Contact and Friction Simulation for Computer Graphics

SHELDON ANDREWS, École de technologie supérieure, Canada
KENNY ERLEBEN, University of Copenhagen, Denmark
ZACHARY FERGUSON, New York University, USA

Efficient simulation of contact is of interest for numerous physics-based animation applications. For instance, virtual reality training, video games, rapid digital prototyping, and robotics simulation are all examples of applications that involve contact modeling and simulation. However, despite its extensive use in modern computer graphics, contact simulation remains one of the most challenging problems in physics-based animation.

This course covers fundamental topics on the nature of contact modeling and simulation for computer graphics. Specifically, we provide mathematical details about formulating contact as a complementarity problem in rigid body and soft body animations. We briefly cover several approaches for contact generation using discrete collision detection. Then, we present a range of numerical techniques for solving the associated LCPs and NCPs. The advantages and disadvantages of each technique are further discussed in a practical manner, and best practices for implementation are discussed. Finally, we conclude the course with several advanced topics such as methods for soft body contact problems, barrier functions, and anisotropic friction modeling. Programming examples are provided in our appendix as well as on the course website to accompany the course notes.

SIGGRAPH '22 Courses, August 07-11, 2022, Vancouver, BC, Canada

ACM ISBN 978-1-4503-9362-1/22/08.
https://doi.org/10.1145/3532720.3535640

CCS Concepts: • **Computing methodologies** → **Simulation by animation**; **Physical simulation**; *Real-time simulation*; • **Applied computing** → *Physics*.

Additional Key Words and Phrases: contact, friction, complementarity problems, Newton methods, iterative methods, pivoting methods, splitting methods, physics-based animation

ACM Reference Format:
Sheldon Andrews, Kenny Erleben, and Zachary Ferguson. 2022. Contact and Friction Simulation for Computer Graphics. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Courses (SIGGRAPH '22 Courses), August 07-11, 2022*. ACM, New York, NY, USA, 172 pages. https://doi.org/10.1145/3532720.3535640

Contents

PREFACE

Why a SIGGRAPH course on contact and friction simulation? The answer, quite simply, is that it is an important topic. As a classic problem in physics-based animation, contact simulation has been the focus on many scientific works over the past several decades in computer graphics. Furthermore, there is a demand for friendly teaching material about the topic since it is difficult to approach for non-specialists.

Contact and friction simulation in computer graphics field has evolved a lot since the 1980s. During this time, a lot of effort has gone into uncovering fast and robust methods, and leveraging hardware such as GPUs. The performance and robustness of these methods now enables techniques developed in computer graphics to be deployed into other fields. For instance, robotics and medical simulation are two fields where computer graphics simulations are now pushing the limits of the state-of-the-art. Digital prototyping, learning and training are other important application contexts, and there is still a need for advances here. Currently, differentiable physics, rich friction models, machine learning-based physics, soft-rigid and soft-soft body contact are open topics that the research community is pursuing. We hope these notes will give the next generation of researchers the necessary foundation to begin solving these challenges.

These course notes have been assembled over a period of several years. Topics have been introduced and pulled out continuously in this process, and we will likely continue to do so in future versions of these course notes. A few topics kept coming back to us as being more fundamental, in that they form the foundation for understanding a lot of recent work in the field. Hence, this material makes up the core of the course notes, and they are mainly about understanding the physical models and numerical methods. By that we mean the mathematical equations we write up when we describe the models, along with the methods we apply to compute solutions for our models. Ideally, we try to keep models and methods separate, as they should be. Historically, a huge part of the field has been devoted to fast and robust numerical methods, and in particular constraint-based approaches, such as the ones based on complementarity problem formulations, make up a big part of the picture. Therefore, we devote a lot of effort to give a firm understanding of this popular approach.

New additions in this version of the course include coverage of multivariate continuous collision detection, an extended treatment of penalty-based methods to include barrier methods, and a new chapter on implicit time integration schemes and methods for solving soft body contact problems has been added. Additionally, we now include a more in-depth coverage of anisotropic friction simulation, as well as a tutorial that guides the reader through the steps of creating a rigid body simulator with frictional contact.

It is impossible to cover all topics related to contact and friction simulation in a three-hour SIGGRAPH course, and we had to cut out many topics in order to remain relatively concise. For instance, in this version of the course we ignore impulse-based methods that use collision laws and event propagation. The coverage of collision detection is also limited to just the principled

ideas of how to generate contact points for common shape representations. However, we do introduce the cone-complementarity problem formulation for modeling contact and friction, as these are a recent very interesting addition to the field, yet only brief details about the numerical methods related to this type of model are covered. Lastly, we do not cover position-based and projective dynamics directly. However, solvers applied here bear a resemblance to the proximal operators and iterative solvers, which we do cover in full details. There is a bundle of more work that deserves mentioning here, and we apologize for not having the time and space to give the wonderful work on those topics the attention they deserve.

We would like to acknowledge that a lot of people have been involved in the creation of these course notes. A community of peers has helped us to generate ideas and discussion how to best present and explain topics, given feedback, proofread the content at various stages of development, pointed us to relevant work in the field, and much more. The list is long, but in particular we would like to acknowledge the support and help from Miles Macklin, Paul Kry, Mihai Francu, Eric Paquette and many others. We also thank Greg Klár, Löic Nassif, and Bin Wang for sending us corrections to the first version of these course notes.

Finally, we hope that you will enjoy this course, and that you find contact and friction simulation a fun topic to learn about! Please visit the course website siggraphcontact.github.io, where you will find links to programming examples and other supplementary material related to these notes.

Sheldon, Kenny, and Zach

SYLLABUS

Introduction to Contact Simulation. We introduce the idea of time stepping a simulation and present the ordinary differential equations (ODE) that govern the Newton dynamics of a simulation. Stepping in time an ODE is solved using an integrator and for this we introduce the semi-implicit and implicit Euler methods. Kinematic and geometric constraints are added using the method of Lagrange multipliers, and in this framework a unilateral constraint for non-interpenetration is realized by defining a gap function between bodies. We explain how the complementarity conditions generate constraint impulses that vanish when contact breaks. The concept of maximal dissipation is also introduced and used to derive a model of friction. Non-linear and linearized versions of Coulomb friction are presented. Finally, constraint stabilization is presented as a remedy for constraint errors.

Contact Generation. This section focuses on the practicalities of generating contacts. The primary focus is on guidelines for efficient and robust collision detection, and extracting the contact information from various surface representations. Collision detection methods for primitive geometric features and signed distance fields are presented. Finally, we cover both univariate and multivariate formulations for continuous collision detection.

Numerical Methods. We focus on the underlying mathematical derivation and recast the physical model into a problem that can be solved using various numerical techniques. First, pivoting methods are presented for solving the contact linear complementarity problem by estimating the index set of simulation variables. We then introduce the fixed-point methods as a framework for solving general non-linear models, and proceed to derive the projected Gauss Seidel, SOR, and Jacobi methods. Finally, we reformulate the problem of satisfying the contact conditions and equations of motion as a root finding problem, providing the basis for Newton-type methods, such as minimum-map and Fischer-Burmeister reformulations. The importance of preconditioning and regularization are also emphasized as a way to achieve performance and robustness.

Soft Body Contact Approaches. Various implicit time-integration schemes for soft body contact are presented. First, root-finding approaches are covered, where both the primal and dual forms are considered. Next, time-integration of soft bodies with contact is reformulated as a constrained minimization problem, and velocity-level and position-level versions of the problem formulation are given. The section concludes with an extensive treatment of methods for computing the system matrix of the dual form, along with the implications of each approach.

Penalty and Barrier Methods. This section covers penalty and barrier methods. In particular, the latter have demonstrated robust contact handling by applying barrier potentials within an optimization-based implicit time integration framework. We demonstrate how projected Newton methods can efficiently solve these optimization problems.

Anisotropic Friction. Several recent approaches for simulating anisotropic friction are surveyed. Details about how numerical solvers may be extended to support anisotropic friction cones are also provided.

1 INTRODUCTION TO CONTACT SIMULATION

Contact simulation is mainly concerned with computing the forces that exist at interfaces between physical objects. Physical objects can range from rigid bodies, such as a billiard ball or vehicle chassis, to soft bodies, such as cloth and hair. We simply refer to a collection of such objects and the forces acting on them and between them as a physical system. The interface forces are termed *contact forces* and they prevent objects from penetrating each other during motion and deformation, and they additionally model the resistance due to objects against each other. These course notes are primarily focused on how to model and compute contact forces in an efficient and robust manner.

Historically, constraint based methods for computing contact forces have been used by the graphics community for simulation of rigid bodies. Many earlier works focused on improving performance by developing faster methods for rigid body simulations. Hence, a lot of the notation we use in these notes are influenced by these early works. One might primarily read these notes while thinking about contact between rigid bodies, and indeed many of the examples demonstrate the rigid case as it is the easier case after all. Nevertheless, much of the material we cover here applies to elastic models and soft bodies too, and we will address some of these differences when they arise. Furthermore, a complicating factor in contact simulation is the generation of contacts using collision detection, which is a topic that we cover briefly in order to convey the principal ideas.

At a high-level, there are three main paradigms for contact simulation. First, contact may be simulated by constraint based methods, and these course notes are rooted in this paradigm. This paradigm bares a lot of resemblance to constrained optimization where constraints are solved exactly. Another paradigm is penalty based approaches, and these tend to rely on force-based modeling to resolve interpenetration at the interface. Often this paradigm is introduced conceptually as small abstract springs working between objects to keep them from overlapping. Hence, the springs "penalize" overlap which gives rise to the name for this class of methods. There are similarities here to penalty methods in numerical optimization, in particular barrier methods that have recently become popular for contact simulation. The last paradigm accounts for impulse-based methods where the contact between objects is conceptualized as sequences of micro-collisions. These notes take their outset in the constraint based approaches as these have been dominating real-time interactive simulation for decades. Constraint-based methods also have the nice property that they are able to guarantee solving for contact exactly, at least in such a way that constraints are always fulfilled. However, we later explain formulations based on penalty methods and explain best practices when using this class of approach for modeling contact in simulation pipelines. 6

These course notes seek to not only give the reader intuition about the theoretical models that are used as the basis for a wide variety of contact simulations, but also to highlight some best practices for all stages of the simulation pipeline. Note that the “An Introduction to Physics-Based Animation” SIGGRAPH course [Bargteil et al. 2020] is an excellent related

reference for an in-depth discussion of preliminaries, while this course on contact and friction naturally picks up from where the other course wraps up.

We begin by introducing the equations of motion and numerical integration by time stepping. Then, kinematic constraints are derived to resolve non-interpenetration and we explain the importance of the complementarity condition. This leads into a presentation of Coulomb friction and how it is included in the formulation of the contact problem. Finally, the introduction ends with a discussion on how the formulation may be adapted to soft body simulations.

