope surface) normal.

The remaining task is to implement a QP that uses acceleration cone constraints to solve the contact problem with friction. First, it is clear that we need to calculate the full

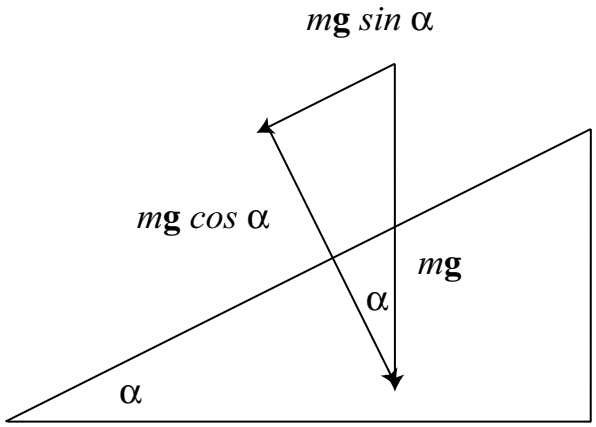


Figure 6.20: Deduction of forces on a frictional slope: friction has to counteract the downward force $m\mathbf{g}\sin\alpha$ to prevent slipping.

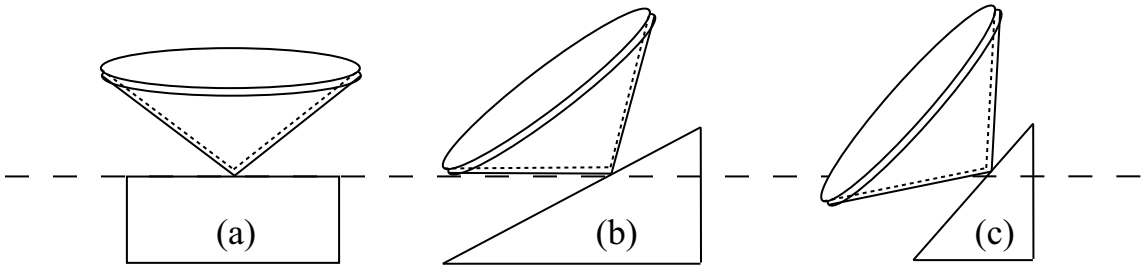


Figure 6.21: Three possible cases for the acceleration cone: (a) firm contact, (b) just strong enough, and (c) slipping contact.

contact acceleration vector following Equation 6.29:

$$\mathbf{a} = \begin{pmatrix} a_x \\ a_y \\ a^{\perp} \end{pmatrix} = \begin{pmatrix} \mathbf{n}_x \cdot (\ddot{\mathbf{q}}_b - \ddot{\mathbf{q}}_a) + 2\dot{\mathbf{n}}_x \cdot (\dot{\mathbf{q}}_b - \dot{\mathbf{q}}_a) \\ \mathbf{n}_y \cdot (\ddot{\mathbf{q}}_b - \ddot{\mathbf{q}}_a) + 2\dot{\mathbf{n}}_y \cdot (\dot{\mathbf{q}}_b - \dot{\mathbf{q}}_a) \\ \mathbf{n} \cdot (\ddot{\mathbf{q}}_b - \ddot{\mathbf{q}}_a) + 2\dot{\mathbf{n}} \cdot (\dot{\mathbf{q}}_b - \dot{\mathbf{q}}_a) \end{pmatrix}. \tag{6.40}$$

Instead of the combined inequality from Equations 6.30 and 6.31, add the equalities from Equation 6.40 to the frictionless QP for each contact and also the acceleration cone constraint with the following modification. We introduce *target accelerations*. For each body, set $\mathbf{a}_t = -\Delta t^{-1}\mathbf{v}$ and $\boldsymbol{\alpha}_t = -\Delta t^{-1}\boldsymbol{\omega}$. These are the accelerations that will bring the body to a halt in one time-step $\Delta t = 1/30$ sec. For each contact $\mathbf{q}$, plug these values into Equation 6.40 to calculate $\mathbf{a}(\mathbf{q})$. Set $\hat{\mathbf{a}}(\mathbf{q}) = \mathbf{a}(\mathbf{q})$. Add an acceleration cone constraint with

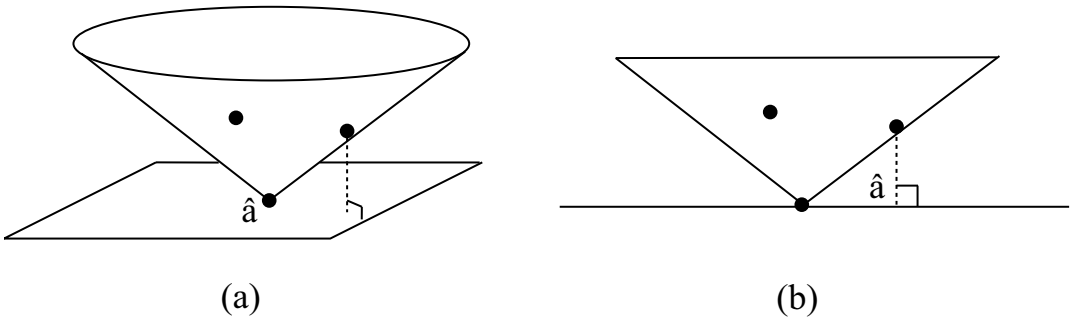


Figure 6.22: Acceleration cone update: if $\mathbf{a}$ lies on side of cone, new $\hat{\mathbf{a}}$ is projection to $\mathbf{n}_x, \mathbf{n}_y$ plane. (a) points out the current $\hat{\mathbf{a}}$, (b) shows the updated $\hat{\mathbf{a}}$ in a 2-dimensional cut through the cone for clarification.

slope μ, axis $\mathbf{n}$, and tip at $\hat{\mathbf{a}}(\mathbf{q})$.

To implement a QP, we linearize the acceleration cone constraint in the same way that we linearized the force cone in Section 6.2. Calculate inward pointing face normals for the sides of the linearized acceleration cone. For $h = 0, 1, 2, \ldots, 7$:

$$\left(\mu^{-1} \cos\frac{\pi}{8}\mathbf{n} + \cos\frac{\pi h}{4}\mathbf{n}_x + \sin\frac{\pi h}{4}\mathbf{n}_y\right) \cdot (\mathbf{a}(\mathbf{q}) - \hat{\mathbf{a}}(\mathbf{q})) \geq 0. \tag{6.41}$$

Note that $\mathbf{a}(\mathbf{q})$ is a variable. We solve this system for the total body accelerations and the relative contact accelerations $\mathbf{a}(\mathbf{q})$ at each contact. There are three possible cases for the relative contact acceleration $\mathbf{a}(\mathbf{q})$. Figure 6.22 indicates the possible types of position for $\mathbf{a}(\mathbf{q})$: at the tip, in the interior (Fig. 6.22, left contact), or on the side (Fig. 6.22, 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. 6.22), *i.e.* we project it straight down onto the $\mathbf{n}_x$, $\mathbf{n}_y$ plane:

$$\hat{\mathbf{a}}(\mathbf{q}) = \mathbf{a}(\mathbf{q}) - (\mathbf{n} \cdot \mathbf{a}(\mathbf{q}))\mathbf{n}. \tag{6.42}$$

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 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 successive system will have a smaller value of the objective (Eq. 6.38) and this iteration must converge.

Unfortunately, the algorithm often requires many iterations to converge and is therefore not superior to Baraff's pivoting [9] 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. The structure of each individual QP as solved in the iterated algorithm for friction is summarized in Appendix A.3.3.

6.3.2 Impulsive Static Contact Response

As mentioned in Section 4.3, we experienced static contact forces with friction can be tricky when we developed our QP-based approach of Section 6.3.1. We experimented with several methods to calculate forces and verified that it is a non-trivial problem. Mirtich's work used impulsive forces exclusively [59], and other researchers have stated that impulsive forces can be applied when non-impulsive contact forces are not easily calculated [8, 70, 80, 86]. We realized that our QP momentum update can be modified and used in the vein of an impulse-based approach to resolve static contacts very reliably, efficiently, and extremely stably.

Under true physics, the force of gravity accelerates the bodies. A force arises at each contact that provides non-negative relative contact normal acceleration $a^{\perp} \geq 0$. Since the impulse update ensures non-negative relative normal velocity $v^{\perp} \geq 0$ at the contacts, the integration of $a^{\perp}$ ensures that $v^{\perp}$ remains non-negative. Contact forces do no work on the bodies, *i.e.* they do not change the body energies. In other words, the contact forces are

"just strong enough" to prevent negative contact normal acceleration.

We mimic this behavior as follows. First, our algorithm integrates the accelerations without regard to contacts: add Δt times the acceleration of gravity to the linear velocity of each body. At this point, the relative contact normal velocities may have changed from non-negative to negative. The algorithm then applies small impulses at all contacts to make all relative contact normal velocities non-negative again. It does so by solving a QP that is the momentum update QP with ϵ set to zero but otherwise with the same objective and constraints. Consider the example of a single body at rest on the ground. Initially, its relative contact normal velocity $v^{\perp}$ is zero. As a result of gravity, $v^{\perp} < 0$ after the next time-step. The QP generates an impulse at the contact sufficiently strong to set $v^{\perp}$ equal to zero again. Since $\epsilon = 0$, the body will not "bounce up". Because the QP minimizes kinetic energy, it will not give the body a spurious tangential impulse or otherwise violate the laws of physics.

This reduces our system to impulse-based physics and works very well in practice. The simulations are stable and efficient. Note that impulsive static contact resolution does *not* make use of the priority queue momentum update. It is only necessary to solve a single QP to calculate the impulses for static contact resolution. Other methods use pivoting strategies [9] or iterated QP [58] to calculate contact forces. A single QP implementation has a significant speed advantage over the latter.

Section 2.1.8 states that we use the simple Euler method to approximate integration. The Euler method in conjunction with our large time step Δt which we set equal to the frame time is in fact equivalent to impulsive forces:

$$\mathbf{j} = \mathbf{f}\Delta t. \tag{6.43}$$

On the other hand, our QP momentum algorithm is very much like an implicit method (Sec. 2.1.7). It calculates impulses under the constraints that bodies cannot interpenetrate, which is reminiscent of implicit integration's feedback character. A strictly impulsive algorithm is therefore a well founded way to handle static contacts in the OBA simulator. For a summary of the impulsive static contact QP refer to Appendix A.3.2.

We have now completely introduced the individual component algorithms of OBA. To summarize and give an overview of the developments so far, Table 6.2 gives the overall structure of our OBA algorithm in pseudo-code:

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

Table 6.2: The OBA simulation loop.

6.4 Joints

Animation of articulate bodies requires modeling of joints between partial bodies. There is a good introduction to multi-body physics in Armstrong and Green [2]. The simple standard text book example for articulate body motion is the multi-body pendulum (Fig. B.3). It is useful in this approach to follow a constraint-based model. Once the positions and velocities for each link are known, the links can be analyzed individually [63]. We handle joints, or more generally links between bodies, as bi-directional constraints. 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 bodies connected by a joint are not regarded as a close pair. Instead, we add a simple constraint to our position update algorithm to ensure

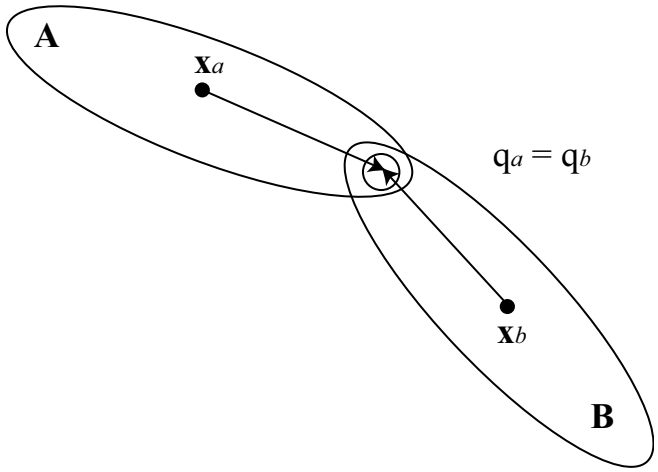


Figure 6.23: Two linked parts must stay connected at the join points.

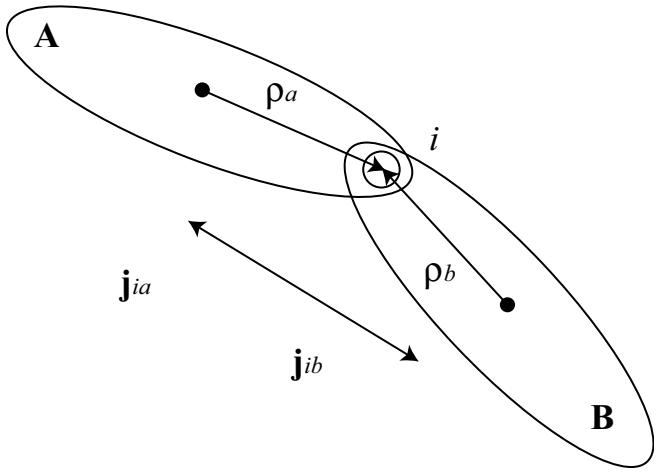


Figure 6.24: Equal and opposite attachment impulses.

that points on bodies where they are joined together (join points) are perturbed to the same coordinate in space (Fig. 6.23):

$$\mathbf{q}_a + \Delta\mathbf{x}_a + \Delta\mathbf{r}_a \times (\mathbf{q}_a - \mathbf{x}_a) = \mathbf{q}_b + \Delta\mathbf{x}_b + \Delta\mathbf{r}_b \times (\mathbf{q}_b - \mathbf{x}_b), \tag{6.44}$$

where $\mathbf{q}_a = \mathbf{q}_b$ is the common link point of bodies **A** and **B**. This way, they will stay connected.

We furthermore calculate attachment impulses to simulate realistic motion of multi-bodies [63]. The attachment impulses act on the bodies at the join points in a way that

ensures uniform, non-separating motion of the join points, *i.e.* equal velocities:

$$\mathbf{v}_{q_a} = \mathbf{v}_{q_b}. \tag{6.45}$$

We use notation $\mathbf{j}_{ik}$ for the attachment impulse that body k feels in joint i and ρ_{ik} for the linkage lever from body k's center of mass to joint i. If bodies **A** and **B** are linked in joint i, then we have the equality $\mathbf{j}_{ia} = -\mathbf{j}_{ib}$ (Fig. 6.24). At the absence of direct collisions, *i.e.* only attachment impulses, we can calculate the resulting body velocities with:

$$\begin{aligned}
\mathbf{v}_a^{new} &= \mathbf{v}_a + \sum_i \frac{\mathbf{j}_{ia}}{m_a}, \\
\mathbf{v}_b^{new} &= \mathbf{v}_b + \sum_i \frac{\mathbf{j}_{ib}}{m_b}, \\
\boldsymbol{\omega}_a^{new} &= \boldsymbol{\omega}_a + \sum_i \frac{\rho_{ia} \times \mathbf{j}_{ia}}{\mathbf{I}_a}, \\
\boldsymbol{\omega}_b^{new} &= \boldsymbol{\omega}_b + \sum_i \frac{\rho_{ib} \times \mathbf{j}_{ib}}{\mathbf{I}_b}.
\end{aligned} \tag{6.46}$$

According to Equation 6.45, we insert for each joint i connecting the bodies **A** and **B** the constraint:

$$\mathbf{v}_a^{new} + \boldsymbol{\omega}_a^{new} \times \rho_{ia} = \mathbf{v}_b^{new} + \boldsymbol{\omega}_b^{new} \times \rho_{ib}. \tag{6.47}$$

6.5 Non-Convex Bodies

Non-convex bodies (Fig. B.4) are implemented as collections of convex parts. It is a standard procedure in graphics and computational geometry to partition non-convex bodies into convex components. 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. In particular, the separation plane constraints have to be constructed with respect to pairs of

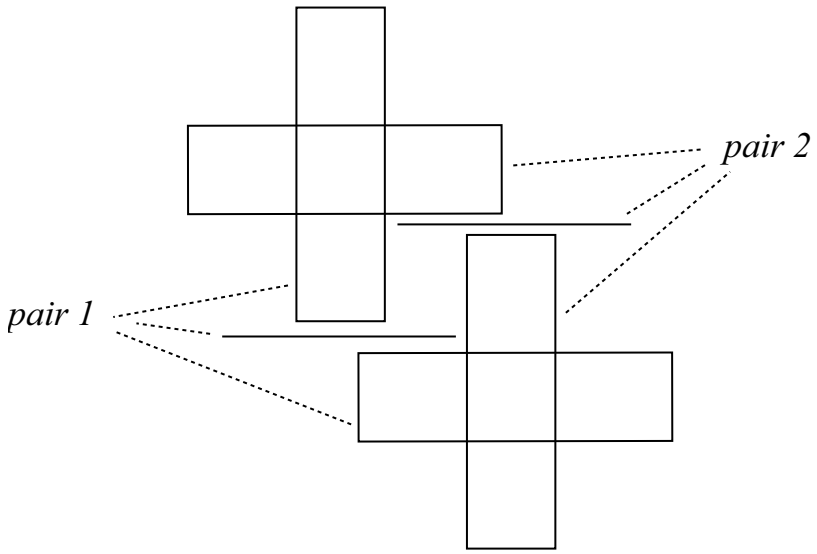


Figure 6.25: One pair of non-convex bodies, but two pairs of convex components with a separating plane each.

convex components (Fig. 6.25). The rest of the calculations, *i.e.* the momentum and force update can be made with respect to the non-convex bodies. We maintain mass, inertia, and center of mass for the whole bodies therefore.

6.6 Hybrid Position Update

It has been noted by other researchers that bodies can pass through each other if the simulation time-step is too large or the body velocities are fast [3, 60, 63]. Some earlier works ignore this problem and hope for the best. This is clearly not a reliable solution, although it works for most "reasonable" simulations. An unreasonable simulation would be for instance a scene with a bullet that hits a thin wall, passing through it in infinitesimal time. We don't simulate this kind of scenario every day, but we cannot exclude the possibility of a similar case entirely. Mirtich solves the problem properly by calculating an upper bound on the admissible time-step with conservative advancement [59]. Another way to handle

it is to employ four-dimensional space-time swept volume algorithms [20]. This approach is unfortunately prohibitively expensive for simulation purposes.

While OBA can handle most scenes, pure OBA is not suited for extremely fast bodies moving more than their own thickness in a single frame time. Up to now, we calculated new body positions by optimization under the objective to move bodies as close as possible to their target positions without overlap. When two bodies approach each other at very high velocity, their target positions might be highly overlapping, or the bodies might even go completely past each other. Using these targets will result in highly unrealistic final positions for the bodies. If bodies pass completely through each other, the algorithm determines there is no overlap to be minimized and chooses the targets as final positions. We implemented a hybrid algorithm of OBA and RD to solve the problem.

The hybrid algorithm introduces a heuristic that looks at a pair of colliding bodies and their relative velocities and thicknesses. Currently, we use as heuristic a limit on the body velocity. If a pair has relative velocity fast enough to interpenetrate more than a certain percentage (10%) of the smaller body's thickness, we need to add a mechanism for further processing of this pair. Instead of ignoring all collisions for this pair right away, we allow a maximum $c_{\max}$ on the number of bounces with a minimum collision velocity $v_{\min}$ for this pair to be resolved by a modified RD algorithm. 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 the RD collision list. In other words, that pair of bodies no longer "sees" each other and collision detection is turned off for this time-step. This modified RD is run until the next frame time. The output is a set of possibly overlapping target positions, which are then used in an optimization position update step to find the final positions. If the bodies of a pair are thick and slow, we consider this pair

not to be potentially "dangerous", *i.e.* we give it no further heightened attention.

First, note we assume that we can specify a $c_{\max}$ depending on the simulation at hand, which will make sure that colliding bodies lose enough energy with each bounce until they are slow enough to no longer pose any danger by passing through each other. The hybrid algorithm is reduced to pure OBA if $c_{\max} = 0$ (or $v_{\min} = \infty$): collision detection is turned off always and bodies simply follow their Newtonian trajectories without regard to collisions. For $c_{\max} = \infty$, the hybrid algorithm is the same as pure RD and every micro time-step will be executed, hence slowing the simulation eventually to a crawl. 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 1.1 a number of times equal to $c_{\max}$ times the number of colliding pairs.

By specifying $c_{\max}$ and $v_{\min}$, we have a way to trade off simulation speed for realism. Therefore, we can perform a simulation that is sufficiently realistic, yet 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 algorithm is realistic enough to generate 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.

After saying much about the benefits of OBA through the use of a fixed time-step, we need to point out that the hybrid method is not a drastic step back into the stone age of simulation. The hybrid re-introduces some micro-steps, but only a limited number. Furthermore, most scenes can be simulated by pure OBA, and therefore we almost never

need to make use of micro-steps that are smaller than the frame time.

6.7 The Need for Speed

The algorithm as presented so far can be used to generate stable and efficient simulations of several hundreds of bodies with high numbers of concurrent contacts. Yet, we found it to be too slow when the number of bodies is raised to over 1000. The running time of the algorithm is overall determined by the solution time for each QP and thus its individual size. As mentioned, we use our impulsive contact suite [76] in an implementation of OBA. OBA finds plausible non-overlapping body positions by inserting separating plane constraints for close pairs. It is clear that the QP size for the position update, and therefore its solution time, is determined by how many separating plane constraints we have, *i.e.* by how many close pairs there are in a scene. Collision and contact handling on the other hand clearly depend on the number of actual contacts and thus implicitly also on the number of close pairs.

It is a common approach to speed up simulations by model simplification [34, 67] or visibility culling [23]. This section presents two techniques for reducing the number of close pairs and contacts, ultimately achieving model simplification: "bouncing at a distance" and "freezing". Although these techniques are designed for use in OBA, any simulator can benefit from them as well. Reducing close pairs, and hence contacts, will reduce computational effort in any existing implementation of a rigid body simulator. Similarly as mentioned in the introduction to this chapter, these methods for speed-up make the simulations less physically accurate. However, at a normal playback rate, the human eye cannot perceive the difference.

6.7.1 Bounce at a Distance

In order to reduce the number of close pairs, we first introduced "bouncing at a distance". Assume we have updated positions for the current time-step t. A pair that is not contacting at t will not have impulses applied. If the same pair is overlapping at the next step $t' = t + \Delta t$, it will have its overlap resolved by an OBA position update step. Therefore, the position update QP at t' will have separating plane constraints added for this pair, and as a result the QP will be larger.

Instead of allowing this, we detect at the current time t all pairs that will have overlap at t'. For these pairs, we compute *virtual contact points* and feed them to the momentum update algorithm as if these bodies were touching. The virtual contact points are computed as follows. The distance calculation algorithm described in Section 3.1 is used to compute separating planes for these pairs [58]. By definition, the separating plane lies halfway between the planes that bound the thickest slab that separates a pair of bodies. For each body in a pair, calculate the subset of its boundary that touches the corresponding slab. Use a small numerical threshold distance to implement "touches" in the presence of numerical rounding error. This subset will be a point, line segment, or convex polygonal region. Project both convex subsets onto the separating plane of the pair. Compute the intersection of the two convex projections. The intersection is a point (one vertex), line segment (two vertices), or a convex polygonal region (multiple vertices). The vertices of this intersection are the virtual contact points.

Because of non-linearities, bouncing at a distance cannot completely eliminate overlapping bodies, but it prevents most pairs from overlapping and greatly reduces the overlap of the remaining pairs. Therefore, the position update QP in the next step is much smaller.

We gain the most speed-up for certain dynamic simulations, that is, simulations with bodies which are not settling on top of each other, but are flying in a more free fashion. If bodies settle in a stack, the performance gain is diminished because bouncing at a distance will not always avoid close pairs. Instead, we found the method discussed in the next section to be more aggressive in terms of speed-up for simulation of stacks.

6.7.2 Freezing Bodies

We observed that much computation goes into calculation of static contact response once bodies settle into a stable arrangement. Although the motion has visibly stopped for a settled stack of bodies, contact has to be resolved at every time-step just to keep the bodies from sinking into each other. Also the position update has to enforce non-overlap with separating plane constraints. It turned out to be a very powerful add-on for our simulator to allow "freezing" of bodies.

Freezing introduces a heuristic, which looks at the sum of kinetic energies $E_{lin} + E_{rot}$ for each body:

$$\frac{1}{2}m\mathbf{v}^2 + \frac{1}{2}\boldsymbol{\omega}^T\mathbf{I}\boldsymbol{\omega} < \frac{\mathbf{p}_g^2}{2m}. \tag{6.48}$$

We calculate the momentum $\mathbf{p}_g = m\mathbf{g}\Delta t$, which a body with mass m picks up during a time-step $\Delta t = 1/30sec.$ as a result from gravity[4]. Allocate a counter to each body, initially zero. If Equation 6.48 is satisfied and the body was in contact with a fixed or frozen body, increment the counter. Otherwise, reset it to zero. If a counter exceeds c_f, freeze the body. For purposes of setting up the QPs in OBA, ignore contacts and close pairs between frozen bodies or between a frozen body and a static body. A frozen body does not pick up momentum through gravity or any other external potential. The only

[4]For simplicity, our only external potential is gravity but others can be added.

way for a frozen body to become free again is by picking up momentum through a collision, which increases its sum of energies above the threshold of Equation 6.48, and hence its counter is reset to zero.

Consider a single cube sitting on a solid, static surface. Without freezing, the cube picks up momentum from gravity during each time-step, interpenetration has to be prevented with separating plane constraints, and static contact has to be resolved just to keep the cube at a visually resting state. If Equation 6.48 was satisfied while the cube was in contact with the static surface, its counter is incremented. If the counter exceeds c_f, we *freeze* the cube. This now frozen cube will no longer pick up energy through gravity, the pair frozen cube-static surface will not be considered a close pair anymore, and its contacts are not considered in the impulsive contact update. Note that the contacts are inserted into the queue, but are not processed. It will become clear soon why we find them at all. For this simple example, the number of close pairs and number of effective contacts has shrunk to zero. Therefore, the simulation will literally fly through the time-steps.