1.1 The Equations of Motion

The Newton-Euler equations of motion [Goldstein et al. 2002] that govern the dynamics of a physical system form a second-order ODE that can be written as

$$\mathbf{M}(t)\,\dot{\mathbf{u}}(t) = \mathbf{f}(\mathbf{q}(t), \mathbf{u}(t), t)\,, \tag{1}$$

where $\mathbf{M}(t)$ are the masses, $\mathbf{q}(t)$ are the generalized positions, $\mathbf{u}(t)$ are the generalized velocities, and $\mathbf{f}(t, \mathbf{q}(t), \mathbf{u}(t))$ are the generalized forces acting on the system, which depend on the positions and velocities. Notice that we are agnostic about what we are simulating at this point, and the equation can be seen as describing the motion of rigid bodies, thin shells, or elastic solids. Also notice that all terms in Equation 1 depend on the time t. However, in computer graphics, we are often only interested in determining the dynamical behavior at a particular instant in time, and so the Newton-Euler equations may be written more succinctly as

$$\mathbf{M}\,\dot{\mathbf{u}} = \mathbf{f}\,. \tag{2}$$

The time instants for evaluating Equation 2 are determined by a numerical integrator, and in the next section we briefly outline how to "step" the equations of motion using a popular integration method in computer graphics.

1.2 Time Integration

Physical simulations for computer graphics are typically performed using a discrete numerical integration with time step h. There are two main types of numerical integrators: explicit and implicit. The former evaluate the force at the beginning of the time step $\mathbf{f}(\mathbf{q}^-, \mathbf{u}^-)$, while the latter evaluate it at the end $\mathbf{f}(\mathbf{q}^+, \mathbf{u}^+)$. Note that the superscript $\square^+$ is used to indicate an implicit quantity and $\square^-$ an explicit quantity. We can denote the forces for now simply as $\mathbf{f}$ without specifying whether they are implicit or explicit, and this particular choice will depend upon the application context. However, we will drop the latter from here on since computer graphics applications largely use implicit integration schemes due to their increased numerical stability. A Taylor expansion of the implicit velocities gives the first-order approximation

$$\mathbf{u}^+ \approx \mathbf{u} + h\dot{\mathbf{u}}\,,$$

where the change in velocities from the start of the time step to the end of the time step is determined by the accelerations $\dot{\mathbf{u}}$. Using this approximation to substitute for the accelerations in Equation 2, and by applying a simple Euler integration scheme, we can write the linear relationship governing the motion of a system with n degrees freedom at each time step as

$$\mathbf{M}\mathbf{u}^+ = \mathbf{M}\,\mathbf{u} + h\,\mathbf{f} \tag{3}$$

with the mass matrix $\mathbf{M} \in \mathbb{R}^{n\times n}$, the momentum term $\mathbf{M}\,\mathbf{u} \in \mathbb{R}^n$ and applied forces $\mathbf{f} \in \mathbb{R}^n$. Here, the velocities $\mathbf{u}^+ \in \mathbb{R}^n$ are determined at the end of the time step, and for simplicity we will consider the mass matrix $\mathbf{M}$ to be constant over the time step. For a rigid body system, the mass matrix will consist of small blocks containing the total mass of the rigid bodies and their 3×3 inertia tensors, whereas for soft bodies the mass matrix can be given by a volume integral of mass density over some elements or a diagonal matrix of nodal mass values, like in a particle system. The specific form can vary depending on the chosen discretization method (such as finite elements, finite volumes or mass-spring systems).

An interesting observation is that Equation 3 gives the velocity-level Newton-Euler equations of motion, and by evaluating the dynamics over a time step h this effectively transforms the instantaneous forces into *impulses*. Impulses are instantaneous quantities that abruptly change the value of the momentum due to the action of a force over a small period of time. But we can also consider them as we do here as integrals over the period h of the force $\mathbf{f}$.

Solving the linear system in Equation 3 to determine the velocities is an important step in the numerical integration of a physical system. The velocities may then be used to update the positions in an implicit fashion by

$$\mathbf{q}^+ = \mathbf{q} + h\,\mathbf{H}(\mathbf{u}^+),$$

where $\mathbf{H}$ defines a kinematic mapping between the generalized velocities $\mathbf{u}$ and positions $\mathbf{q}$. The kinematic mapping is needed when the generalized positions $\mathbf{q}$ require more than n components. This is usually the case when using redundant parameters like quaternions or orthogonal matrices for representing rotations.

In contact simulation, simple Euler schemes such as the one described in this section are often considered sufficient. The reason for this is that contact models often assume a myopic view point of a flat planar surface at the interface. This means motion with respect to these models can be described in a linear fashion. For instance, the Coulomb model we introduce later is for planar sliding motion only. As such, one will not get better performance or accuracy from higher order integration schemes as they will be limited by the linear models of contact. However, higher order integration schemes can be very efficient for simulating the free motion of rigid bodies and elastic models without contact, and there is some work that considers contact models for curved surfaces. In this case, higher order time integration may make sense.

Next, we consider how constraint equations may be used to couple the movement of the degrees of freedom of a dynamical simulation.

1.3 Constraints

Kinematic constraints can be used to couple the movement of bodies in a simulation, and indeed this is exactly the approach we will describe in this section for simulating contact. A collection of m constraint functions $\phi(\mathbf{q}) \in \mathbb{R}^m$ implicitly define a manifold that is embedded in the n-dimensional space of the simulation degrees of freedom, and movement of constrained bodies is restricted to these manifolds. Observe that a constraint defines a relationship based on the degrees of freedom of the system, $\mathbf{q}$.

There are two types of constraint equations commonly found in dynamics simulation: *bilateral* and *unilateral*. Bilateral constraint functions have the form $\phi(\mathbf{q}) = 0$. For instance, hinges, ball-and-socket, and prismatic joints are modeled using bilateral constraint functions. Whereas functions of the form $\phi(\mathbf{q}) \geq 0$ are called unilateral constraints.

By assuming that constraints are initially satisfied, and that the constraint equations will not be violated if there is no movement that leads to violation (i.e., movement is limited to the constraint manifold), we can instead formulate the constraint equations in terms of the velocities by computing the gradient of $\phi(\mathbf{q})$ with respect to $\mathbf{q}$, such that $\mathbf{J} = \frac{\partial \phi(\mathbf{q})}{\partial \mathbf{q}} \in \mathbb{R}^{m \times n}$ contains the constraint gradients of the m constraint equations. In rigid body systems, there is often a dimensionality mismatch when changing from positions to velocities due to the orientation. For instance, if quaternions are used to store body rotations, a kinematic map is needed. Hence, we have $\mathbf{J} = \frac{\partial \phi(\mathbf{q})}{\partial \mathbf{q}} \mathbf{H} \in \mathbb{R}^{m \times 7N}$ for N bodies. We describe the kinematic mapping in detail in Section 1.10.1. For soft bodies, typically no such mapping is required. The velocity level constraint equations can then be written as

$$\mathbf{J}\,\mathbf{u} = 0 \tag{4}$$

for bilateral constraints, and

$$\mathbf{J}\,\mathbf{u} \geq 0 \tag{5}$$

for unilateral constraints. The constraints can be imposed through the inclusion of constraint forces in the dynamical equations. These are forces that act in the direction of the constraint gradient $\mathbf{J}$, which encodes the directions in which bodies may be "pushed" or "pulled" without doing any real work on the system, but that will force bodies to remain on the constraint manifold. Thus, the constraint forces of a system are computed as

$$\mathbf{f}_c = \mathbf{J}^T \boldsymbol{\lambda}, \tag{6}$$

where $\boldsymbol{\lambda}$ are Lagrange multipliers. These multipliers can be interpreted as the magnitudes of the constraint forces if the constraint directions (i.e., gradients) are normalized.

In order to enforce the position constraints $\phi(\mathbf{q}) = 0$ or $\phi(\mathbf{q}) \geq 0$, we need to integrate the constraint forces in Equation 6 by applying the impulse $h\mathbf{f}_c$. Therefore, the notation $\boldsymbol{\lambda} \equiv h\,\boldsymbol{\lambda}$ is used to denote constraint impulse magnitudes. Notice that lambda changes font when we include the time step to make it an impulse magnitude. It may feel a bit confusing with two different notations for these lambdas, but it helps underlining when we talk about a force-based

or impulse-based quantity, i.e. whether time integration has been done to get to a velocity level form of our models. In computer graphics, the velocity-based form is dominant, and hence we hope the reader will focus more on the mathematical and algebraic forms used for describing the different kind of constraints.

Revising the equations of motion in Equation 3, we include the constraint impulse magnitudes as an implicit term, such that

$$\mathbf{M}\,\mathbf{u}^{+} - \mathbf{J}^{T}\boldsymbol{\lambda}^{+} = \mathbf{M}\,\mathbf{u} + h\mathbf{f}, \tag{7}$$

where $\boldsymbol{\lambda}^{+} \in \mathbb{R}^{m}$ is a vector of constraint impulse magnitudes. Note that we have not specified when $\mathbf{J}$ is evaluated, and in this course we will consider it to be explicit and constant throughout the time step. And so, while the constraint forces are integrated explicitly, it is only in terms of direction, as the Lagrange multipliers enforce the constraint at the end of the time step, i.e., $\mathbf{J}\mathbf{u}^{+}$. However, there are cases when $\mathbf{J}$ it is treated implicitly, and thus requires evaluating the gradient of $\phi(\mathbf{q} + h\Delta\mathbf{u}^{+})$ and resolving for the constraint forces in an iterative fashion. This may mean performing additional collision detection tests, which can negatively impact performance.

For a moment, let us consider just the bilateral constraints in the simulation. The linear system combining the equations of motion and the velocity constraint equations can be written as

$$\begin{bmatrix} \mathbf{M} & -\mathbf{J}^{T} \\ \mathbf{J} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{u}^{+} \\ \boldsymbol{\lambda}^{+} \end{bmatrix} = \begin{bmatrix} \mathbf{M}\,\mathbf{u} + h\,\mathbf{f} \\ 0 \end{bmatrix}. \tag{8}$$

We use the first row to solve for $\mathbf{u}^{+}$ and substitute our result into the second row to obtain a reduced system that we can use to solve for $\boldsymbol{\lambda}^{+}$. This technique is called forming the Schur complement of the upper left block in Equation 8 and it results in the reduced linear system

$$\underbrace{\left[\mathbf{J}\,\mathbf{M}^{-1}\,\mathbf{J}^{T}\right]}_{\mathbf{A}}\boldsymbol{\lambda}^{+} + \underbrace{\mathbf{J}\,\mathbf{M}^{-1}(\mathbf{M}\mathbf{u} + h\mathbf{f})}_{\mathbf{b}} = 0\,, \tag{9}$$

which is a commonly used form to solve for the constraint impulses. Once $\boldsymbol{\lambda}^{+}$ are known, the motion of the degrees of freedom can be recovered using Equation 7. Note that for rigid bodies and particle systems using lumped masses, the block diagonal form of $\mathbf{M}$ makes it trivial to invert, and the resulting matrix $\mathbf{A}$ will also be positive semi-definite. Whereas, for implicitly integrated elastic systems, they will have sparse damping and stiffness matrix contributions added to the mass, leading us to choose solvers for Equation 9 that do not involve inversion of the upper left block of Equation 8.

In the next section, we consider that contact can be modeled as a unilateral constraint with some special *complementarity* conditions on the constraint impulses and the relative velocities of bodies.