Now consider a more interesting scene with many cubes and a static container. We define "cold" volumes in which the cubes can freeze. For example, the cold volume could be the entire lower container portion. Once the cubes settle in the cold volume, we examine their energies according to Equation 6.48. Each cube maintains a counter, which we set appropriately. If a counter exceeds c_f, we freeze the cube. Again, a frozen cube no longer picks up energy from gravity, and neither a pair frozen-static, nor a pair frozen-frozen will be considered a close pair, nor will its contacts require immediate resolution.

It is important to note that *only* the hybrid momentum update will generate realistic and lively simulations. QP bouncing alone will look rather un-lively. Consider the situation of Figure 6.26 with a frozen pair of contacting bodies **A** and **B**. A third body **C** bounces

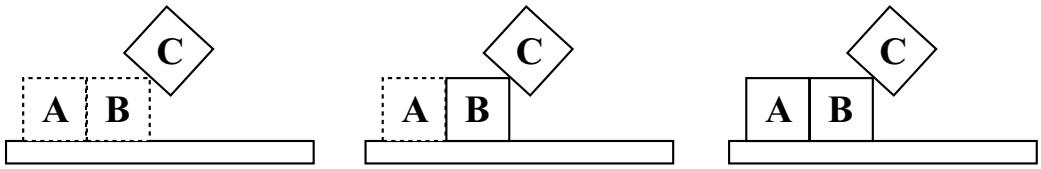


Figure 6.26: Frozen bodies (dashed) are revived one at a time: priority queue bouncing progresses left to right.

against **B**. The QP momentum update will revive only body **B** because it resolves collisions simultaneously. The contacts between **A** and **B** are inactive and will not be processed in the QP. Instead, the sequential mechanism will revive first **B**. Now, also the contacts between **A** and **B** are activated because no longer both **A** and **B** are frozen. Therefore, also these contacts are processed in the priority queue and eventually also **A** is revived. For a large cluster of frozen bodies, the number of revived bodies is dependent on the limit of bounces c_b after which we terminate the priority queue. We have therefore a way to specify liveliness and realism versus efficiency by selecting c_b. With this method of freezing bodies, we have a very powerful and efficient, yet simple way to speed up our simulations.

Chapter 7

Experiments and Results

All of the described techniques and algorithms have been implemented. The beauty of applied work as ours is that it has experimental hands-on character, which makes it sometimes more tangible than pure theory. In addition, and unlike for instance experimental physics where expensive materials and devices are employed, most experiments in computer science can be conducted with PCs and some simple software. In our case, we used medium fast Pentium PCs, standard compilers, and an expensive mathematical programming package called CPLEX. Cost of software is not a reasonable measure for the efficiency of an algorithm. Unlike hardware, software has zero marginal cost.

Our research aims at applicability and practical value. In this sense, a working implementation is absolutely necessary to emphasize the contribution of our work and to point out its advantages over existing techniques. The experiments we present here were set up in this manner to reflect the benefits of OBA. After some initial experiments, it became soon clear that free, dynamically flying bodies are more easily simulated due to the lower number of static contacts. Therefore, we followed the challenge of simulating stacks of bodies or other arrangements where bodies are very densely packed. As explained earlier,

many bodies in a crowded arrangement are a nightmare for some simulation techniques, because of the small time-step problem. 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.

7.1 Scenes

We start with a brief description of each scene. The *stack* simulation (Fig. B.1(a)) simply shows how 10 cubes become stacked on top of each other in a vertical shaft until they come to a complete rest. Although this seemingly simple scene does not look very complicated, it is actually not easily simulated. Traditional techniques have immense problems with even small stacks of cubes, since micro time-steps slow down the simulator tremendously. Similarly, *c-s stack* simulates a mix of spheres and cubes as they become stacked (Fig. B.1(b)).

Cubejam shows a large group of cubes in contact as they squeeze around obstacles. Figure B.5 shows two stills from this scene. In the first snapshot, the simulation has just started and only a few cubes are in the shaft. The second shows the final arrangement. It is visible that the simulator has to be able to model many concurrent contacts. Therefore, this more complicated example demonstrates the advantages of OBA over traditional techniques in simulation of crowded scenes. All cubes form one large contact group, and micro-steps would be unavoidable with traditional methods. The *cubejam* resembles somewhat a winding river. Although the final arrangement is less ordered, in essence also here we simulate a stack of cubes.

In *wall* (Figs. B.6 and B.7), we see how two blast waves cause a wall to collapse. The

tricky part of the simulation is actually before the wall's bricks fly apart. Tight contact is again simulated efficiently with OBA without micro-steps. Once the wall explodes, the bricks move in a very dynamic fashion and other methods could handle this part of the scene.

The *jacks* simulation (Fig. B.4) shows an example with non-convex bodies. Each jack is a collection of three convex boxes. Since our separating plane constraints are formed with respect to the convex parts, the effective number of bodies in the scene is therefore three times the number of jacks.

Pendulum (Fig. B.3) is a simple example of linked bodies, and the *hybrid* scene (Fig. B.8) 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 become wedged between the thin walls. The *hybrid* is the only scene for which we had to use the hybridized position update. All other scenes can be simulated without a problem using pure OBA.

The *sphereglass* scene (Fig. B.2(b)) has 1000 spheres without friction. Opposed to Milenkovic's position-based physics (Fig. B.2(a)), this hourglass has bouncing and parabolic trajectories. In comparison, it looks much more lively and realistic, yet we found the two simulations to run in approximately the same time per frame. Figure B.2(b) clearly shows the superior level of realism in comparison to Figure B.2(a).

Our *cubeglass* (Figs. B.11 and B.12), is a very simplified hourglass with 1000 cubes. It uses the freezing technique and pure impulse collision and static contact handling. It has friction and cubes, which are more complicated to simulate than spheres due to the arising

	Stack	C-S stack	Cubejam	Wall	Jacks	Pendulum
#bodies	10	12	100	90	50(150)	6
close pairs/fr.	6.6	11.8	172	129.2	123.3	0
collisions/fr.	29	13.4	411.7	404.3	167.1	0
contacts/fr.	25.7	13.4	278.4	220.6	94	0
#frames	600	600	1500	600	1500	1000
sec./fr.	0.9	0.6	22.2	12.3	15.5	0.1
avg. #qp/fr.	3	4.6	4.8	4	4.6	2.1
rollbacks [%]	0	0.3	0	0	0.03	0

Table 7.1: Complexity of scenes and efficiency issues I.

	Hybrid	Robot	Sphereglass	Cubeglass	Office toy
#bodies	100	306	1000	1000	10
close pairs/fr.	159.1	308	1723.4	302.6	8
collisions/fr.	518.7	507.7	1673.6	471.5	7.9
contacts/fr.	497.6	507.7	1673.6	471.5	7.9
#frames	1500	580	2000	1500	300
sec./fr.	25.9	75.5	13.5	87.2	2.1
avg. #qp/fr.	4.8	6	2.6	1.9	2.3
rollbacks [%]	0.07	2.7	0	0	0.3

Table 7.2: Complexity of scenes and efficiency issues II.

contact geometry. At the current state of the implementation, this particular simulation would not have been possible without the freezing technique due to running time issues.

In our *robot* 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 (Fig. B.10). Combining *a priori* animation with dynamic simulation is of interest for applications in film. It turns out to be a non-trivial endeavor.

The last scene, the *office toy* (Fig. B.9), is included to demonstrate that our hybrid momentum update can realistically simulate sequential bouncing when this is necessary.

7.1.1 Discussion

Tables 7.1 and 7.2 give information about the complexity of the scenes and provide insight into running time and efficiency related data. All scenes except the *cubeglass* were simulated without the freezing technique. It is intuitively clear that the complexity rises with the number of bodies in a scene. Collision and contact handling depends clearly on the number of contacts in a scene. The position update inserts separating plane constraints for close pairs and will therefore depend on the number of such close pairs. Although our scenes are non-trivial, we achieve good performance. The time spent to calculate one frame stays reasonable even for scenes with many bodies, collisions, and static contacts.

The *pendulum* 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. Therefore, traditional techniques would have been a better choice for simulating this scene.

As expected, the *hybrid* method has a higher running time than the otherwise similarly complex *cubejam*, since it makes partial use of RD. The *sphereglass* scene illustrates that OBA runs fast even for 1000 spheres. It is frictionless but, unlike position-based physics [57], it has bouncing and parabolic trajectories. In comparison, the *cubeglass* shows that OBA can simulate even 1000 cubes with friction if the freezing technique is implemented. Our initial experiments without freezing limited the number of bodies that could be simulated in acceptable time to a few hundred. The motion of cubes is more complicated than rotationally invariant spheres, and the contact geometry between cubes gives rise to a larger number of contacts, and critical vertices, *i.e.* constraints. Since the

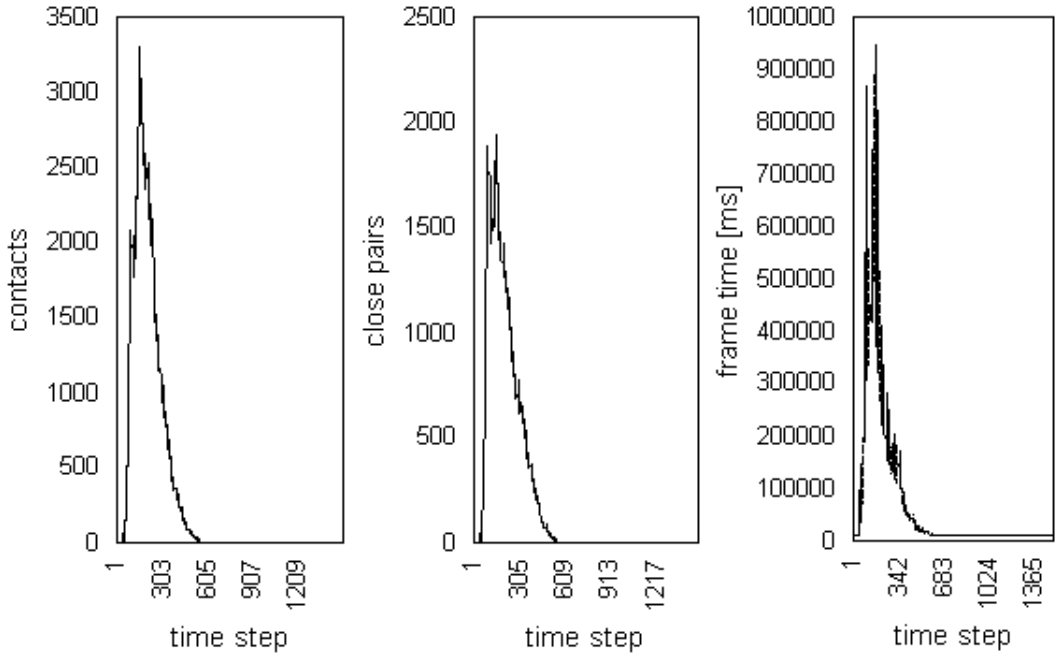


Figure 7.1: Development of number of contacts, number of pairs, and frame time with freezing.

cubeglass has friction, the cubes tend to become wedged into the funnel opening, thus clogging the downward flow of cubes. Contacts persist longer than in the *sphereglass*. In comparison to the *sphereglass* it becomes clear that freezing eliminates close pairs and contacts, thus achieving the performance that makes simulation of 1000 cubes possible.

Table 7.2 has numbers for close pairs and contacts for the two hourglass simulations. We see that the *cubeglass* takes about 5x as long to simulate per frame than the *sphereglass*. This is surprising at first glance since both the numbers of close pairs and the numbers of contacts are less for the *cubeglass*. However, Figure 7.1 shows the development of numbers of contacts, close pairs, and simulation time per frame when freezing is used. The number of contacts per frame peaks at over 3000, the number of close pairs at 2000 per frame. As the cubes settle and lose most kinetic energy, they become frozen. Pairs of frozen cubes are not counted as close pairs, and they do not have contacts declared. Therefore, we see a sharp decrease of number of contacts, number of close pairs, and simulation time

per frame as the simulation develops over time and more cubes are frozen. Figures B.11 and B.12 show stills from this simulation. The large number of contacts is easily conceived by looking at these images. In this scene, the cold volume is the part of the container where the cubes are collected once they fall through the funnel. Comparing the three plots reveals a strong correlation between the development of all three observed quantities. Finally, all nine scenes are visually extremely stable. Stability is an important feature of any simulator.

The experiments show that our iterated position QP algorithm 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. We also observe a higher number of QPs solved. The algorithm needs to do more optimization work to accommodate non-overlap constraints when *a priori* animated bodies are present.

Table 7.3 makes a little comparison between scenes with and without the freezing technique when it is possible. We examine the simulation time per frame for the *stack*, the *c-s stack*, and the *cubejam*. We find a tremendous speed-up by a factor 30. For the *c-s stack*, the speed-up is only by a factor 15. Simulation of spheres is at a lesser cost to start with, because involved computations are cheaper. Therefore, the gap between the two timings is smaller. The final simulations are as plausible as without freezing.