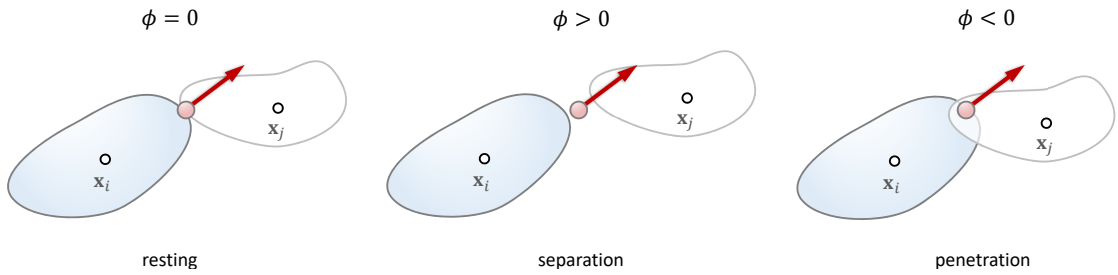


Fig. 1. Illustration of the three contact states determined by the value of the gap function. Observe that a single function value allows us to classify the state.

1.4 Non-interpenetration Contact Constraint

Unlike bilateral constraints, contact constraints are only active when a collision exists between two bodies in the simulation. The state of this relationship between bodies is represented using a gap function, ϕ, which also happens to be the constraint function. The gap function, measures the distance between two bodies, and a positive value $\phi > 0$ indicates a separation between the bodies (see Figure 1).

At locations where the gap is zero or negative, $\phi \leq 0$, there is a contact between the bodies. For smooth objects, their unit surface normals will face in opposite directions at the point of contact. This defines a contact plane containing the point of contact and having a normal vector that is parallel to one of the surface unit normals. Both the contact point and normal direction are determined during the collision detection phase. For discretized geometries, such as triangle meshes, it is more tricky to define the contact plane and we discuss this later in Section 2. For now we can continue using our idealized smooth concept of a contact plane.

An impulse with magnitude $\lambda_{\hat{n}}$ is applied at the contact location in order to keep the bodies from interpenetrating. The non-interpenetration impulse is applied in a direction that is perpendicular to the contact plane. That is, in the normal direction of the surfaces. This leads to two considerations when applying contact forces. Either the bodies are not in contact and there is no contact force, in which case

$$\phi > 0, \quad \lambda_{\hat{n}} = 0\,,$$

or the bodies are touching (for instance, there is resting contact) and a non-zero contact force is applied at the contact location, in which case

$$\phi = 0, \quad \lambda_{\hat{n}} > 0\,.$$

These two cases are exclusive, and often they are succinctly written as the complementarity conditions of contact as

$$0 \leq \phi \quad \perp \quad \lambda_{\hat{n}} \geq 0\,. \tag{10}$$

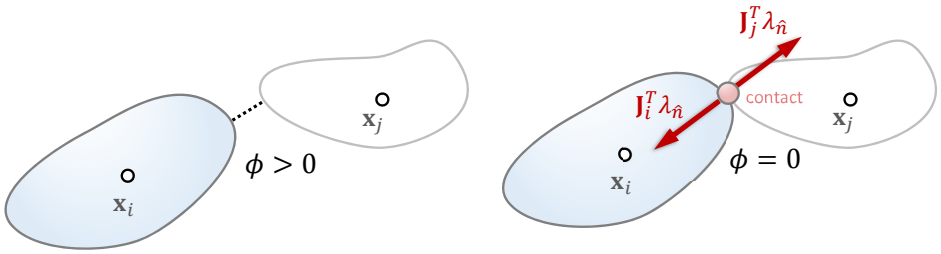


Fig. 2. Left: Two bodies i and j not in contact have a positive gap function $\phi > 0$. Right: A non-interpenetration constraint is created between two colliding bodies. The gap function is zero, and the constraint is enforced by an impulse with magnitude $\lambda_{\hat{n}}$ and the direction of the impulse is determined by the constraint Jacobian $\mathbf{J} = \begin{bmatrix} \mathbf{J}_i & \mathbf{J}_j \end{bmatrix}$.

Notice that we have used the complementarity operator "$\perp$". If a and b are two scalars, then $0 \leq a \perp b \geq 0$ is defined to mean that $a \geq 0$, $b \geq 0$, and $a\,b = 0$. The $\perp$ notation is a short hand version that means the same thing.

The Equation 10 is termed the position level non-penetration complementarity condition. Observe that this type of condition says nothing about the tangential motion. Hence, objects can be in sustained touching contact while sliding relative to each other. Non-sliding sustained contact state is often referred to as a resting contact.

Figure 2 illustrates the gap function before and after collision, which is positive and zero, respectively. Usually, when the bodies are sufficiently close to touch, a non-interpenetration constraint is created between the two bodies in order to keep them from overlapping. The direction of the non-interpenetration impulse is determined by the contact plane, and the impulse $\mathbf{J}^T \lambda^+$ is applied to both bodies involved in the collision.

Although the positions are used by collision detection to determine if two bodies are in contact, the constraints in Equation 8 are formulated at the velocity-level . Recognizing that contact constraints are generated only when $\phi = 0$, we can reason about conditions on the velocities of bodies during contact.

Recall that with semi-implicit and implicit integration techniques, the velocities at the end of a time step are used to advance the simulation. Therefore, if two bodies are touching in the current time step, they will not be touching in the subsequent time step if $\dot{\phi} > 0$. This is because the two bodies will separate due to a positive relative velocity at the contact location. In this case, there is no need to apply a contact force, and $\lambda_{\hat{n}} = 0$. However, if the relative velocity is zero, $\dot{\phi} = 0$, then a force is needed to prevent the bodies from further interpenetrating, i.e., $\lambda_{\hat{n}} > 0$. Therefore, the velocity level non-penetration complementarity condition may be formally stated as

$$0 \leq \dot{\phi} \quad \perp \quad \lambda_{\hat{n}} \geq 0. \tag{11}$$

Here, we note that $\dot{\phi}$ is the normal component of the relative velocity between the bodies at a specific contact point. Hence, we use the notation $\dot{\phi} = v_{\hat{n}}$ for the constraint velocities in the remainder of the notes. This means that if there is a separation velocity $v_{\hat{n}} > 0$ in the normal direction, then there can be no normal contact impulse. On the other hand, if there is a normal contact impulse such that $\lambda_{\hat{n}} > 0$, then there is a *resting* contact and therefore $v_{\hat{n}} = 0$.

Observe that Equation 11 is the result of approximating Equation 10 by substituting the first order Taylor expansion for the gap function, $\phi^+ \approx \phi + h\, v_{\hat{n}}$, and applying the knowledge that $\phi = 0$. Changing the expansion point leads to different schemes and can be used to foresee contact as well as correcting drifting error that arise during simulation from computing approximate solutions or numerical precision. Detecting and adding constraints prior to touching contact would result in the scheme using the non-penetration complementary condition

$$0 \leq (\phi + h v_{\hat{n}}) \quad \perp \quad \lambda_{\hat{n}} \geq 0\,, \tag{12}$$

where $\phi > 0$ since contact has not yet happened. This is essentially an explicit first order approximation of the gap function. This form is popular to avoid tunneling artifacts of objects as it anticipates future contact points, and this idea is further discussed in Section 2 of these notes.

1.4.1 Non-interpenetration Jacobian. Next, let us consider how to compute $v_{\hat{n}}$. We will first present the ideas for the case of rigid bodies and following this we will demonstrate how to work with soft bodies. Noting that $v_{\hat{n}}$ is the rate of change of the gap function, $\dot{\phi}$, and by applying some straightforward calculus, we get

$$v_{\hat{n}} = \dot{\phi} \equiv \frac{\partial \phi}{\partial t} = \frac{\partial \phi}{\partial \mathbf{q}} \frac{\partial \mathbf{q}}{\partial t} = \underbrace{\frac{\partial \phi}{\partial \mathbf{q}} \mathbf{H}}_{\mathbf{J}} \mathbf{u} = \mathbf{J}\,\mathbf{u}\,. \tag{13}$$

In the last step, we have used the kinematic relationship for rigid bodies $\dot{\mathbf{q}} = \mathbf{H}\,\mathbf{u}$ from Equation 1.2. This gives us a mathematical understanding of $\mathbf{J}$. Looking more closely we notice that the role of the Jacobian matrix is to map body-space velocities into a velocity for the gap function. That is, we map from body-space to measure how fast the gap function is changing.

Using this insight, $\mathbf{J}$ for the non-penetration constraints may be constructed more precisely. Consider two rigid bodies, A and B, with centre of mass positions $\mathbf{x}_A$ and $\mathbf{x}_B$ that are colliding at contact point $\mathbf{p}$ with unit contact normal $\hat{n}$ pointing from A towards B. The relative velocity in the normal direction $v_{\hat{n}}$ is then given by

$$v_{\hat{n}} = \hat{n} \cdot \Delta \mathbf{v}\,, \tag{14}$$

where $\Delta\mathbf{v}$ is the relative contact point velocity given by

$$\Delta \mathbf{v} = (\mathbf{v}_B + \omega_B \times (\mathbf{p} - \mathbf{x}_B)) - (\mathbf{v}_A + \omega_A \times (\mathbf{p} - \mathbf{x}_A))\ . \tag{15}$$

Here, $\mathbf{v}_A$ and $\mathbf{v}_B$ are the linear velocities of center of masses and ω_A and ω_B are the angular velocities. This notation is illustrated in left side of Figure 3.

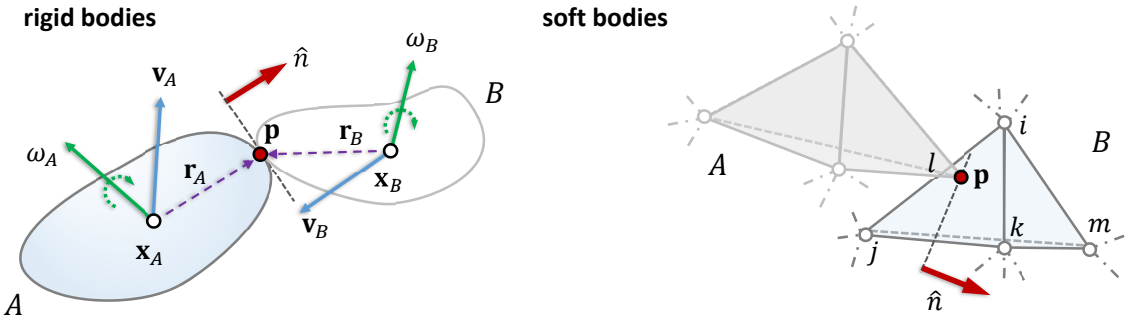


Fig. 3. Left: the notation involved in computing the relative contact point velocity of two rigid bodies are depicted. Right: The normal and vector arms and barycentric coordinates are the information needed to evaluate the contact Jacobian for two soft bodies modeled by tetrahedral elements.

Using the shorthand $\mathbf{r}_A = (\mathbf{p} - \mathbf{x}_A)$ and $\mathbf{r}_B = (\mathbf{p} - \mathbf{x}_B)$, the definition of the skew-symmetric cross product matrix of a vector $\mathbf{r} = \begin{bmatrix} x & y & z \end{bmatrix}^T$ is

$$\mathbf{r}^\times \equiv \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix}. \tag{16}$$

The relative velocity in the normal direction can then be written as the matrix-vector product

$$v_{\hat{n}} = \underbrace{\begin{bmatrix} -\hat{n}^T & \hat{n}^T\mathbf{r}_A^\times & \hat{n}^T & -\hat{n}^T\mathbf{r}_B^\times \end{bmatrix}}_{\mathbf{J}} \underbrace{\begin{bmatrix} \mathbf{v}_A \\ \omega_A \\ \mathbf{v}_B \\ \omega_B \end{bmatrix}}_{\mathbf{u}}. \tag{17}$$

This two-body system may be extended to include more body velocities in the system generalized velocity vector $\mathbf{u}$ and by adding corresponding zero-blocks to $\mathbf{J}$. Extending to multiple contact points means we get multiple rows in the Jacobian. Each one will have a structure similar to Equation 17, but with kinematics that are specific to each contact.