Assuming this factor, the *cubeglass* would have taken 45 days to simulate vs. 1.5 days with the freezing method. Although 1.5 days is still a long time, it is acceptable considering that a run over a weekend provides the whole simulation. In contrast, 45 days

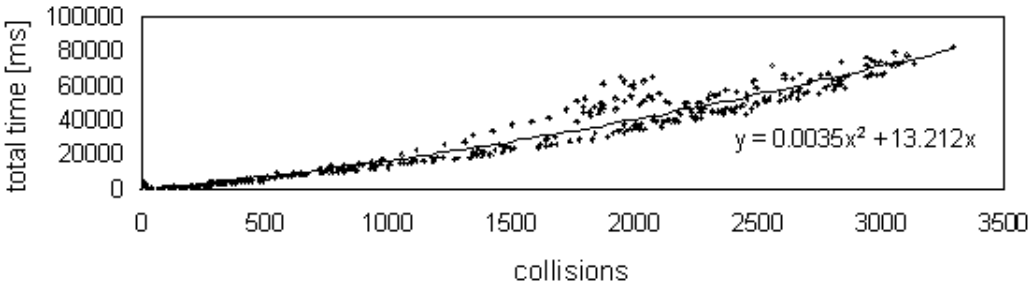


Figure 7.2: Number of collisions vs. time per QP.

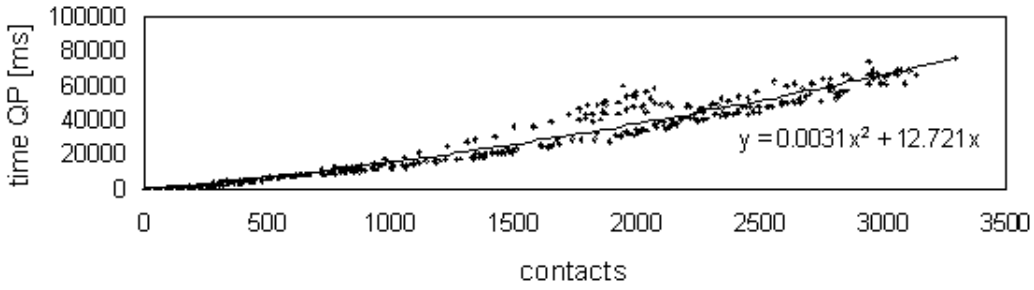


Figure 7.3: Number of static contacts vs. time per QP.

is completely unacceptable at any rate.

Figure 7.2 shows the dependence of the solution time per frame in the collision resolution phase. We can see that there is a polynomial dependence. The same goes for the static contact impulse phase (Fig. 7.3). The two plots are similar, because we solve similar QPs. Figure 7.2 shows a slightly steeper trend since we also solve some bounces in the priority queue.

For the *stack*, we set $c_f = 1$ (Sec. 6.7.2). In contrast, we have to use a higher threshold for the *cubejam*. As cubes squeeze around the obstacles, they come to a rest only seemingly due to friction although they are still far from the bottom. We had to set $c_f = 3$ and declare a cube then as frozen.

	Stack	C-S stack	Cubejam	Cubeglass
sec./fr. no freezing	0.9	0.64	22.2	—
sec./fr. with freezing	0.03	0.05	0.6	85

Table 7.3: Comparison of OBA with and without freezing.

7.1.2 Analysis

Since each of the algorithm's three stages is implemented by solving QPs, the overall efficiency will be determined by how fast we can solve an individual QP. 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 body can have up to a constant number of neighbors depending on the body geometry. Solid state physics, more concisely the physics of crystals, provides theory and different models for such tightest packing problems. It turns out 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 demand about 20-25% of the total computations. The remainder goes into other tasks like maintaining data structures and general bookkeeping.

It is therefore clear that the overall simulation time of a scene will depend on the particular shape of its bodies and how crowded the bodies are. Hence, it is hard to compare the *stack* simulation with the *jacks*, since the latter are non-convex. Our *wall* can be simulated much faster than *cubejam*, although the number of bodies 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, which take time to process.

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 contacts. The two simulations, *jacks* and *wall*, still run in comparable time since there are multiple convex components in each jack, and thus the effective number of bodies is larger.

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. As mentioned in Section 7.1, the *a priori* animated claw and robot move on fixed paths. They bounce off dynamic bodies. This can in some instances be not possible due to the arrangement of bodies. Imagine a robot hand which pushes a cube perfectly perpendicular against a wall. There is no place for the cube to go to prevent interpenetration if the hand keeps moving closer to the wall. This will result in a non-physical situation and either causes the cube to jump abruptly to either side if it can or will result in an infeasibility. The claw grabs the robot and drags him out of the pile. It is the animator's task to generate a claw and robot which move in a way that will not lead into infeasibilities. This hides a potential danger through human error. We currently alleviate this problem by slanting the container walls. When the claw grabs the robot and drags him out of the pile, the risk of having the robot pressed perpendicularly against the walls is smaller. However, it is not zero.

To investigate the dependency of OBA's time complexity on number of close pairs and contacts, we ran a separate simulation that analyzes the efficiency of OBA under uniform conditions. We dropped 200 cubes in packages of nine at every ten frames into a

vertical shaft from a small height. 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 were recorded. The cubes settle into a stable, tightly packed arrangement. 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. For a given body shape, a close pair will have a corresponding contact geometry. The contact geometry dictates the number of critical vertices and therefore the number of constraints. For these reasons, also the number of constraints in a QP is proportional to the number of bodies.

If the objective is convex, as it is in our case, the theoretical time to solve a QP is a polynomial of high degree. However, CPLEX software uses a variety of techniques to obtain good running times in practice. We verified that the time to solve an individual QP using CPLEX is roughly $O(m^2)$, 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 $O(\log n)$. 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 $O(n^2)$. Figure 7.4 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

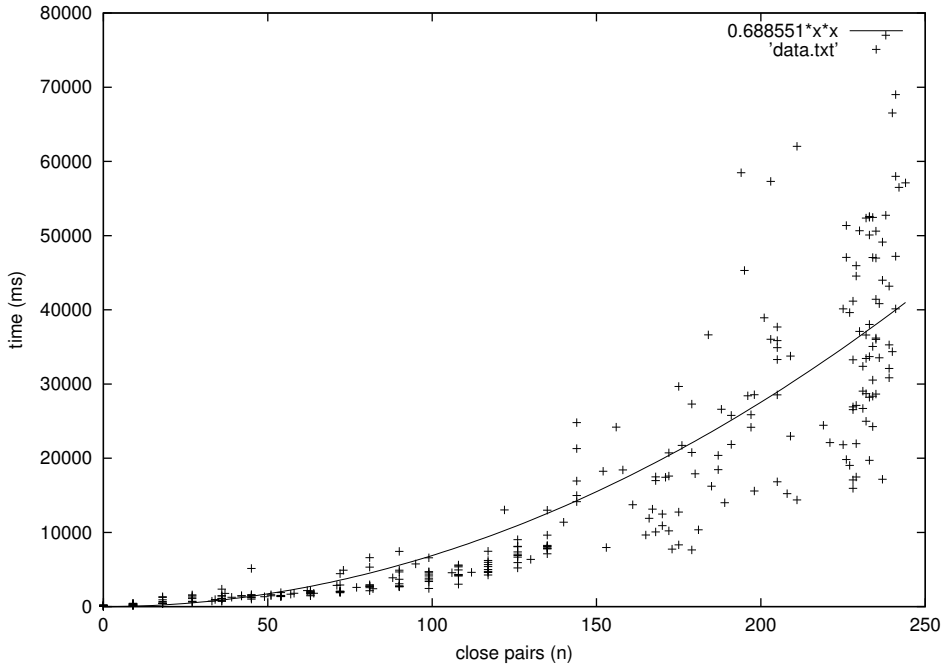


Figure 7.4: Solution time of one QP vs. number of close pairs for the position update.

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 QPs which are outlying regarding their solution time. The latter depends obviously on the QP solver that is used. QP solver software uses internal tolerances, which affect the solution quality. It is important to be certain about the reliability of such black box software modules whenever they are used.

There is a possibility to improve the running time by decomposing the position update. It has been investigated for two-dimensional polygonal compaction problems [46], and we intend to generalize this work to three dimensions. Rather than solving one large problem, we break it down appropriately and solve it in a parallel or otherwise distributed fashion. 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. Let n be the number of constraints, and

therefore $O(n^2)$ is the time complexity for solving the problem. If we break down the problem into k parts with n/k constraints each, then the complexity for each is $O((n/k)^2)$. Solving all k problems sequentially yields overall complexity $O(n^2/k)$. It turns out to be a somewhat optimistic prediction, because breaking down a problem this way usually makes more iterations necessary. This is due to the fact that after breaking up a problem, we have a loss of global communication, *i.e.* satisfaction of constraints over separated partial problems needs more iterations. Naturally, distributing the sub-problems in a parallel fashion further improves running time.

7.2 Implementation

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 *sphereglass* were coded in Java on a 450MHz Pentium III processor running Windows NT. The *sphereglass* 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 [54]. 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 the *sphereglass* 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 value for gravity $|\mathbf{g}| = 10$ m/sec^2 throughout our experiments. The

coefficient of restitution for the *sphereglass* was $\epsilon = 0.7$ and for all other simulations was $\epsilon = 0.3$. Except for the frictionless *sphereglass*, the Coulomb friction coefficient was $\mu = 0.3$. Our bodies were given material density 0.8 g/cm^2. These values correspond to some "normal" types of wood. We also experimented with some rubber- and steel-like bodies. Rubber was given higher density, restitution and friction than wood, steel had even higher density, but lower values for both ϵ and μ. Although the simulations change greatly visually, changing these body parameters has little effect on the performance of the algorithm. **Note**: very high ϵ will make bodies bouncier, and hence avoids them from becoming stacked easily. Therefore, the number of close pairs is less, and thus the QPs in the position update are smaller. Overall, and in average, the performance is the same regardless of material if we run the simulation long enough.

Chapter 8

Future Work

We briefly describe now possible future directions of our work and make suggestions for improvement. The current OBA algorithm focuses on rigid bodies. We have shown how OBA can simulate even 1000 cubes in an hourglass-like scene. Application to other domains is a possible, interesting extension to our work, which we plan to investigate.

Flexible bodies have been described in various papers [10, 82, 83, 84]. This moves us away from the simpler rigid body mechanics into particle systems. The usual approach to flexibility is to build a mesh of n particles, which are connected by springs. The transition to rigid bodies is via stiffening these springs. For perfectly stiff springs, we arrive at point masses connected by stiff rods. This is a possible representation for rigid bodies. For elastic springs, each of the n particles has six degrees of freedom and obeys the laws of mechanics. That is, all equations from rigid bodies as in Section 2.1 still apply. Only now we apply these equations to idealized point masses. For n particles with six degrees of freedom each, we have to solve a system of $6n$ differential equations. Depending on the application, it may make sense to reduce the degrees of freedom for each particle to less than six, *e.g.* do not allow them to twist the connecting springs, but only allow elongation and retraction or

vice versa. Specifying the degrees of freedom, the particle properties (mass etc.), and the spring specifications allows building of a flexible body with the desired properties. Particle systems are versatile. They can be applied for simulation of various phenomena including gases [31] and fluids [44]. Just as from flexible to rigid bodies, there is a transition from gases to liquids. This is as intuitively expected from observations in the real world, where we observe phase transitions from gaseous, to liquid, to solid.

Realism in animation necessitates flexible models for simulation of expression and sculpting [28, 85]. Modeling facial expressions, but also cloth draped over a character are very important. Much research has been done lately in the field of cloth simulation [11, 18, 29, 73]. Cloth falls into the sector of deformable bodies [87], but is special in that it is usually modeled as a two-dimensional layer of particle meshes. Cloth can bend easily, but has limitations regarding stretching. It has been reported that it is difficult to simulate cloth-cloth interactions [42].

Closely related to cloth and general flexible bodies, we find applications in computational biology and chemistry, namely simulation of molecules. We feel that minimization of energy potentials is a natural and physically correct approach for such problems. We can easily add other potentials into the OBA algorithm to model far and near forces between atoms. Modeling molecules has been explored, and parallel algorithms have been employed [25, 65, 66].

The ability to simulate hair stimulates great interest in the animation community. Many animation features of the past had insects as main characters because the simulation of hair is so difficult. Progress was made lately by moving away from physical simulation of hair to simplified, approximate simulation. It would be an enormous challenge and an interesting project to extend OBA in a way that allows simulation of hair.

The simple techniques for speed-up discussed in Section 6.7 proved to be effective for a system that settles and forms static contacts. It seems a promising direction to find methods for further model simplification not only for settled bodies. Simulation level of detail (SLOD) techniques reduce accuracy, and hence computations, for regions that are currently not the focus of interest. SLOD computation fits therefore well into the concept of plausible animation techniques. For particle systems [67], O'Brien *et al.* have shown clusters of particles can be advanced simultaneously following similar paths. Opposed to three-dimensional rigid bodies this is possible due to the simplified motion of particles, namely the absence of rotation. We could use a similar approach of automatic decomposition based on certain criteria, such as distance from the viewer, bodies that form "contact groups", or bodies with similar physical properties otherwise. For instance, it would speed up computations if we treated contacts in hidden bodies in a simplified manner.