1.4.2 Non-interpenetration Constraint for Soft Bodies. The case of a soft body can be derived similar to the rigid body case. Often, a computational mesh is used to model the surface or volume of a soft body (e.g., a linear tetrahedral mesh). We will use this as our working example without loss of generality. Imagine a node of soft body A touches a face of a tetrahedron from soft body B. Let the position and velocity of the node from body A be given by $\mathbf{x}_{l,A}$ and $\mathbf{v}_{l,A}$ where l is the node index from body A. Let the four nodes defining the tetrahedron from body B be given by $\mathbf{x}_{i,B}$, $\mathbf{v}_{i,B}$, $\mathbf{x}_{j,B}$, $\mathbf{v}_{j,B}$, $\mathbf{x}_{k,B}$, $\mathbf{v}_{k,B}$, $\mathbf{x}_{m,B}$, and $\mathbf{v}_{m,B}$ where i, j, k, and m are the node

indices in body B. At the point of contact $\mathbf{p}$ we have

$$\mathbf{p} = \mathbf{x}_{l,A} = w_i \mathbf{x}_{i,B} + w_j \mathbf{x}_{j,B} + w_k \mathbf{x}_{k,B} + w_m \mathbf{x}_{m,B}\,, \tag{18}$$

where w_i, w_j, w_k, and w_m are the barycentric coordinates of $\mathbf{p}$ with respect to the tetrahedron. See the right side of Figure 3 for an illustration of the concepts. The relative contact point velocity in the contact normal direction is then given by

$$v_{\hat{n}} = \hat{n} \cdot \left(w_i \mathbf{v}_{i,B} + w_j \mathbf{v}_{j,B} + w_k \mathbf{v}_{k,B} + w_m \mathbf{v}_{m,B} - \mathbf{v}_{l,A} \right)\,, \tag{19}$$

and this can again be rewritten as a matrix-vector product, such that

$$v_{\hat{n}} = \underbrace{\begin{bmatrix} -\hat{n}^T & w_i \hat{n}^T & w_j \hat{n}^T & w_k \hat{n}^T & w_m \hat{n}^T \end{bmatrix}}_{\mathbf{J}} \underbrace{\begin{bmatrix} \mathbf{v}_{l,A} \\ \mathbf{v}_{i,B} \\ \mathbf{v}_{j,B} \\ \mathbf{v}_{k,B} \\ \mathbf{v}_{m,B} \end{bmatrix}}_{\mathbf{u}}\,. \tag{20}$$

Here, only the nodes involved in the contact are include in the Jacobian matrix $\mathbf{J}$ and the vector of node velocities $\mathbf{u}$. However, soft bodies usually contain many more nodes that need to be included into the system velocity vector $\mathbf{u}$, and thus corresponding zero blocks should be added to the Jacobian. As with the rigid body case, adding more contact points results in adding more rows of similar pattern to the system Jacobian. One may derive other specific formulas for the Jacobian in cases of contact between two nodes only, or two tetrahedra, or one node from soft body and one rigid body, and so on.

1.5 The Coulomb Friction Law

Friction is an important phenomena we must simulate in graphics applications. This introduces additional constraints on the movement of bodies in the simulations and the forces generated within the plane of contact.

In order to describe friction we will define a local contact frame for a single point of contact. This is called the contact frame and it consist of two orthogonal unit vectors that span the contact plane, let us call them $\hat{t}$ and $\hat{b}$, the contact plane can be viewed as the shared tangent plane between two smooth surfaces at a point of contact, $\mathbf{p}$. Orthogonal to the contact plane we have the contact normal direction. One can think of this as the direction where one at myopic scale must prevent motion to avoid penetration, we denote this direction by $\hat{n}$. There are two choices for which the direction of the normal could point in. If the two surfaces in contact are labelled A and B then we adopt the convection that the normal points from A towards B, and that $\hat{t}$, $\hat{b}$ and $\hat{n}$ forms a right handed coordinate system. This local coordinate system is used to describe the contact physics in. The contact frame is illustrated in Figure 4. Sometimes the frame is referred to as the contact basis, contact coordinate system or contact space. Observe

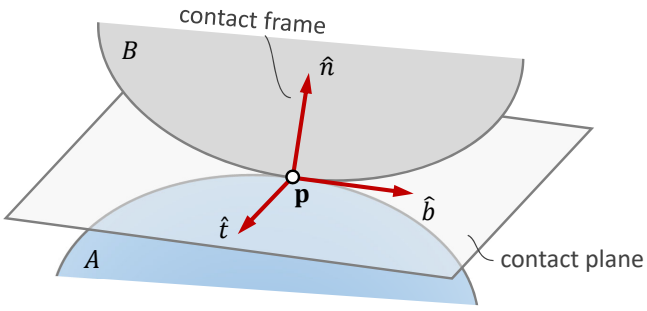


Fig. 4. The contact frame defines a right-handed coordinate system and is defined by the two tangent plane vectors $\hat{t}$ and $\hat{b}$ and the plane normal vector $\hat{n}$.

that the normal can be defined from the spatial derivative of the gap function we introduced in Section 1.4. There exist infinite many choices for generating the tangent vectors $\hat{t}$ and $\hat{b}$. When working with isotropic friction model it is from the model viewpoint not important how these vectors are generated but for anisotropic friction their choice is critical in aligning the friction cone properly between the two surfaces. We treat this in more detail in Section 3.4.

When describing friction forces in the contact space then we need to know the relative velocity of the surfaces in our contact frame. Let $\Delta\mathbf{v}$ be the relative velocity of the surfaces in world space. Then, the contact space version is:

$$\mathbf{v} = \underbrace{\begin{bmatrix}\hat{n} & \hat{t} & \hat{b}\end{bmatrix}^T}_{\mathbf{C}^T} \Delta\mathbf{v} \tag{21}$$

Notice that we picked the normal as the first basis vector in this transformation. This is just a convenient convention when we would like to have the normal part of the contact problem solved before the tangential components.

Realizing that the relative contact point velocity in world space between two rigid bodies is given by Equation 15, we can now write the contact space velocity as

$$\mathbf{v} = \underbrace{\mathbf{C}^T \begin{bmatrix}-\mathbf{I}_{3\times3} & \mathbf{r}_A^\times & \mathbf{I}_{3\times3} & -\mathbf{r}_B^\times\end{bmatrix}}_{\mathbf{J}} \underbrace{\begin{bmatrix}\mathbf{v}_A \\ \omega_A \\ \mathbf{v}_B \\ \omega_B\end{bmatrix}}_{\mathbf{u}}, \tag{22}$$

where $\mathbf{I}_{3\times3}$ is the 3-by-3 identity matrix. We see here that $\mathbf{J}$ has simply been extended with the tangent vectors compared to our previous non-frictional case. The $\mathbf{J}$ is called the contact Jacobian and from above equation we observe that is role is simply to map body-velocities from world-space into the contact space velocity. We have now explained how the Jacobian

from Equation 17 can be extended to include tangential velocities and the same extension can trivially be done for the soft body Jacobian in Equation 20. In the following we let $v_{\hat{n}}$ denote the normal component of the contact space velocity and $\mathbf{v}_{\hat{t}}$ the tangential component spanned by $\hat{t}$ and $\hat{b}$, similar notation is used for the contact impulse, where $\lambda_{\hat{n}} \in \mathbb{R}$ is the normal impulse and $\boldsymbol{\lambda}_{\hat{t}} \in \mathbb{R}^2$ is the tangential impulse (i.e., friction force between two surfaces). In this section we present a physical law that describes this force.

1.5.1 Theory of Friction. Coulomb friction couples the normal impulse $\lambda_{\hat{n}}$ and tangential impulses. The exact isotropic planar Coulomb friction cone constraint, for tangential friction force $\boldsymbol{\lambda}_{\hat{t}}$ applied to colliding bodies, can be written as

$$\|\boldsymbol{\lambda}_{\hat{t}}\|_2 \leq \mu\lambda_{\hat{n}} \,. \tag{23}$$

The non-linear inequality in Equation 23 defines a quadratic cone. The coefficient μ is called the coefficient of friction and is a unit-less non-negative value that relates friction force to the normal force. The coefficient of friction is specific in regards to the two types of materials that are in contact. When setting up simulations one often have to specify its value which can be bound measured data. We added a small table below with common values. The μ-coefficient is from a practical viewpoint considered to be a material-parameter only. We list some typical values for the coefficient of friction in Table 1

Table 1. Coefficients of friction for various pairs of materials from "The Engineering Toolbox" (https://www.engineeringtoolbox.com/). Observe that the coefficient of friction depends on environmental changes.

Materials and Material Combinations	Surface Conditions	Frictional Coefficient	
		Static	Kinetic (sliding)
Aluminum - Aluminum	Clean and Dry	1.05 - 1.35	1.4
Aluminum - Aluminum	Lubricated and Greasy	0.3	
Cast Iron - Cast Iron	Clean and Dry	1.1	0.15
Car tire - Asphalt	Clean and Dry	0.72	
Car tire - Grass	Clean and Dry	0.35	
Ice - Wood	Clean and Dry	0.05	
Leather - Oak	Parallel to grain	0.61	0.52
Rubber - Dry Asphalt	Clean and Dry	0.9	0.5 - 0.8
Steel - Steel	Clean and Dry	0.5 - 0.8	0.42
Steel - Steel	Lubricated and Greasy	0.16	
Steel - Steel	Castor oil	0.15	0.081

The Coulomb model is an empirical model, meaing that it has been derived from observations from measurements. The field of tribology is, among other things, concerned with explaining

the physical cause to the friction force. It turns out that friction is a quite complicated matter and many factors influence its behavior, and friction is essentially a system response. The causes of friction are explained from theory of asperities. At micro-scale level the asperities are sticking out of the material surfaces and when objects come into contact the asperities deform, break and plow through the surfaces thereby causing "resistance" to motion. It is this resistance that we humans perceive as the friction force. Besides the micro-scale geometry many other factors influence the behavior such as lubrication, electrostatic effects, third-party obstacles like small grains or dust, humidity and temperature, elastic and plastic micro-deformations of the surfaces, tear and wear and many more effects. The coefficient of friction can be understood as boiling all that complexity down to a single number. This is convenient from a modeling viewpoint as it makes it feasible for us to simulate many objects with complex shapes subject to frictional contact interaction. The Coulomb model is not perceived as being very accurate, but its simplicity makes it very appealing and thus is has wide-scale adoption for contact simulation. It is generally accepted that the coefficient of friction changes value when a transition from static friction (sticking) to dynamic friction (sliding) happens. It may be explained from bonding of asperities breaking when sliding happens and the dynamic coefficient of friction is therefore lower than that static one. The actual transition from sticking to sliding is called the onset of friction.