We also feel that further development of OBA should point towards a parallel implementation. Instead of solving one large problem, spatial decomposition is used to break it down into smaller parts, which can be solved in a distributed fashion. This seems to be a promising future direction for general simulation and animation. It has been used in computer graphics for rendering since long. Arrays of high-speed machines are used for rendering in high quality movie productions to achieve acceptable performance and meet tight deadlines.

We have mentioned in Chapter 7 that we experienced some difficulties in interaction of dynamic OBA simulation with *a priori* animation. There is room for improvement from which especially the movie industry could benefit greatly. Animated characters might have to be added into the optimization in a way that treats their targets with a high priority. That is, whenever there is a conflict between an *a priori* body and a dynamic body, priority

is given to the *a priori* body to meet its target. If the latter is not possible and infeasibility would be the result, also the *a priori* body experiences a perturbation to avoid non-physical situations.

Our elaborations targeted much the drawbacks of traditional techniques with the small time-step problem, yet we deliberately re-introduce "some" small time-steps in our hybrid position update approach. Although the method works acceptably fast and addresses the problem of bodies passing through each other, we think it is a direction with possible benefits to explore alternatives to the hybrid method.

As mentioned in Section 6.2.2, traditional bouncing that does not minimize the kinetic energies can increase the total energy after a collision. Solving also individual collisions in the PQ algorithm with our QP algorithm would be too expensive due to computational overhead in the QP solver. Instead, we feel it is advisable to develop efficient special code that solves QPs for one collision only.

Chapter 9

Applications and Conclusions

This completes our description of the optimization-based animation algorithm. We will now round off our treatise by discussing possible applications and presenting conclusions. The current implementation is clean and stable, yet it is in an experimental stage. For commercial use, a different, more user-friendly GUI would be necessary. The character of our work is highly applied and practical. It can be therefore used in application programs that are concerned with the simulation of many-body systems.

9.1 Applications

The OBA algorithm can be used in any existing simulation package that follows a modular structure as our simulator. The heart of OBA is without a doubt the position update. It can be included easily into any existing simulator, reusing available contact response modules. This makes it a versatile and simple plug-in for applications. Also our impulsive contact suite [76] can be turned into a plug-in. Many animation software packages work based on plug-ins, *i.e.* extension packages for special purposes. OBA or any of its components

can be implemented as a plug-in to any of these commercial products as a special tool for animation of crowded rigid body systems. We do not foresee real time performance at this point though, as it would be necessary for computer games. Processing overhead in the QP solver currently limits our simulations to offline.

9.2 Conclusions

We presented a new algorithm for animation of large systems of convex bodies. Due to computational overhead in solving small QPs, there is a performance disadvantage in using OBA for smaller, simple systems with few or no collisions. OBA can be perfectly employed for scenes where *plausible* animation is adequate. It is efficient for large systems, very stable, and realistic where it matters within visual limits. Stability is an important trait of any simulation algorithm. 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 algorithm is superior in this particular application.

OBA is a plausible simulation algorithm. However, also analytic methods that avoid simulation problems by use of heuristics (Sec. 6.2.1) become plausible. It seems that use of heuristics is common, yet usually not even published. If people use plausibility methods anyways to deal with problems in simulation, then a solid, algorithmic approach as in OBA has a much stronger basis than heuristics. The latter give rise to instabilities in simulations.

Contacts and collisions are resolved with impulses only. This follows Mirtich's *impulse* simulations and addresses the findings of other researchers who noted that impulsive

forces are necessary whenever true contact forces cannot be calculated. Despite the aspect of plausibility in our simulations, they exhibit no sign of visually lacking realism. With hybridization of priority queue and pure QP momentum update, we have a method for trading off simulation liveliness for simulation speed when simulating large numbers of collisions. The freezing technique allows tremendous speed-up of rigid body simulation. Collisions and static contacts are each solved with a single QP. The whole contact suite is implemented by solving two successive QPs.

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. The use of mathematical programming facilitates implementation greatly.

Appendix A

Basic Geometry, Physics, and QPs

A.1 Orientation Representations and Conversions

Quaternions implement rotation by specifying an axis of rotation $\boldsymbol{\gamma}$ and an angle φ around the axis. We write the corresponding quaternion with a real part r and a vector part $\mathbf{v}$:

$$\mathbf{Q} = [r, \mathbf{v}] = [\cos(\varphi/2), \sin(\varphi/2)\boldsymbol{\gamma}]. \tag{A.1}$$

We convert a quaternion $\mathbf{Q} = [r, \mathbf{v}]$ to a rotation matrix $\mathbf{R}$:

$$\mathbf{R} = \begin{pmatrix} 1 - 2\mathbf{v}_y^2 - 2\mathbf{v}_z^2 & 2\mathbf{v}_x\mathbf{v}_y - 2r\mathbf{v}_z & 2\mathbf{v}_x\mathbf{v}_z - 2r\mathbf{v}_y \\ 2\mathbf{v}_x\mathbf{v}_y + 2r\mathbf{v}_z & 1 - 2\mathbf{v}_x^2 - 2\mathbf{v}_z^2 & 2\mathbf{v}_y\mathbf{v}_z - 2r\mathbf{v}_x \\ 2\mathbf{v}_x\mathbf{v}_z - 2r\mathbf{v}_y & 2\mathbf{v}_y\mathbf{v}_z + 2r\mathbf{v}_x & 1 - 2\mathbf{v}_x^2 - 2\mathbf{v}_y^2 \end{pmatrix}. \tag{A.2}$$

The rotation vector $\boldsymbol{\phi}$ also implements an axis of rotation and has magnitude equal to $\tan\varphi$, where φ is the angle of rotation in radians counterclockwise about the rotation axis. We need to know how to transform $\mathbf{Q} = [r, \mathbf{v}]$ into the corresponding $\boldsymbol{\phi}$ by applying

trigonometry:

$$\begin{aligned}
\cos(\varphi/2) &= r, \\
\sin(\varphi/2) &= \sqrt{1-(\cos(\varphi/2))^2}, \\
\sin(\varphi) &= 2\sin(\varphi/2)\cos(\varphi/2), \\
\cos(\varphi) &= \cos(\varphi/2)^2 - \sin(\varphi/2)^2, \\
\tan(\varphi) &= \sin(\varphi)/\cos(\varphi), \\
\boldsymbol{\phi} &= \frac{\mathbf{v}}{\sin(\varphi/2)}\tan(\varphi).
\end{aligned} \tag{A.3}$$

A.2 Distance Calculation with Spheres

A.2.1 Sphere-Sphere

This is the easiest case. For spheres **A** and **B** with radii r_a and r_b, and positions $\mathbf{x}_a$ and $\mathbf{x}_b$ respectively the normal direction is simply:

$$\mathbf{n} = |\mathbf{x}_b - \mathbf{x}_a|, \tag{A.4}$$

and the distance D:

$$D = \mathbf{n} \cdot (\mathbf{x}_b - \mathbf{x}_a) - r_a - r_b. \tag{A.5}$$

A.2.2 Sphere-Polyhedron

To generate $\mathbf{n}$, we test the cases *v-v*, *v-e*, and *v-f* of the sphere's center $\mathbf{x}$ against the vertices, edges, and faces of the polyhedron. For a sphere with radius r, we calculate the distance D:

$$D = \left(\mathbf{x}_b - \max_{\mathbf{q}_a \in A} \mathbf{n} \cdot \mathbf{q}_a\right) - r_b, \tag{A.6}$$

if the polyhedral body is **A**, and

$$D = \left(\min_{\mathbf{q}_b \in B} \mathbf{n} \cdot \mathbf{q}_b - \mathbf{x}_a \right) - r_a \tag{A.7}$$

if the polyhedral body is **B**.

A.3 Summary of QPs

We follow the conventions from earlier chapters to summarize the QPs that are solved in our OBA algorithm. In particular, for a pair of *convex* bodies **A** and **B**, the normal **n** points from **A** to **B**. All lowercase, boldfaced variables are vectors and need to be represented with three component variables each.

A.3.1 Position Update QP

Each QP that is solved in the iterated position update algorithm from Section 6.1 is of the following structure:

$$\begin{aligned}
&\textit{Minimize} && \textstyle\sum_{i=1}^{n} m_i \Delta\Delta\mathbf{x}_i \cdot \Delta\Delta\mathbf{x}_i + \Delta\Delta\boldsymbol{\phi}_i^T \mathbf{I}_i \Delta\Delta\boldsymbol{\phi}_i \\
& && (\textit{or } \textstyle\sum_{i=1}^{n} \Delta\Delta\mathbf{x}_i \cdot \Delta\Delta\mathbf{x}_i + D_i^2 \Delta\Delta\boldsymbol{\phi}_i \cdot \Delta\Delta\boldsymbol{\phi}_i) \quad (1) \\
&\textit{subject to} && \\
& && \Delta\Delta\mathbf{x}_i = \Delta\mathbf{x}_i - \mathbf{x}_i^{tgt} + \mathbf{x}_i, \quad (2) \\
& && \Delta\Delta\boldsymbol{\phi}_i = \Delta\boldsymbol{\phi}_i - \Delta\boldsymbol{\phi}_i^{tgt}, \quad (3) \\
&\mathbf{q}_i \in \mathbf{A}: && \mathbf{n} \cdot \Delta\mathbf{x}_i - (\mathbf{n} \times (\mathbf{q}_i - \mathbf{x}_i)) \cdot \Delta\boldsymbol{\phi}_i + \\
& && (\mathbf{n}_x \cdot \mathbf{q}_i)\delta_x + (\mathbf{n}_y \cdot \mathbf{q}_i)\delta_y - d \leq -\mathbf{n} \cdot \mathbf{q}_i, \quad (4) \\
&\mathbf{q}_i \in \mathbf{B}: && \mathbf{n} \cdot \Delta\mathbf{x}_i - (\mathbf{n} \times (\mathbf{q}_i - \mathbf{x}_i)) \cdot \Delta\boldsymbol{\phi}_i + \\
& && (\mathbf{n}_x \cdot \mathbf{q}_i)\delta_x + (\mathbf{n}_y \cdot \mathbf{q}_i)\delta_y - d \geq -\mathbf{n} \cdot \mathbf{q}_i, \quad (5) \\
& && -0.1 \leq \Delta\boldsymbol{\phi}_i \leq +0.1. \quad (6)
\end{aligned}$$

The objective function (1) is a choice of two possibilities that implement our distance to target measure (Eqs. 6.7 and 6.8). The first is perhaps more physically motivated because it will have a heavier body push aside a lighter body. The equality constraint (2) relates current and target position with perturbation and target deficit for a body in the linear case. For the angular case (3), only the target orientation with respect to the current orientation, the angular perturbation, and the angular target deficit appear. For the previous two constraints, refer to Equations 6.3 through 6.5 in the text. Assume a close pair of bodies $\langle \mathbf{A}, \mathbf{B} \rangle$ with critical vertices on each body. The non-overlap constraint (4) is for a vertex on body $\mathbf{A}$, and (5) is for a vertex on body $\mathbf{B}$ (Eqs. 6.18 and 6.19). The inequality (6) bounds the rotational perturbation on a body per iteration of the position update in order to assure convergence of the algorithm in practice (Section 6.1.6).

Indices:

i body index, $i = 1 \ldots n$.

Variables:

$\Delta\Delta\mathbf{x}_i$ linear target deficit,

$\Delta\mathbf{x}_i$ linear body perturbation,

$\Delta\Delta\boldsymbol{\phi}_i$ angular target deficit,

$\Delta\boldsymbol{\phi}_i$ angular body perturbation,

d separating plane's distance to the origin for a pair $\langle \mathbf{A}, \mathbf{B} \rangle$,

δ_x separating plane's rotation around $\mathbf{n}_x$ for a pair $\langle \mathbf{A}, \mathbf{B} \rangle$,

δ_y separating plane's rotation around $\mathbf{n}_y$ for a pair $\langle \mathbf{A}, \mathbf{B} \rangle$.

Constants:

m_i body mass,

$\mathbf{I}_i$ body inertia,

D_i body diameter,

$\mathbf{x}_i^{tgt}$ linear target position,

$\mathbf{x}_i$ linear current position,

$\Delta\boldsymbol{\phi}_i^{tgt}$ angular target orientation with respect to the current orientation,

$\mathbf{n}_x, \mathbf{n}_y, \mathbf{n}$ collision coordinate frame with normal $\mathbf{n}$ for a pair $\langle \mathbf{A}, \mathbf{B} \rangle$,

$\mathbf{q}_i$ a vertex on body i.