Let us return to setting up our model of friction that we use in multibody simulations for rigid and soft bodies in the field of computer graphics. The isotropic Coulomb friction law is in fact a two-part law:

- **Slip:** If the bodies are sliding relative to each other, then the direction of the friction force is opposed to the tangential relative velocity and its magnitude is $\mu\lambda_{\hat{n}}$.
- **Stick:** If the objects are not moving relative to each other, then the friction force can have any direction as long as the inequality in Equation 23 holds.

We can express both cases mathematically as follows

$$\mu \cdot \lambda_{\hat{n}} - \sqrt{\boldsymbol{\lambda}_{\hat{t}} \cdot \boldsymbol{\lambda}_{\hat{t}}} \geq 0\,, \tag{24a}$$

$$\|\mathbf{v}_{\hat{t}}\| \left(\mu\lambda_{\hat{n}} - \sqrt{\boldsymbol{\lambda}_{\hat{t}} \cdot \boldsymbol{\lambda}_{\hat{t}}}\right) = 0\,, \tag{24b}$$

$$\|\mathbf{v}_{\hat{t}}\|\|\boldsymbol{\lambda}_{\hat{t}}\| = -\mathbf{v}_{\hat{t}} \cdot \boldsymbol{\lambda}_{\hat{t}}\,. \tag{24c}$$

Observe here how the term $\|\mathbf{v}_{\hat{t}}\|$ act as a switch for selecting between the cases of slipping or sticking. If $\|\mathbf{v}_{\hat{t}}\| = 0$ then the last two conditions are trivially fulfilled and the first condition essentially only tells us that the friction force must belong to the friction cone. On the other hand, if $\|\mathbf{v}_{\hat{t}}\| > 0$ then the two last conditions kick in. The first one ensures the friction force is maximized, and the last one ensures that the friction force is opposing the sliding direction. This is illustrated in Figure 5. This form of the isotropic Coulomb friction model is frequently used for modeling frictional contact, and it represents a non-linear complementarity problem (NCP) formulation of frictional contact. However, in Section 1.6 and Section 1.7,

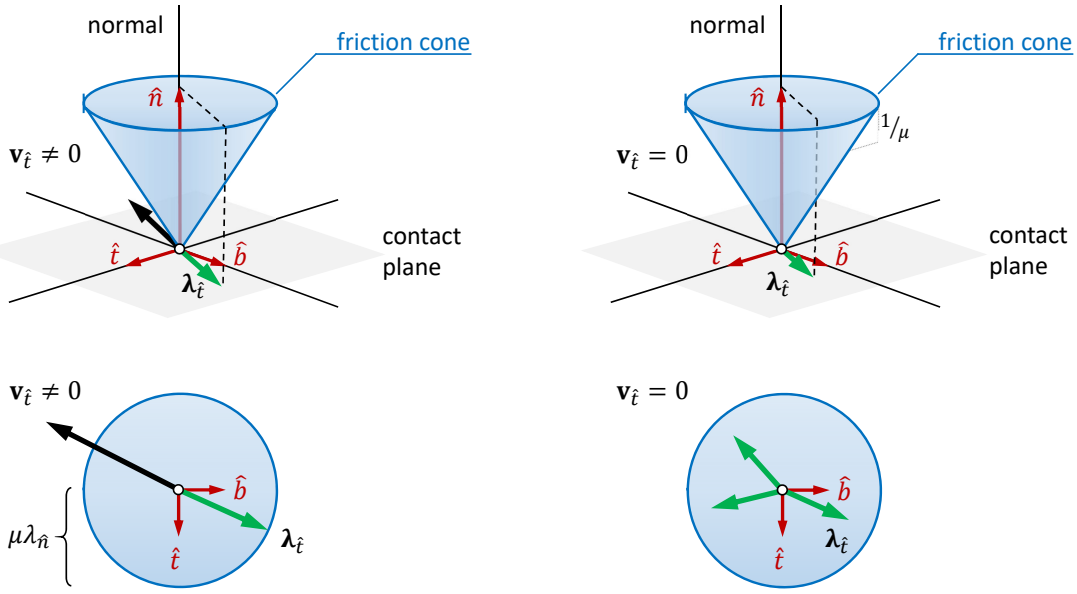


Fig. 5. Graphical illustration of the isotropic Coulomb law. Left: the case of slipping is depicted with a unique solution. Right: the case of sticking is shown to have multiple solutions. The bottom row shows the cone as seen from the top. Observe how the circular shape of the cone makes it particular easy to compute the friction force.

we present details on how to linearize the Coulomb friction model into the standard *linear complementarity problem* (LCP) and Boxed-LCP forms, which are the origin for the many LCP numerical methods we cover in Section 3.

The stick-slip conditions can be written up in a more general form as they are related to energy dissipation. If one assumes that the friction force maximally dissipates energy, then in the case of an isotropic circular friction cone this principle of maximum dissipation reduces to the stick-slip conditions we introduced above. For the more general force, we start by writing up the definition of the friction cone using a set notation:

$$\mathcal{F}(\mu\lambda_{\hat{n}}) \equiv \left\{ \gamma \in \mathbb{R}^2 \,\middle|\, \|\gamma\|_2 \leq \mu\lambda_{\hat{n}} \right\} . \tag{25}$$

This set denotes all feasible friction forces for the case of isotropic planar Coulomb friction. One can define the friction cone differently all dependent on the type of materials. For instance, an elliptical shaped cone can be used to model anisotropic friction. The principle of maximum dissipation can now be written as a minimization problem,

$$\boldsymbol{\lambda}_{\hat{t}} = \underset{\gamma \in \mathcal{F}(\mu\lambda_{\hat{n}})}{\arg\min}\, \mathbf{v}_{\hat{t}} \cdot \gamma \, . \tag{26}$$

That is to say the friction force is given by the force that instantaneously removes most (i.e. maximum dissipation) energy from the system.

If the friction cone $\mathcal{F}$ is a strict convex set then the minimization problem has an unique solution that are characterized by the first order necessary optimality conditions. If we momentarily denote the objective function as $f(\gamma) \equiv \mathbf{v}_{\hat{t}} \cdot \gamma$ and the convex set as $\Omega = \mathcal{F}(\mu\lambda_{\hat{n}})$ then these conditions can be written as

$$-\nabla_{\gamma} f(\lambda_t) \in \mathcal{N}_{\Omega}(\boldsymbol{\lambda}_{\hat{\mathbf{t}}}) \tag{27}$$

where $\mathcal{N}_{\Omega}(\boldsymbol{\lambda}_{\hat{\mathbf{t}}})$ is the normal cone of $\mathcal{F}(\mu\lambda_{\hat{n}})$ at the position $\boldsymbol{\lambda}_{\hat{t}}$. The normal cone of a convex set can be defined as follows,

$$\mathcal{N}_{\Omega}(\boldsymbol{\lambda}_{\hat{\mathbf{t}}}) \equiv \left\{ \mathbf{z} \,\middle|\, \mathbf{z} \cdot \left(\gamma - \boldsymbol{\lambda}_{\hat{\mathbf{t}}}\right) \leq 0,\ \forall \gamma \in \Omega \right\} . \tag{28}$$

If the set Ω is strictly convex, then we have $\mathcal{N}_{\Omega}(\boldsymbol{\lambda}_{\hat{t}}) \equiv -\mathbf{v}_{\hat{t}}$. If the set Ω is only convex, then the normal cone may be a multi-set. This first order optimality condition can be written concisely as the condition

$$\forall \gamma \in \mathcal{F}(\mu\lambda_{\hat{n}}) \quad \text{and} \quad (\gamma - \boldsymbol{\lambda}_{\hat{t}}) \cdot \mathbf{v}_{\hat{t}} \geq 0 \,. \tag{29}$$

This form is known as a variational inequality (VI) and is often the mathematical form of principle of maximum dissipation that is used as starting point for deriving nonlinear complementarity problem (NCP) formulations of the frictional contact problem. These NCP problems can be solved quite nicely with Newton type of methods. The VI-form given above can be used more directly in a numerical method without needing to do the linearizations we cover below in Sections 1.6 and 1.7. It offer one with a trade-of between a more "difficult" to solve nonlinear problem or an "easier" to solve linear version. The nonlinear form is more compact than the linear form in the sense that much fewer variables are needed. Due to its capability to inherently express the non-linearity the VI-form generalizes easily to an-isotropic friction and proximal operators allow for quite general shaped convex cones.

Above we have covered the most typical and classical models for planar dry friction. There exist other friction models. The above models can be extended trivially to include torsional (Coulomb–Contensou friction) and rolling friction by incorporating angular counter parts, we show this for torsional friction when we introduce proximal operators in Section 3.4. There are many more models as described by Sheng Chen and Liu [2016]. Like the Tresca friction which limits the friction force to a constant magnitude or the Stribek friction model which is dependent on the velocity magnitude to model effects of lubrication. Some models seek inspiration in micro-scale geometry interaction such as the bristle friction model. In Section 6 we present a new friction model that has originated in the graphics field that take a micro-scale modeling approach to present more interesting friction phenomena that go beyond the typical isotropic Coulomb friction model.

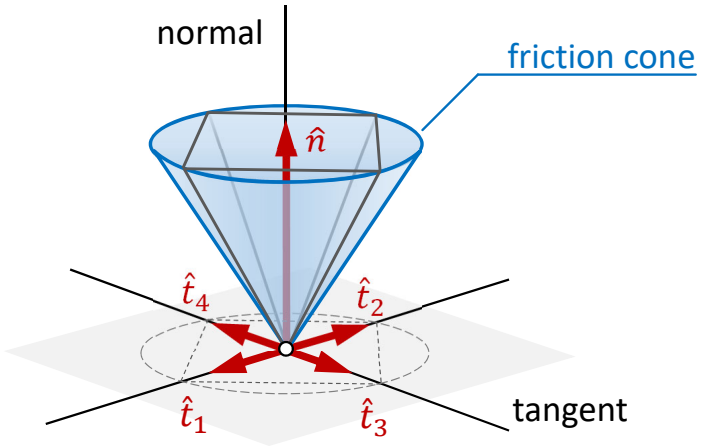


Fig. 6. A friction cone approximated by normal direction $\hat{n}$ and four friction directions $\{\hat{t}_1, \hat{t}_2, \hat{t}_3, \hat{t}_4\}$ that are tangent to the contact plane.

1.6 The Linear Complementarity Problem Model

A linear friction model is preferred for many applications due to their efficiency and compatibility with a wide variety of numerical solvers. Hence a linearized approximation of the friction cone introduced in the previous section is often used. Stewart and Trinkle, Anitescu and Potra, among others, linearize the 3D friction model using a polyhedral cone [Anitescu and Potra 1997; Stewart and Trinkle 1996]. The polyhedral cone is given by a span of $k+1$ unit vectors $\{\hat{n}, \hat{t}_1, \ldots, \hat{t}_k\}$, where $\hat{n}$ is the normal direction of the contact plane and $\{\hat{t}_1, \ldots, \hat{t}_k\}$ all lie in the contact plane (see Figure 6). We use a positive span for the tangent vectors $\{\hat{t}_1, \ldots, \hat{t}_k\}$, which means that each tangent vector has a twin vector that is oriented in the exact opposite direction. For instance, notice in Figure 6 that $\hat{t}_2$ is opposite $\hat{t}_1$. The advantage of using tangent vectors that point in opposite directions is that it allows the friction force to be written with all non-negative numbers in this basis. The disadvantage, from a computational viewpoint, is that one needs more numbers.

One advantage of the non-negative numbers is that they help us couple the sliding direction to the friction force direction. Let us just show a 1D example to make this modeling trick more clear. Assume we have some friction force measure c along the axis $\hat{t}$. We can express this measure instead using two numbers $a \geq 0$ and $b \geq 0$, such that

$$c\,\hat{t} = a\,\hat{t} - b\,\hat{t}\,. \tag{30}$$

Notice that c can be both a positive and negative number depending on the friction force direction and that c is now replaced by two non-negative numbers a and b. In the above

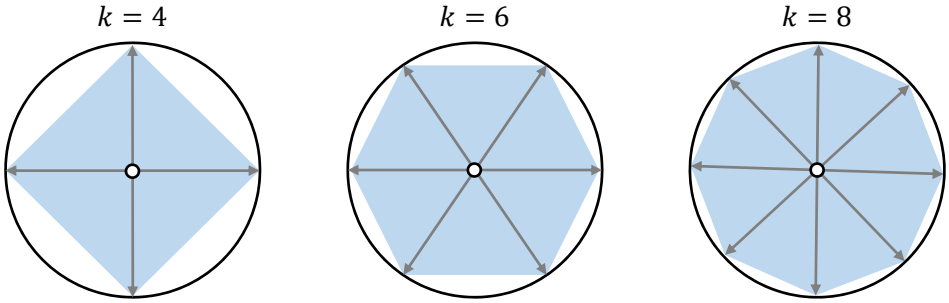


Fig. 7. The accuracy of the polyhedral cone approximation – light blue shading – increases as the number of tangential directions increases. The improvement in accuracy is at the cost of an increasing problem size.

example, the positive span is made of the two vectors $\{\hat{t}, -\hat{t}\}$. Observe also that we must have

$$0 \leq a \quad \perp \quad b \geq 0\,. \tag{31}$$

From this it is obvious that a is the positive part of c and b is the negative part of c, and as c can not be both positive and negative at the same time then a and b must be complementary.

The positive span approximation allows us to rewrite Equation 23 as

$$0 \leq \left(\mu\,\lambda_{\hat{n}} - \sum_i \lambda_{\hat{t}_i}\right), \tag{32}$$

where $\boldsymbol{\lambda}_{\hat{t}} = \begin{bmatrix}\lambda_{\hat{t}_1} & \cdots & \lambda_{\hat{t}_k}\end{bmatrix}^T$ is a vector of friction impulses along the tangent directions. In this way, the nonlinear Euclidean norm from the Coulomb model is replaced with the linear form in Equation 32. In principle, one can add as many tangent directions as wanted to get a better approximation to the exact cone as illustrated in Figure 7.

One challenge with this model is to correctly formulate the stick-slip transitions. We have introduced k possible directions for measuring a slipping velocity and we need just one scalar value– the slack variable– to determine the correct stick-slip behavior. The idea here is that we do not need to measure the exact slipping velocity. We only need information about whether there is slipping or not. Hence, one idea is to simply identify the direction of maximum slipping velocity. If we let the maximum slipping velocity along any direction denoted by $\beta \geq 0$ then we must have

$$0 \leq (\beta\,\mathbf{e} + \mathbf{v}_{\hat{t}}) \tag{33}$$

where $\mathbf{v}_{\hat{t}} = \begin{bmatrix}v_{\hat{t}_1} & \ldots & v_{\hat{t}_k}\end{bmatrix}$ is the sliding velocity measured along each tangent vector and where $\mathbf{e}$ is a k dimensional vector of ones. Observe here that β can only be zero if $\mathbf{v}_{\hat{t}}$ is zero. If β is positive, then it will have the same value of the maximum component of $\mathbf{v}_{\hat{t}}$. Hence, we say β is an estimate of the maximum sliding velocity along the directions of the positive span of tangent vectors.

We next observe that if there is slipping, then the friction force must have its maximum value in the opposite direction according to the principle of maximum dissipation. This translates into saying that the friction force should work such that it removes the maximum amount of energy from the system. To ensure this we will make sure that when $\beta > 0$ then $\left(\mu\,\lambda_{\hat{n}} - \sum_i \lambda_{\hat{t}_i}\right)$ becomes zero. This means the friction force will have its maximum allowed value when sliding. However, to get the direction of the friction correct we will force $(\beta\,\mathbf{e} + \mathbf{v}_{\hat{t}})$ to become zero when we have a friction component. This means when $\boldsymbol{\lambda}_{\hat{t}} > 0$ then $(\beta\,\mathbf{e} + \mathbf{v}_{\hat{t}}) = 0$.

Historically, the variable β for switching between the slip and stick regimes is called a "slack" variable. This name may appear non-intuitive at first, but consider it a measure of the amount of sliding that is present during the current state. It is these stick-slip transitions that make the frictional problem difficult, and hence why specialized numerical solvers are required to deal with complementarity conditions (as we will see in Section 3).

We may now summarize the four ingredients that went into stating a linear model for isotropic Coulomb friction:

- The non-penetration constraints that turn on and off whether we have sustained contact or separation.
- Linearization of the friction cone to remove the non-linear terms.
- Replace the measure of slipping velocity with the measure of maximum direction of slipping velocity, to give us a single switching variable to model stick-slip transitions.
- Using the principle of maximum dissipation to pick a unique friction direction when slipping occurs.

Finally, we can write up the complete mathematical model for our derivation. Letting $\boldsymbol{\lambda}_{\hat{t}} = \begin{bmatrix}\lambda_{\hat{t}_1} & \cdots & \lambda_{\hat{t}_k}\end{bmatrix}^T$ be a vector of friction impulses, the linearized model, including complementarity conditions, can be formally stated as

$$0 \leq v_{\hat{n}} \quad \perp \quad \lambda_n \geq 0, \tag{34a}$$

$$0 \leq (\beta\,\mathbf{e} + \mathbf{v}_{\hat{t}}) \quad \perp \quad \boldsymbol{\lambda}_{\hat{t}} \geq 0, \tag{34b}$$

$$0 \leq \left(\mu\lambda_{\hat{n}} - \sum_i \lambda_{\hat{t}_i}\right) \quad \perp \quad \beta \geq 0. \tag{34c}$$

The first complementarity constraint in Equation 34 models the non-penetration constraint as before. The second equation makes sure that in case we do have friction $\lambda_{\hat{t}_i} > 0$ for some i, then β will estimate the maximum sliding velocity along the $\hat{t}_i$'s directions. Observe that Equation 34b is a k-dimensional vector equation whose main purpose is to choose the direction $\hat{t}_i$ that is best for the direction of maximum dissipation. The last equation makes sure the friction force is bounded by the Coulomb friction cone. Notice that if $\beta > 0$, the last equation will force the friction force to lie on the boundary of the polyhedral friction cone. If $\beta = 0$, then the two last equations model static friction. That is, no sliding can occur and any friction force inside the friction cone is feasible.

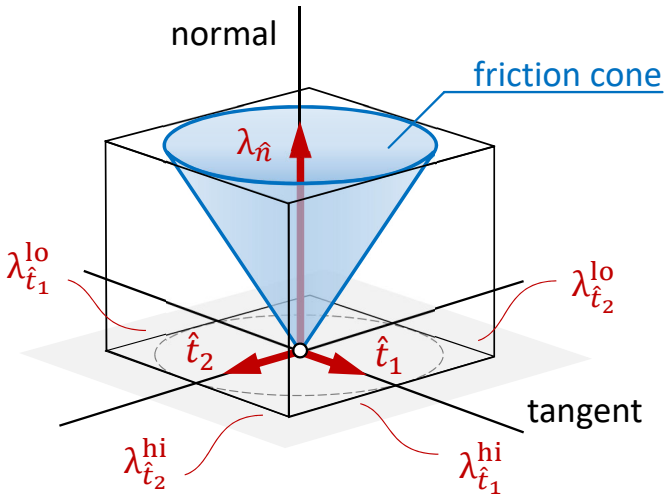


Fig. 8. The friction cone approximated by a box, where the limits of friction impulses in each tangential direction $\mathbf{t}_i$ are determined by the box limit equation $\lambda^{lo}_{t_i} \leq \lambda_{t_i} \leq \lambda^{hi}_{t_i}$.

For a general system consisting of p contacts, we can assemble the global Jacobian matrices containing the non-interpenetration and tangential frictional constraints for all contacts. The non-interpenetration Jacobian has a similar form to Equation 22, but with a single row per contact. For example, the ith contact with normal $\hat{n}_i$ is

$$\mathbf{J}_{\hat{n},i} = \begin{bmatrix} -\hat{n}_i^T & \hat{n}_i^T \mathbf{r}_{i,A}^{\times} & \hat{n}_i^T & -\hat{n}_i^T \mathbf{r}_{i,B}^{\times} \end{bmatrix}, \tag{35}$$

and for all p contacts we have $\mathbf{J}_{\hat{n}} = \begin{bmatrix} \mathbf{J}_{\hat{n},1}^T & \cdots & \mathbf{J}_{\hat{n},p}^T \end{bmatrix}^T$. Here the Jacobian is shown in its concise form, but in practice care must be taken so that the dimensions of the global matrix match the degrees of freedom of the dynamical system.

The frictional Jacobian is assembled in a similar way, but with k rows per contact that correspond to the friction basis in Figure 7, such that for contact i the matrix is

$$\mathbf{J}_{\hat{t},i} = \begin{bmatrix} -\hat{t}_{i,1}^T & \hat{t}_{i,1}^T \mathbf{r}_{i,A}^{\times} & \hat{t}_{i,1}^T & -\hat{t}_{i,1}^T \mathbf{r}_{i,B}^{\times} \\ \vdots & \vdots & \vdots & \vdots \\ -\hat{t}_{i,k}^T & \hat{t}_{i,k}^T \mathbf{r}_{i,A}^{\times} & \hat{t}_{i,k}^T & -\hat{t}_{i,k}^T \mathbf{r}_{i,B}^{\times} \end{bmatrix}, \tag{36}$$

and for all p contacts we have $\mathbf{J}_{\hat{t}} = \begin{bmatrix}\mathbf{J}_{\hat{t},1}^T & \cdots & \mathbf{J}_{\hat{t},p}^T\end{bmatrix}^T$. Finally, the constrained equations of motion for the system (i.e., in the form of Equation 8) may be written as

$$\begin{bmatrix}\mathbf{M} & -\mathbf{J}_{\hat{n}}^T & -\mathbf{J}_{\hat{t}}^T & 0 \\ \mathbf{J}_{\hat{n}} & 0 & 0 & 0 \\ \mathbf{J}_{\hat{t}} & 0 & 0 & \mathbf{E} \\ 0 & \bar{\boldsymbol{\mu}} & -\mathbf{E}^T & 0\end{bmatrix}\begin{bmatrix}\mathbf{u}^+ \\ \boldsymbol{\lambda}_{\hat{n}}^+ \\ \boldsymbol{\lambda}_{\hat{t}}^+ \\ \beta\end{bmatrix} + \begin{bmatrix}-\mathbf{M}\mathbf{u} - h\mathbf{f} \\ 0 \\ 0 \\ 0\end{bmatrix} = \begin{bmatrix}0 \\ \mathbf{s}_n \\ \mathbf{s}_\beta \\ \mathbf{s}_\lambda\end{bmatrix} \tag{37a}$$