A.3.2 Momentum Update QP

We summarize our QP-model from Section 6.2 for the momentum update. The same QP implements also the impulsive static contact update with $\epsilon_s = 0$:

$$\textit{Minimize} \qquad \sum_{i=1}^{n} \tfrac{1}{2}\, m_i (\mathbf{v}_i^{new})^2 + \tfrac{1}{2}\, (\boldsymbol{\omega}_i^{new})^T \mathbf{I}_i \boldsymbol{\omega}_i^{new} \tag{1}$$

subject to

$$i \equiv \mathbf{A}: \qquad \mathbf{v}_i^{new} = \mathbf{v}_i - m_i^{-1} \sum_{s=1}^{t} z_{is} \mathbf{j}_s \quad \text{and}$$

$$\boldsymbol{\omega}_i^{new} = \boldsymbol{\omega}_i - \mathbf{I}_i^{-1} \sum_{s=1}^{t} z_{is} (\mathbf{r}_{is} \times \mathbf{j}_s), \tag{2}$$

$$i \equiv \mathbf{B}: \qquad \mathbf{v}_i^{new} = \mathbf{v}_i + m_i^{-1} \sum_{s=1}^{t} z_{is} \mathbf{j}_s \quad \text{and}$$

$$\boldsymbol{\omega}_i^{new} = \boldsymbol{\omega}_i + \mathbf{I}_i^{-1} \sum_{s=1}^{t} z_{is} (\mathbf{r}_{is} \times \mathbf{j}_s), \tag{3}$$

$$k \equiv \mathbf{A},\ l \equiv \mathbf{B}: \quad v_s^+ = \mathbf{n} \cdot ((\mathbf{v}_l^{new} + \boldsymbol{\omega}_l^{new} \times \mathbf{r}_{ls}) - (\mathbf{v}_k^{new} + \boldsymbol{\omega}_k^{new} \times \mathbf{r}_{ks})), \tag{4}$$

$$\text{if } \ v_s^- \geq 0 \ \text{ then } \ v_s^+ \geq 0 \ \text{ else } \ v_s^+ \geq -\epsilon_s \cdot v_s^-, \tag{5}$$

$$\mathbf{u}_{hs} \cdot \mathbf{j}_s \geq 0. \tag{6}$$

The objective function (1) is the sum of kinetic body energies after impulses were applied (Eq. 6.28). Assume a contacting pair of bodies $\langle \mathbf{A}, \mathbf{B} \rangle$ and contacts indexed by s. The equalities (2) calculate the body velocities after application of all impulses for body $\mathbf{A}$ of a pair (Eq. 6.26), and (3) does the same for body $\mathbf{B}$ (Eq. 6.25). Equality (4) calculates the relative normal velocity at contact s after impact assuming contacting bodies $\mathbf{A}$ and

B with indices k and l respectively (Eq. 6.24). The inequalities in (5) implement Newton's collision law for contact s depending on whether the relative contact normal velocity before application of impulses was interpenetrating or not (Eq. 6.27). We have finally eight inequalities to implement the linearized friction cone constraint (6) for each contact s (Eq. 6.23).

Indices:

i body index, $i = 1 \dots n$,

k, l body indices for contacting pair $\langle \mathbf{A}, \mathbf{B} \rangle$ respectively,

s contact index, $s = 1 \dots t$,

h index for friction cone inward pointing face normals, $h = 0 \dots 7$.

Variables:

$\mathbf{j}_s$ the impulse at contact s,

$\mathbf{v}_i^{new}$ linear velocity of body i after application of impulses,

$\boldsymbol{\omega}_i^{new}$ angular velocity of body i after application of impulses,

v_s^+ relative contact normal velocity at contact s after impulse.

Constants:

z_{is} 1 if contact s lies on body i, 0 otherwise,

m_i body mass,

$\mathbf{I}_i$ body inertia,

$\mathbf{v}_i$ linear body velocity before impulse,

$\boldsymbol{\omega}_i$ angular body velocity before impulse,

v_s^- relative contact normal velocity at contact s before impulse,

$\mathbf{r}_{is}$ contact lever body i at contact s,

ϵ_s coefficient of restitution at contact s,

$\mathbf{u}_{hs}$ inward pointing friction cone face normals for contact s,

$\mathbf{n}$ normal for a pair $\langle \mathbf{A}, \mathbf{B} \rangle$.

A.3.3 Force Update QP

We summarize our QP-model for the acceleration or force update (Sec. 6.3.1). First, for the simpler frictionless case, then for the friction case.

Frictionless Case

$$\begin{aligned}
&\textit{Minimize} && \textstyle\sum_{i=1}^{n} -m_i \mathbf{g} \cdot \mathbf{a}_i + \frac{1}{2} m_i \mathbf{a}_i \cdot \mathbf{a}_i + \frac{1}{2} \boldsymbol{\alpha}_i^T \mathbf{I}_i \boldsymbol{\alpha}_i && (1)\\
&\textit{subject to} && && \\
&k \equiv \mathbf{A},\ l \equiv \mathbf{B}: && \mathbf{n} \cdot (\mathbf{a}_l - \mathbf{a}_k + \boldsymbol{\alpha}_l \times \mathbf{r}_{ls} - \boldsymbol{\alpha}_k \times \mathbf{r}_{ks}) = a_s^{\perp}, && (2)\\
& && a_s^{\perp} \geq 0. && (3)
\end{aligned}$$

We derive the objective function (1) by plugging the body accelerations into the sum of kinetic body energies and the potential energy (Eq. 6.38). Equality (2) calculates the relative normal acceleration at contact s assuming a pair of contacting bodies $\langle \mathbf{A}, \mathbf{B} \rangle$ with indices k and l respectively. The inequality (3) implements non-interpenetration by constraining the relative normal acceleration at contact s to be positive. For the previous two constraints, refer to Equation 6.37 in the text.

Indices:

i body index, $i = 1 \ldots n$,

k, l body indices for contacting pair $\langle \mathbf{A}, \mathbf{B} \rangle$ respectively,

s contact index, $s = 1 \ldots t$.

Variables:

$\mathbf{a}_i$ linear acceleration body i,

$\boldsymbol{\alpha}_i$ angular acceleration body i,

$a_s^{\perp}$ relative contact normal acceleration at contact s.

Constants:

m_i body mass,

$\mathbf{I}_i$ body inertia,

$\mathbf{g}$ acceleration of gravity,

$\mathbf{r}_{is}$ contact lever body i at contact s,

$\mathbf{n}$ normal for a pair $\langle \mathbf{A}, \mathbf{B} \rangle$.

Friction Case

Take the frictionless QP from Equation A.10, calculate the full three-dimensional contact acceleration vector, and add the acceleration cone constraints:

$$\begin{aligned}
&\textit{Minimize} && \textstyle\sum_{i=1}^{n} -m_i \mathbf{g} \cdot \mathbf{a}_i + \frac{1}{2} m_i \mathbf{a}_i \cdot \mathbf{a}_i + \frac{1}{2} \boldsymbol{\alpha}_i^T \mathbf{I}_i \boldsymbol{\alpha}_i && (1)\\
&\textit{subject to} && && \\
&k \equiv \mathbf{A},\ l \equiv \mathbf{B}: && \mathbf{n}_x \cdot (\mathbf{a}_l - \mathbf{a}_k + \boldsymbol{\alpha}_l \times \mathbf{r}_{ls} - \boldsymbol{\alpha}_k \times \mathbf{r}_{ks}) = a_s^x, && (2)\\
& && \mathbf{n}_y \cdot (\mathbf{a}_l - \mathbf{a}_k + \boldsymbol{\alpha}_l \times \mathbf{r}_{ls} - \boldsymbol{\alpha}_k \times \mathbf{r}_{ks}) = a_s^y, && (3)\\
& && \mathbf{n} \cdot (\mathbf{a}_l - \mathbf{a}_k + \boldsymbol{\alpha}_l \times \mathbf{r}_{ls} - \boldsymbol{\alpha}_k \times \mathbf{r}_{ks}) = a_s^{\perp}, && (4)\\
& && \mathbf{u}_{hs} \cdot (\mathbf{a}_s - \hat{\mathbf{a}}_s) \geq 0. && (5)
\end{aligned}$$

We use the same objective function (1) as in the frictionless case by plugging the body accelerations into the sum of kinetic body energies and the potential energy (Eq. 6.38). The equalities (2)-(4) calculate the relative acceleration components at contact s in x-, y-, and normal-direction assuming a pair of contacting bodies $\langle \mathbf{A}, \mathbf{B} \rangle$ with indices k and

l respectively (Eq. 6.40). We have finally eight inequalities to implement the linearized acceleration cone constraint (5) for each contact s (Eq. 6.41).

Indices:

i body index, $i = 1 \dots n$,

k, l body indices for contacting pair $\langle \mathbf{A}, \mathbf{B} \rangle$ respectively,

s contact index, $s = 1 \dots t$,

h index for acceleration cone inward pointing face normals, $h = 0 \dots 7$.

Variables:

$\mathbf{a}_i$ linear acceleration body i,

$\boldsymbol{\alpha}_i$ angular acceleration body i,

a_s^x relative contact acceleration at contact s in x-direction,

a_s^y relative contact acceleration at contact s in y-direction,

$a_s^\perp$ relative contact normal acceleration at contact s,

$\mathbf{a}_s$ the contact acceleration vector $(a_s^x, a_s^y, a_s^\perp)$ at contact s.

Constants:

m_i body mass,

$\mathbf{I}_i$ body inertia,

$\mathbf{g}$ acceleration of gravity,

$\mathbf{r}_{is}$ contact lever body i at contact s,

$\hat{\mathbf{a}}_s$ target acceleration at contact s,

$\mathbf{u}_{hs}$ inward pointing acceleration cone face normals for contact s,

$\mathbf{n}_x, \mathbf{n}_y, \mathbf{n}$ collision coordinate frame with normal $\mathbf{n}$ for a pair $\langle \mathbf{A}, \mathbf{B} \rangle$.

Appendix B

Color Plates

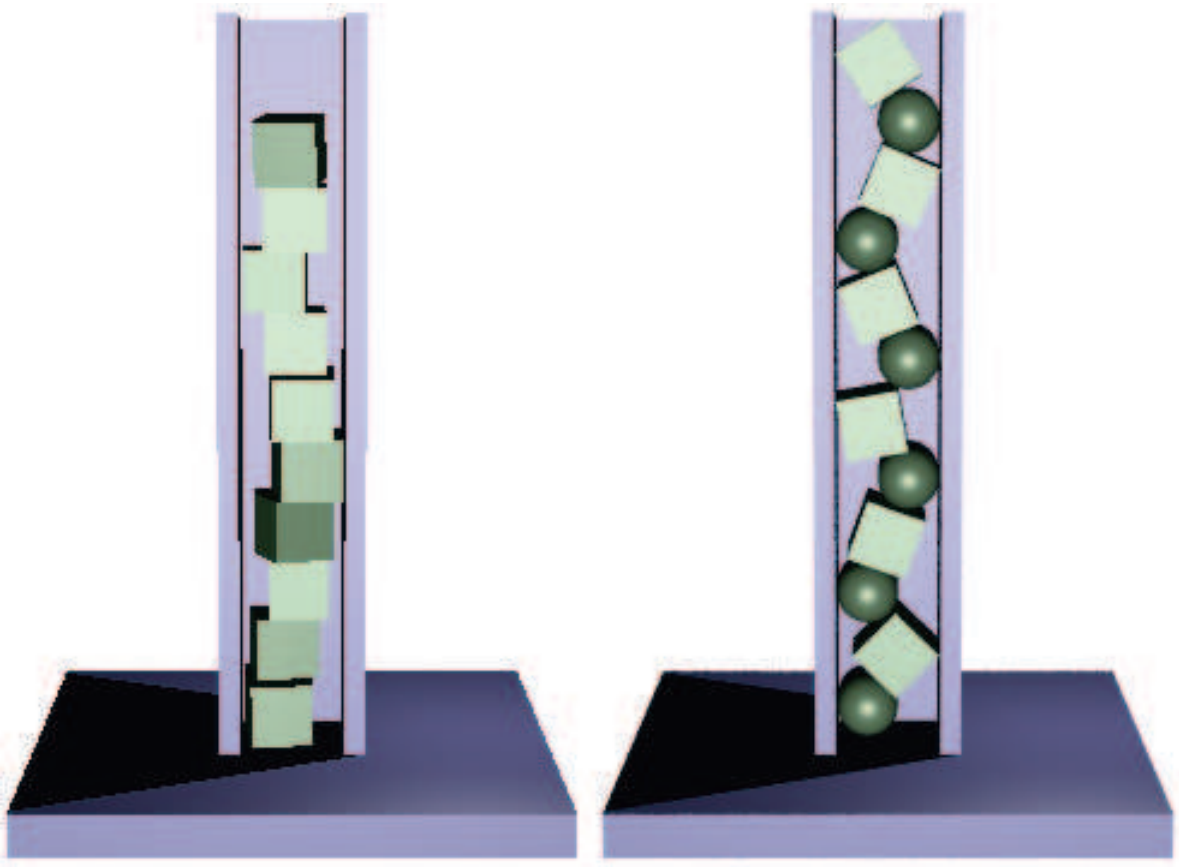
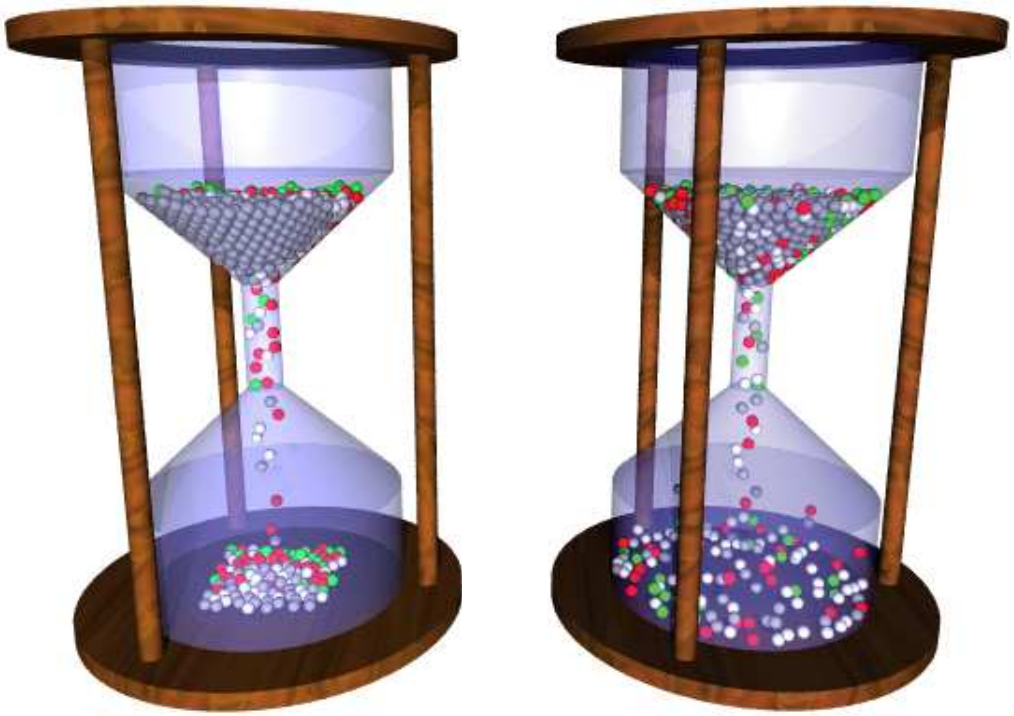