$$0 \leq \mathbf{s}_n \perp \boldsymbol{\lambda}_{\hat{n}}^+ \geq 0 \tag{37b}$$

$$0 \leq \mathbf{s}_\beta \perp \boldsymbol{\lambda}_{\hat{t}}^+ \geq 0 \tag{37c}$$

$$0 \leq \mathbf{s}_\lambda \perp \beta \geq 0 \tag{37d}$$

where $\bar{\boldsymbol{\mu}} = \text{diag}\left(\begin{bmatrix}\mu_1 & \ldots & \mu_p\end{bmatrix}\right)$ is a diagonal matrix containing friction coefficients of each contact, and $\mathbf{E} = \text{diag}\left(\begin{bmatrix}\mathbf{e}_1^T & \ldots & \mathbf{e}_p^T\end{bmatrix}\right)$ is a block diagonal matrix containing k dimensional vectors of ones. The auxiliary variables $\mathbf{s}_{\hat{n}}$, $\mathbf{s}_\beta$ and $\mathbf{s}_\lambda$ are introduced here, which helps to write the complementarity problem in a more concise way. By applying the Schur complement "trick", we can compute the reduced system

$$\underbrace{\left(\mathbf{C} + \mathbf{G}\mathbf{M}^{-1}\mathbf{G}^T\right)}_{\mathbf{A}}\underbrace{\begin{bmatrix}\boldsymbol{\lambda}_{\hat{n}}^+ \\ \boldsymbol{\lambda}_{\hat{t}}^+ \\ \beta\end{bmatrix}}_{\mathbf{x}} + \underbrace{\mathbf{G}\mathbf{M}^{-1}\left(\mathbf{M}\mathbf{u} + h\mathbf{f}\right)}_{\mathbf{b}} = \begin{bmatrix}\mathbf{s}_{\hat{n}} \\ \mathbf{s}_\beta \\ \mathbf{s}_\lambda\end{bmatrix}, \tag{38}$$

where

$$\mathbf{C} = \begin{bmatrix}0 & 0 & 0 \\ 0 & 0 & \mathbf{E} \\ \bar{\boldsymbol{\mu}} & -\mathbf{E}^T & 0\end{bmatrix}, \text{ and } \mathbf{G}^T = \begin{bmatrix}\mathbf{J}_{\hat{t}}^T & \mathbf{J}_{\hat{n}}^T & 0\end{bmatrix}.$$

Then, Equation 37a can be written concisely and compactly in the form of a standard LCP:

$$\mathbf{0} \leq \mathbf{A}\mathbf{x} + \mathbf{b} \quad \perp \quad \mathbf{x} \geq \mathbf{0} \tag{39}$$

This illustrates how powerful the LCP model really is. By simply assembling the $\mathbf{A}$ matrix and $\mathbf{b}$ vector, we can call our favorite LCP solver to compute $\mathbf{x}$. The system may be slightly reduced by noting that the first row in the system can be used to eliminate $\mathbf{u}^+$. This rewrite is similar to how Equation 8 was transformed in Equation 9 using the Schur complement. Another important observation here is that $\mathbf{A}$ is asymmetric for the LCP model. This has consequences with regards to the solvers than can be applied to this model. However, the boxed model we derive in the next section breaks the coupling of the tangent directions and this leads to a matrix $\mathbf{A}$ that is symmetric. .

1.7 The Boxed Linear Complementarity Problem Model

An alternative for discretizing the friction cone can be seen in Figure 8. Using two orthogonal tangent directions, one can write independent inequalities in each friction direction to have the sliding force limited by the coefficient of friction, μ, times the normal force, $\lambda_{\hat{n}}$. That is,

$$\lambda_{\hat{t}_1}^{\text{lo}} \leq \lambda_{\hat{t}_i} \leq \lambda_{\hat{t}_1}^{\text{hi}} \tag{40a}$$

$$\lambda_{\hat{t}_2}^{\text{lo}} \leq \lambda_{\hat{t}_2} \leq \lambda_{\hat{t}_2}^{\text{hi}} \tag{40b}$$

where $\lambda_{\hat{t}_i}^{lo} = -\mu\lambda_{\hat{n}}$ and $\lambda_{\hat{t}_i}^{hi} = \mu\lambda_{\hat{n}}$. Some iterative solvers update these friction bounds during the solve based on the magnitude of the normal forces, yielding a four-sided pyramidal cone that is larger than the true friction cone (in contrast to the approximation in Figure 6 which is inside the true cone). A further approximation is made by other solvers: using a preliminary solve of the normal forces, the bounds can be set and the system can be solved with box friction limits.

The bounds on friction impulses in Equations 40a-40b requires us to develop a numerical method for a problem class which is slightly more general than the classic LCP. Problems in this more general class are called *boxed linear complementarity problems* (BLCP).

One appealing notational benefit of writing the contact and friction models as a BLCP is that both the non-penetration constraints and friction bounds can be expressed with the same algebraic notation simply by changing how the lower and upper bounds are defined. For instance, normal impulses have lower and upper bounds $\lambda_{\hat{n}}^{\text{lo}} = 0$ and $\lambda_{\hat{n}}^{\text{hi}} = \infty$. We demonstrate this by allowing the index i to refer to a friction variable or a normal impulse, and then make appropriate changes to the bounds. Given $\lambda_i, v_i, \lambda_i^{\text{lo}}, \lambda_i^{\text{hi}} \in \mathbb{R}$, the following three conditions must be satisfied:

$$v_i > 0 \Rightarrow \lambda_i = \lambda_i^{\text{lo}} \tag{41a}$$

$$v_i < 0 \Rightarrow \lambda_i = \lambda_i^{\text{hi}} \tag{41b}$$

$$v_i = 0 \Rightarrow \lambda_i^{\text{lo}} \leq \lambda_i \leq \lambda_i^{\text{hi}} \tag{41c}$$

The variables λ_i^{lo} and λ_i^{hi} are the lower and upper bounds on λ_i, and usually it is assumed that $\lambda_i^{\text{lo}} < \lambda_i^{\text{hi}}$. We note that by decomposing the residual velocity as $v_i = {}^{+}v_{\_}{}^{-}v_i$, the BLCP can be written as three separate LCPs:

$$0 \leq {}^{+}v_i \perp (\lambda_i - \lambda_i^{\text{lo}}) \geq 0 \tag{42a}$$

$$0 \leq {}^{-}v_i \perp (\lambda_i^{\text{hi}} - \lambda_i) \geq 0 \tag{42b}$$

$$0 \leq {}^{-}v_i \perp {}^{+}v_i \geq 0 \tag{42c}$$

Similar to the polyhedral cone version of the LCP from Section 1.6, a linear system for p contacts can be assembled. We begin with the Jacobian matrix for a single contact i, which is

given by

$$\mathbf{J}_i = \begin{bmatrix} -\hat{n}_i^T & \hat{n}_i^T \mathbf{r}_{i,A}^\times & \hat{n}_i^T & -\hat{n}_i^T \mathbf{r}_{i,B}^\times \\ -\hat{t}_{i,1}^T & \hat{t}_{i,1}^T \mathbf{r}_{i,A}^\times & \hat{t}_{i,1}^T & -\hat{t}_{i,1}^T \mathbf{r}_{i,B}^\times \\ -\hat{t}_{i,2}^T & \hat{t}_{i,2}^T \mathbf{r}_{i,A}^\times & \hat{t}_{i,2}^T & -\hat{t}_{i,2}^T \mathbf{r}_{i,B}^\times \end{bmatrix}. \tag{43}$$

This is in fact the same Jacobian matrix from Equation 22, where $\hat{t}_1 = \hat{t}$ and $\hat{t}_2 = \hat{b}$. The global Jacobian matrix can then be written as $\mathbf{J} = \begin{bmatrix} \mathbf{J}_1^T & \dots & \mathbf{J}_p^T \in \mathbb{R}^{3p \times n} \end{bmatrix}^T$. Finally, we write global BLCP of the constrained equations of motion as:

$$\underbrace{\begin{bmatrix} \mathbf{M} & -\mathbf{J}^T \\ \mathbf{J} & 0 \end{bmatrix}}_{\mathbf{A}} \begin{bmatrix} \mathbf{u}^+ \\ \boldsymbol{\lambda}^+ \end{bmatrix} + \underbrace{\begin{bmatrix} -\mathbf{M}\mathbf{u} - h\mathbf{f} \\ 0 \end{bmatrix}}_{\mathbf{b}} = \begin{bmatrix} 0 \\ \mathbf{v} \end{bmatrix}, \tag{44a}$$

$$0 \leq {}^+\mathbf{v} \perp \boldsymbol{\lambda}^+ - \boldsymbol{\lambda}^{\text{lo}} \geq 0, \tag{44b}$$

$$0 \leq {}^-\mathbf{v} \perp \boldsymbol{\lambda}^{\text{hi}} - \boldsymbol{\lambda}^+ \geq 0, \tag{44c}$$

$$0 \leq {}^-\mathbf{v} \perp {}^+\mathbf{v} \geq 0. \tag{44d}$$

Note that in the above formulation, the constraint impulses are ordered according to the rows of $\mathbf{J}$, such that $\boldsymbol{\lambda}^+ = \begin{bmatrix} \lambda_{\hat{n}_1} & \lambda_{\hat{t}_{1,1}} & \lambda_{\hat{t}_{1,2}} & \dots & \lambda_{\hat{n}_p} & \lambda_{\hat{t}_{p,1}} & \lambda_{\hat{t}_{p,2}} \end{bmatrix}^T$. The Schur complement may be used to eliminate the variable $\mathbf{u}^+$ from this system, similar to how Equation 8 was transformed in Equation 9, and we need only solve for $\boldsymbol{\lambda}^+$. From top row we obtain $\mathbf{u}^+ = \mathbf{M}^{-1}\mathbf{J}^T\boldsymbol{\lambda}^+ + \mathbf{u} + h\mathbf{M}^{-1}\mathbf{f}$ and substitution into the bottom row yields

$$\underbrace{\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T}_{\mathbf{A}} \boldsymbol{\lambda}^+ + \underbrace{\left(-\mathbf{J}\mathbf{u} - h\mathbf{J}\mathbf{M}^{-1}\mathbf{f}\right)}_{\mathbf{b}} = \mathbf{v}, \tag{45a}$$

$$0 \leq {}^+\mathbf{v} \perp \boldsymbol{\lambda}^+ - \boldsymbol{\lambda}^{\text{lo}} \geq 0, \tag{45b}$$

$$0 \leq {}^-\mathbf{v} \perp \boldsymbol{\lambda}^{\text{hi}} - \boldsymbol{\lambda}^+ \geq 0, \tag{45c}$$

$$0 \leq {}^-\mathbf{v} \perp {}^+\mathbf{v} \geq 0. \tag{45d}$$

1.8 The Cone Complementarity Problem Model

The Coulomb friction model we introduced in Section 1.5 combined with the non-penetration constraint from Section 1.4 is essentially a nonlinear combinatorial problem. It is by its very nature really not an optimization problem. In this section we will introduce a relaxation technique that changes the original model but in such a way that we obtain a convex minimization problem. We will start our rewrite of the contact force model by changing the normal non-penetration constraint to the form [Anitescu and Hart 2004]

$$0 \leq \tfrac{1}{h}\phi + v_{\hat{n}} - \mu\sqrt{v_{\hat{t}}^2 + v_{\hat{b}}^2} \quad \perp \quad \lambda_n \geq 0. \tag{46}$$