Figure B.1: Simulating stacked bodies is known to be difficult: (a) simple stack of 10 cubes, (b) mixed stack of spheres and cubes.



(a) (b)

Figure B.2: Hourglass simulations: (a) note square-shaped collection of spheres on the floor with position-based physics, (b) the same simulation looks much more lively using OBA.

Figure B.3: Simple multi-body pendulum made of six sticks.

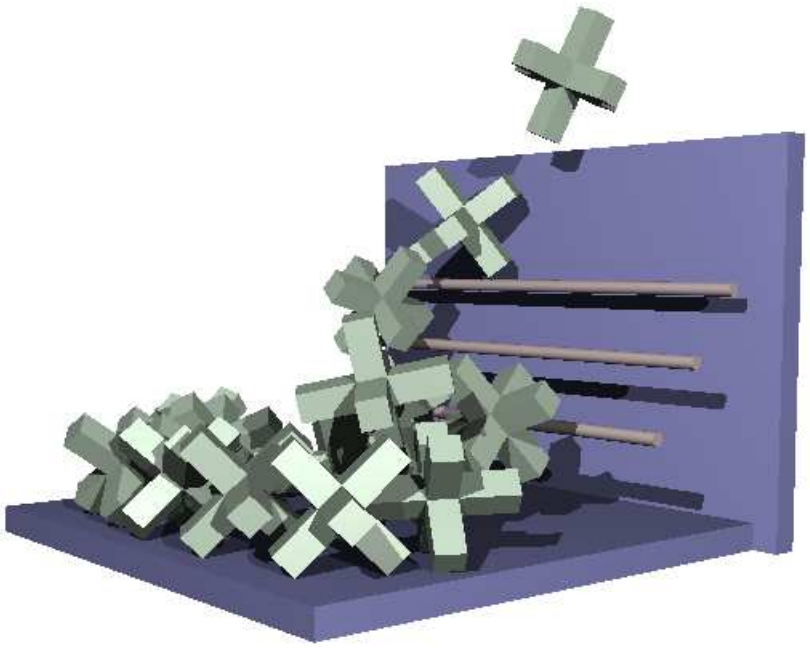


Figure B.4: *Jacks*: an example of non-convexity.

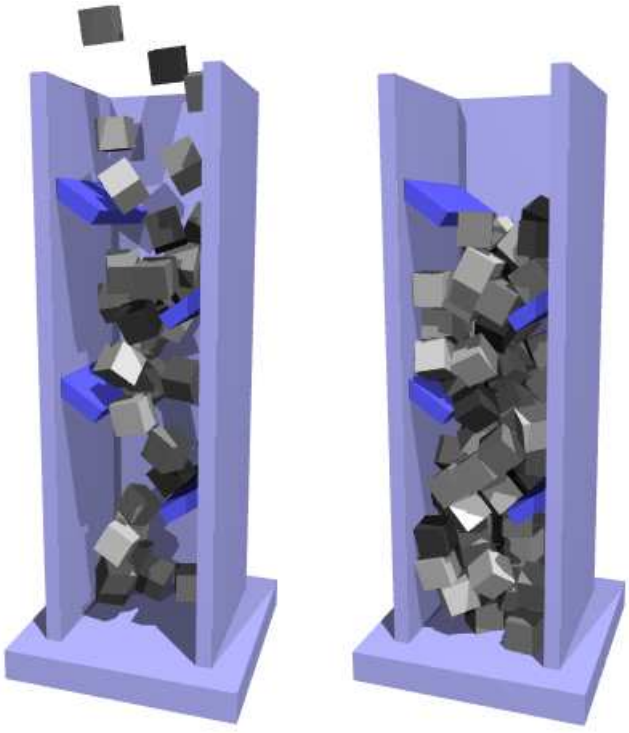


Figure B.5: *Cubejam* after the first cubes started falling, and the finished simulation.

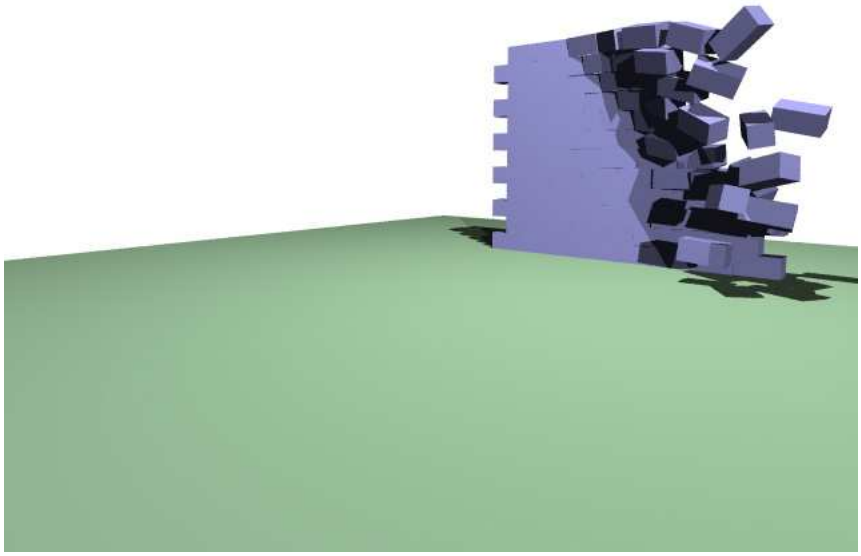


Figure B.6: The *wall* shortly after the first blast wave has hit.

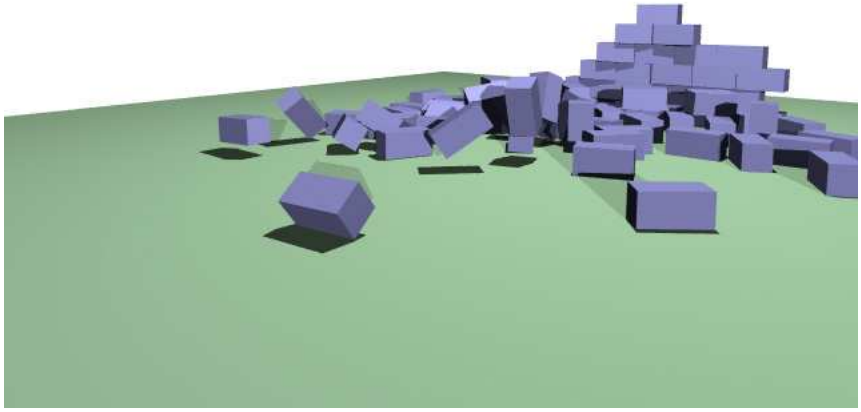


Figure B.7: After the second blast wave: most of the *wall* has collapsed.

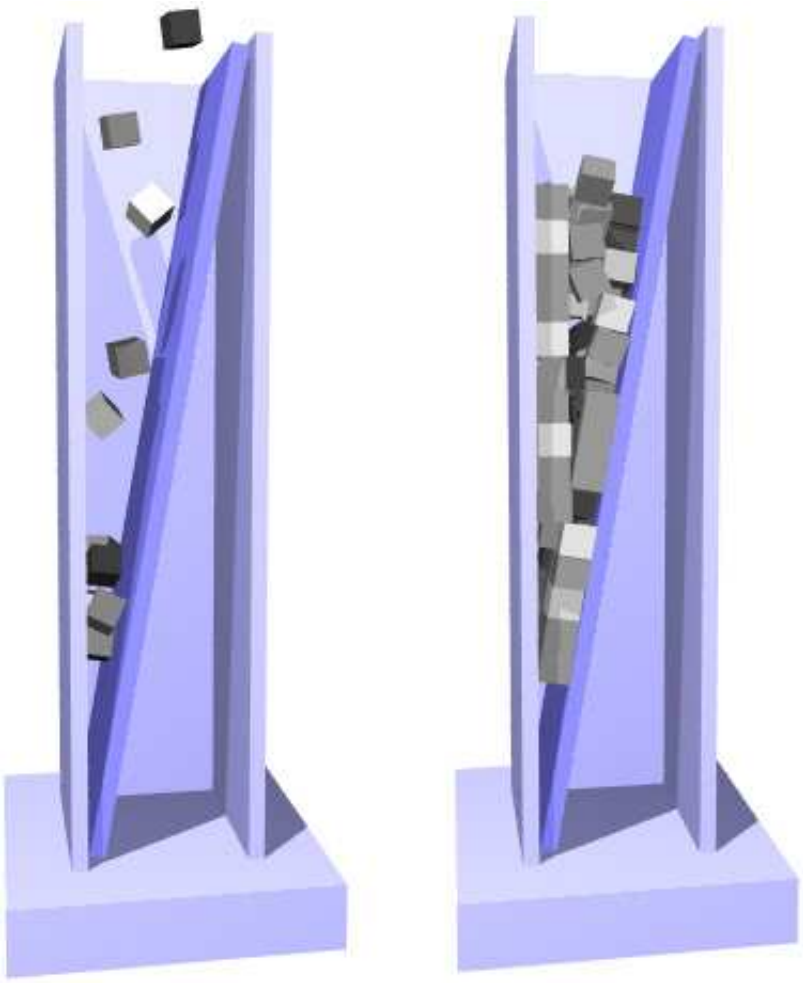


Figure B.8: The *hybrid* shortly after simulation start and at the end when all cubes are tightly wedged into the container.

Figure B.9: The office toy pendulum with only five spheres.

Figure B.10: Detail of *robot* being chased by the claw.

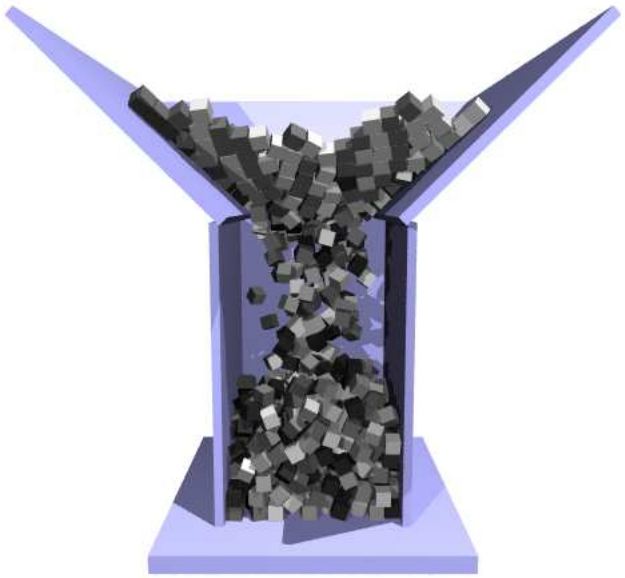


Figure B.11: The 1000 cubes hourglass during simulation.

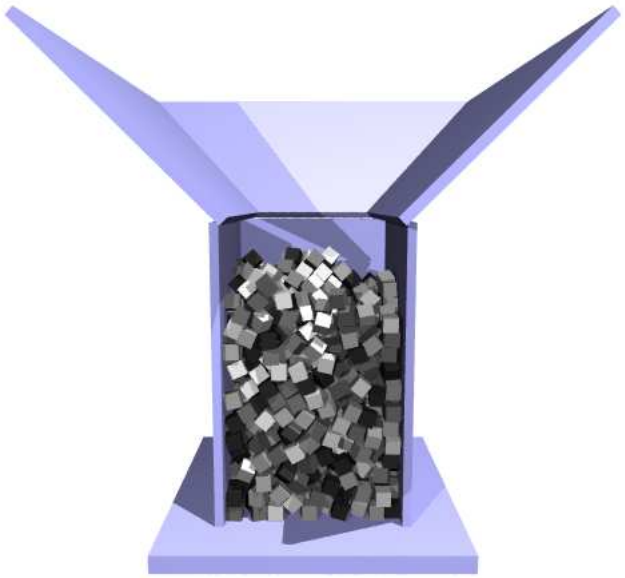


Figure B.12: Same scene at the end with all cubes frozen: the simulation flies.

Bibliography

[1] Ken-ichi Aniyo, Yoshiaki Usami, and Tsuneya Kurihara. A simple method for extracting the natural beauty of hair. *SIGGRAPH 92 Conference Proceedings*, 26(2):111–120, July 1992.

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

[3] D. Baraff. Curved surfaces and coherence for non-penetrating rigid body simulation. *Computer Graphics*, 24(4):19–28, 1990.

[4] D. Baraff. *Dynamic Simulation of Non-Penetrating Rigid Bodies*. Technical report 92-1275, Cornell University, Computer Science Department, 1992.

[5] D. Baraff. Non-penetrating rigid body simulation. *State of the Art Reports, Eurographics '93*, September 1993.

[6] D. Baraff and A. Witkin. Physically based modeling. SIGGRAPH 98 course notes, July 1998.

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

[8] David Baraff. Coping with friction for non-penetrating rigid body simulation. *SIGGRAPH 91 conference proceedings*, 25(4):31–40, July 1991.

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

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

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

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

[13] J. A. Batlle. The sliding velocity flow of rough collisions in multibody systems. *Journal of Applied Mechanics*, 63:804–809, September 1996.

[14] J. A. Batlle and S. Cardona. The jamb (self-locking) process in three-dimensional collisions. *Journal of Applied Mechanics*, 65:417–423, June 1998.

[15] V. Bhatt and Jeff Koechling. Three-dimensional frictional rigid-body impact. *Journal of Applied Mechanics*, 62:893–898, December 1995.

[16] H. B. Bidasaria. A new method for modeling of hair-grass type textures. *Proceedings of the 1995 ACM 23rd annual conference on Computer Science*, pages 109–113, February 1995.

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

[18] D. E. Breen, D. H. House, and M. J. Wozny. Predicting the drape of woven cloth using interactive particles. *SIGGRAPH 94 Conference Proceedings*, pages 365–372, 1994.

[19] Stephen Cameron. Enhancing GJK: computing the minimum and penetration distances between convex polyhedra. *IEEE Int. Conf. Robotics & Automation*, pages 3112–3117, April 1997.

[20] John Canny. Collision detection for moving polyhedra. Technical report, MIT A.I. Lab Memo 806, October 1984.

[21] A. Chatterjee and A. Ruina. A new algebraic rigid-body collision law based on impulse space considerations. *Journal of Applied Mechanics*, 65:939–950, December 1998.

[22] A. Chatterjee and A. Ruina. Two interpretations of rigidity in rigid-body collisions. *Journal of Applied Mechanics*, 65:894–900, December 1998.

[23] S. Chenney and D. Forsyth. View-dependent culling of dynamic systems in virtual environments. *Proceedings of ACM Symposium on Interactive 3D Graphics*, 1997.

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

[25] Terry W. Clark, Reinhard v. Hanxleden, J. Andrew McCammon, and L. Ridgway Scott. Parallelizing molecular dynamics using spatial decomposition. *Proceedings of the Scalable High Performance Computing Conference*, May 1994. available via anonymous ftp from softlib.rice.edu as pub/CRPC-TRs/reports/CRPC-TR93356-S.

[26] 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.

[27] Richard K. Cottle, Jong-Shi Pang, and Richard E. Stone. *The Linear Complementarity Problem*. Academic Press, 1992.

[28] Tony DeRose, Michael Kass, and Tien Truong. Subdivision surfaces in character animation. *SIGGRAPH 98 Conference Proceedings*, pages 85 – 94, 1998.

[29] B. Eberhardt, A. Weber, and W. Strasser. A fast, flexible, particle-system for cloth draping. *IEEE Computer Graphics and Applications*, 16:52–59, 1996.

[30] 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.

[31] Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. Visual simulation of smoke. *SIGGRAPH 01 Conference Proceedings*, pages 15–22, August 2001.

[32] Nick Foster and Ronald Fedkiw. Practical animation of liquids. *SIGGRAPH 01 Conference Proceedings*, pages 23–30, August 2001.

[33] M. Fu, Z. Q. Luo, and Y. Ye. Approximation algorithms for quadratic programming. *Journal of Combinatorial Optimization*, 2:29–50, 1998.

[34] T. A. Funkhouser and C. H. Séquin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. *SIGGRAPH 93 Conference Proceedings*, 27:247–254, 1993.

[35] Christian Gerthsen, H. O. Kneser, and Helmut Vogel. *Physik*. Springer Verlag, 14 edition, May 1982.

[36] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Trans. Robotics & Automation*, 4(2):193–203, April 1988.

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

[38] Radek Grzeszczuk, Demetri Terzopoulos, and Geoffrey Hinton. Neuroanimator: Fast neural network emulation and control of physics-based models. *SIGGRAPH 98 Conference Proceedings*, pages 9–20, 1998.

[39] James K. Hahn. Realistic animation of rigid bodies. *SIGGRAPH 88 Conference Proceedings*, 22(4):299–308, August 1988.

[40] E. Hairer and G. Wanner. Fortran codes. http://www.unige.ch/math/folks/hairer/software.html.

[41] Philip M. Hubbard. Approximating polyhedra with spheres for time-critical collision detection. *ACM Transactions on Graphics*, 15(3):179–210, July 1996.

[42] Suejung Huh, Dimitris N. Metaxas, and Norm Badler. Collision resolutions in cloth simulation. *IEEE-Computer Animation 2001*, pages 122–127, November 2001.

[43] David R. Jefferson. Virtual time. *ACM Transactions on Programming Languages and Systems*, 7(3):404–425, July 1985.

[44] M. Kass and G. Miller. Rapid, stable fluid dynamics for computer graphics. *SIGGRAPH 90 Conference Proceedings*, pages 49–58, 1990.

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

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

[47] Joseph Laszlo, Michiel van de Panne, and Eugene Fiume. Interactive control for physically-based animation. *SIGGRAPH 00 Conference Proceedings*, pages 201–208, 2000.

[48] A. Lewis, R. M'Closkey, and R. M. Murray. Modeling constraints and the dynamics of a rolling ball on a spinning table. Preprint, California Institute of Technology, 1993.

[49] 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.

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

[51] Ming Chieh Lin. *Efficient Collision Detection for Animation and Robotics*. PhD thesis, University of California, Berkeley, 1993.

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

[53] Per Lötstedt. Numerical simulation of time-dependent contact and friction problems in rigid body mechanics. *SIAM Journal of Scientific and Statistical Computing*, 5(2):370–393, June 1984.

[54] Carmine Mangione. Performance tests show Java as fast as C++. http://www.javaworld.com/javaworld/jw-02-1998/jw-02-jperf.html.

[55] D. B. Marghitu and Y. Hurmuzlu. Three-dimensional rigid-body collisions with multiple contact points. *Journal of Applied Mechanics*, 62:725–732, September 1995.

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

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

[58] Victor J. Milenkovic and Harald Schmidl. Optimization-based animation. *SIGGRAPH 01 Conference Proceedings*, pages 37–46, 2001.

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

[60] Brian Mirtich. *Impulse-based Dynamic Simulation of Rigid Body Systems*. PhD thesis, University of California, Berkeley, December 1996.

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

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

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

[64] K. G. Murty. *Linear Complementarity, Linear and Nonlinear Programming.* Heldermann Verlag, Berlin, 1988.

[65] Lars Nyland, Jan Prins, Ru Huai Yun, Jan Hermans, Hye-Chung Kum, and Lei Wang. Achieving scalable parallel molecular dynamics using dynamic spatial domain decomposition techniques. *Journal of Parallel and Distributed Computing*, 47(2):125–138, December 1997.

[66] Lars Nyland, Jan Prins, Ru Huai Yun, Jan Hermans, Hye-Chung Kum, and Lei Wang. Modeling dynamic load balancing in molecular dynamics to achieve scalable parallel execution. *Fifth International Symposium on Solving Irregularly Structured Problems in Parallel*, pages 356–365, 1998.

[67] David A. O'Brien, Susan Fisher, and Ming Lin. Automatic simplification of particle system dynamics. *Proceedings of Computer Animation*, pages 210–219, November 2001.

[68] James F. O'Brien, Perry R. Cook, and Georg Essl. Synthesizing sounds from physically based motion. *SIGGRAPH 01 Conference Proceedings*, pages 529–536, 2001.

[69] P. Painlevé. Sur le lois du frottement de glissemment. *C. R. Acad. Sci. Paris*, 121:112–115, 1895.

[70] J. S. Pang and J. C. Trinkle. Complementarity formulations and existence of solutions of dynamic multi-rigid-body contact problems with coulomb friction. *Mathematical Programming*, 73:199–226, 1996.

[71] Jovan Popovic, Steven M. Seitz, Michael Erdmann, Zoran Popovic, and Andrew Witkin. Interactive manipulation of rigid body simulations. *SIGGRAPH 00 Conference Proceedings*, pages 209–218, 2000.

[72] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C.* Cambridge University Press, 1988.

[73] X. Provot. Deformation constraints in a mass-spring model to describe rigid cloth behavior. *Graphics Interface*, pages 147–155, 1995.

[74] M. B. Rubin. Physical restrictions on the impulse acting during three-dimensional impact of two "rigid" bodies. *Journal of Applied Mechanics*, 65:464–469, June 1998.

[75] 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.

[76] Harald Schmidl and Victor J. Milenkovic. A fast impulsive contact suite for rigid body simulation. submitted December 2001.

[77] Ken Shoemake. Animating rotation with quaternion curves. *SIGGRAPH 85 Conference Proceedings*, 19(3):245–254, July 1985.

[78] Charles E. Smith and Pao-Pao Liu. Coefficients of restitution. *Journal of Applied Mechanics*, 59:963–969, December 1992.

[79] 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.

[80] David E. Stewart. Rigid-body dynamics with friction and impact. *SIAM Review*, 42(1):3–39, 2000.

[81] Subhash Suri, Philip M. Hubbard, and John F. Hughes. Analyzing bounding boxes for object intersection. *ACM Transactions on Graphics*, 18(3):257–277, July 1999.

[82] D. Terzopoulos and K. Fleischer. Deformable models. *Visual Computer*, 4:306–331, 1988.

[83] D. Terzopoulos and K. Fleischer. Modeling inelastic deformation: Viscoelasticity, plasticity, fracture. *SIGGRAPH 88 Conference Proceedings*, 22:269–278, August 1988.

[84] D. Terzopoulos, J. C. Platt, and A. H. Barr. Elastically deformable models. *SIGGRAPH 87 Conference Proceedings*, 21:205–214, 1987.

[85] D. Terzopoulos and H. Qin. Dynamic nurbs with constraints for interactive sculpting. *ACM Transactions on Graphics*, 13:103–136, 1994.

[86] J. C. Trinkle, J. A. Tzitzouris, and J. S. Pang. Dynamic multi-rigid-body systems with concurrent distributed contacts: Theory and examples. *Philosophical Transactions on Mathematical, Physical, and Engineering Sciences*, December 2001. Preprint.

[87] P. Violino, M. Courchesne, and N. Magnenat Thalmann. Versatile and efficient techniques for simulating cloth and other deformable objects. *SIGGRAPH 95 Conference Proceedings*, pages 137–144, 1995.

[88] Andrew Witkin, Michael Gleichner, and William Welch. Interactive dynamics. *Computer Graphics*, 24(4):11–22, March 1990.

VITA

Harald Schmidl was born in Ingolstadt/Donau, Bavaria, Germany. He received his entire pre-university education in Ingolstadt. In October 1988 he entered the Universität Regensburg from which he graduated in October 1991 with the equivalent to a Bachelor of Science in Physics. From November 1991 until August 1993 he studied graduate-level Physics at the Eberhard-Karls Universität Tübingen. During a one year exchange program at the University of Miami from August 1993 until May 1994 he obtained a teaching assistantship in Computer Science and graduated with a Computer Science Masters degree in May 1996. He went into industry to work as a multimedia-software developer until December 1997. In January 1998 he returned to the University of Miami to pursue a Ph.D. degree in Interdepartmental Studies. He was granted the degree of Doctor of Philosophy in May 2002.