Here $\mathbf{v} = \{v_{\hat{n}}, v_{\hat{t}}, v_{\hat{b}}\}$ is the contact space velocity. Usually we have $\phi = 0$ for touching contacts and then the equation simplifies to

$$0 \leq v_{\hat{n}} - \mu\sqrt{v_{\hat{t}}^2 + v_{\hat{b}}^2} \quad \perp \quad \lambda_{\hat{n}} \geq 0\,. \tag{47}$$

This is essentially the only change we are making, everything else we keep the same. What remains is a bit of mathematical convenience in rewriting everything into a simple form. We will now combine this relaxed normal non-penetration constraint with the isotropic planar Coulomb model from previously. This time we will write up both the normal and friction parts simultaneously into one single equation. To do so we need to recall a few definitions from math. The dual cone to a convex cone $\mathcal{K}$ is defined as

$$\mathcal{K}^\star \equiv \{\mathbf{y} \,|\, \mathbf{y} \cdot \mathbf{x} \geq 0, \forall \mathbf{x} \in \mathcal{K}\}\ . \tag{48}$$

The polar cone to a convex cone $\mathcal{K}$ is defined as

$$\mathcal{K}^\circ \equiv \{\mathbf{y} \,|\, \mathbf{y} \cdot \mathbf{x} \leq 0, \forall \mathbf{x} \in \mathcal{K}\}\ . \tag{49}$$

Observe we have $\mathcal{K}^\circ = -\mathcal{K}^\star$. We now have enough cone-formalism to proceed with putting our model together. Recall that the isotropic Coulomb friction law stated that $\mu\lambda_{\hat{n}} \geq \left\|\boldsymbol{\lambda}_{\hat{\mathbf{t}}}\right\|$, where $\boldsymbol{\lambda}_{\hat{t}} = (\lambda_{\hat{t}}, \lambda_{\hat{b}})^T$. This we can rewrite into $\mu^2\lambda_{\hat{n}} - \lambda_{\hat{t}}^2 - \lambda_{\hat{b}}^2 \geq 0$. Now let us define the convex friction cone as follows,

$$\mathcal{F}_\mu \equiv \left\{\boldsymbol{\lambda} = (\lambda_{\hat{n}}, \lambda_{\hat{t}}, \lambda_{\hat{b}})^T \in \mathbb{R}^3 \,\middle|\, \sqrt{\lambda_{\hat{t}}^2 + \lambda_{\hat{b}}^2} \leq \mu\lambda_{\hat{n}}\right\}\ . \tag{50}$$

This is a little different from the previous meaning of $\mathcal{F}(\mu\lambda_{\hat{n}})$ where $\lambda_{\hat{n}}$ was seen as an input parameter that generates a specific cone. The new notation allow us to write $\lambda \in \mathcal{F}_\mu$ to express the bound of the Coulomb friction law. We now write the combined normal and friction constraints simply as a *cone complementarity problem* (CCP),

$$\mathcal{F}_\mu^\star \ni \mathbf{v} \perp \lambda \in \mathcal{F}_\mu\,, \tag{51}$$

or equivalently,

$$\mathcal{F}_\mu^\circ \ni -\mathbf{v} \perp \lambda \in \mathcal{F}_\mu\,. \tag{52}$$

The cone complementary notation means that

$$-\mathbf{v} \in \mathcal{F}_\mu^\circ\,,\ \boldsymbol{\lambda} \in \mathcal{F}_\mu\,, \quad \text{and} \quad -\mathbf{v} \cdot \boldsymbol{\lambda} = 0\,. \tag{53}$$

For the isotropic Coulomb friction cone we see that the dual cone is defined as

$$\mathcal{F}_\mu^\circ \equiv \left\{\mathbf{v} = \{v_{\hat{n}}, v_{\hat{t}}, v_{\hat{b}}\} \in \mathbb{R}^3 \,\middle|\, \sqrt{v_{\hat{t}}^2 + v_{\hat{b}}^2} \leq \tfrac{1}{\mu} v_{\hat{n}}\right\}\ . \tag{54}$$

Figure 9 illustrates the difference between the nonlinear complementarity problem (NCP) formulation of the friction contact we have presented previously and the new *cone complementarity problem* (CCP) model we just presented.

As the figure illustrates the CCP model can be written quite compact and that the sliding velocity $\mathbf{v}$ has been relaxed. That is we have given it more "freedom" not to be confined to the

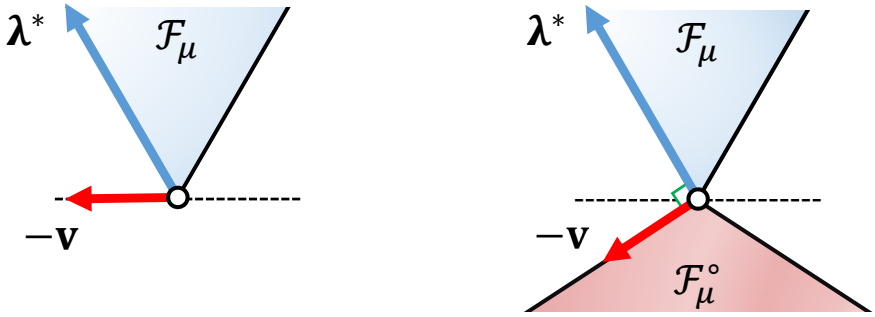


Fig. 9. On the left the solution from the classical complementary problem formulation is shown. Here the sliding velocity is confined to be in the 2D contact plane. On the right the cone-complementarity formulation (CCP) is shown. Observe that the sliding velocity is now orthogonal to the contact force.

2D contact plane. Furthermore, the polar cone keeps the velocity orthogonal to the contact force. Combined those traits give this model many numerical benefits in terms of having very fast iterative solvers with sub-linear convergence rates. Projected gradients methods was initially explored and are similar in spirit to the project Gauss-Seidel methods we cover in this work [Anitescu and Tasora 2010]. The accelerated projected gradient descent method (APGD) is another such method with convergence rate $O\left(\frac{1}{k^2}\right)$ where k is the iteration number [Mazhar et al. 2015]. In comparison to the proximal operator methods we present in Section 3.4 for the NCP type of model they have $O\left(\frac{1}{k}\right)$. A modified Fischer-Burmeister function can be used to rewrite the CCP model into a root finding problem and by applying a splitting strategy one may develop Gauss-Seidel type solvers from this setting too [Daviet et al. 2011].

Hence, there are obvious performance benefits from this formulation of the contact forces. The catch is that the "physics" is different from the classical model. As Figure 9 clearly shows, as the tangential sliding velocity grows large the model will gain a positive normal component based on the relative contact velocity, $v_{\hat{n}} > 0$. In other words, if one has a stack of blocks and pushes a block very fast, then it will try to lift of from the other blocks. The effect can be limited with adding an adhesion term to the model, but it does not completely remove the effect and adhesion will also further change the physical behavior.

Whether the physics of the NCP versus the CCP model is right or wrong is debatable as friction is a system response and surface material interactions are therefore quite a complicated matter to model and there is huge variation. The CCP are disliked by some as it feels like a mathematical rewrite to the physics to get nice numerical properties, and it is loved by others because the numerical traits the CCP model gains results in very fast solvers. Work has been done on validating the CCP model in context of granular flows and here it shows good agreement with reality [Mazhar et al. 2015].

Recalling the Schur complement reduction technique we used to derive the reduced form in Equation 9, then from

$$\mathbf{v} = \mathbf{J}\mathbf{u}^{+}\,, \tag{55a}$$

$$\mathbf{M}\mathbf{u}^{+} = \mathbf{J}^{T}\boldsymbol{\lambda} + \mathbf{M}\mathbf{u} + h\mathbf{f} \tag{55b}$$

one can derive the linear relationship between sliding velocity and contact forces as

$$\mathbf{v} = \mathbf{A}\boldsymbol{\lambda} + \mathbf{b}\,. \tag{56}$$

Substituting this equation into the CCP model, we observe that the model is equivalent to the first order optimality conditions of a second order cone problem (SOCP), which is given by

$$\boldsymbol{\lambda}^{*} \equiv \underset{\boldsymbol{\lambda}\in\mathcal{F}_{\mu}}{\arg\min}\, \tfrac{1}{2}\boldsymbol{\lambda}^{T}\mathbf{A}\boldsymbol{\lambda} + \boldsymbol{\lambda}^{T}\mathbf{b}\,. \tag{57}$$

Notice here that both the objective function and the constraints are written as convex second order cones. This insight opens up for the application of a vast majority of numerical methods for solving the CCP model, such as alternating direction method of multipliers (ADMM) [Tasora et al. 2021]. One may alternatively have used the Schur complement to reduce to a quadratic objective function in $\mathbf{v}$ instead of $\boldsymbol{\lambda}$ as was done in [Acary et al. 2011]. The ADMM method may be used on this alternative velocity form of the CCP model too [Daviet 2020].

We do not cover solvers for CCP frictional contact models in these notes. However, with the material presented later on iterative methods for NCPs, LCPs, BLCPs and proximal operators, the reader should be off to a good start to implement numerical methods for CCP type models too.

1.9 Constraint Stabilization

The system in Equation 8, and its reduced form in Equation 9, are linear approximations of a non-linear dynamical systems. Therefore, they are prone to numerical drift, especially when combined with the low-order numerical integration methods that are commonplace in computer graphics applications. Also, an exact numerical solution of the linear system is practically impossible, which again contributes to the drift. In the context of contact simulation, this results in both position and velocity level artifacts. At the position level, the interpenetration between bodies will increase since non-interpenetration constraints cannot be exactly resolved, whereas at the velocity level, bodies will begin to slide rather than stick. Furthermore, if discrete collision detection is used to generate contact constraints, this means that collision events may not be detected at the time of impact and hence we cannot assume that bodies will be in a non-interpenetrating state at the start of a time step.

The issues described above will cause a multibody system to gradually violate the constraint manifold. Recall that the manifold is an invariant set defined by the gap function, where $\phi(\mathbf{q}) \geq 0$. An approach to solve this problem would be to use a feedback rule to bring the system back to a valid state, where the constraints are not violated. Giving some physical

meaning to this idea, a spring in the constraint space may be used to "push" or "pull" the multibody system back to the constraint manifold. If there is no constraint violation, then this constraint spring generates no forces. However, if there is a constraint violation (e.g., $\phi(\mathbf{q}) < 0$), then the spring generates a force to restore the configuration back to the constraint manifold. Essentially, the spring force stabilizes the constraint.

To realize this behavior, let us assume that all constraint forces are generated by an implicit Hookean spring, such that

$$\lambda^+ = -k\phi^+ - bv^+ . \tag{58}$$

Here, ϕ is the gap function for a single constraint, v is the relative velocity in constraint space, and k and b are the stiffness and damping coefficients of the spring. Note that we again adopt the convention that $\square^+$ is an implicit variable.

Equation 58 is in fact an example of the well known Baumgarte stabilization [Baumgarte 1972] technique. Approximating the constraint error term by $\phi^+ = \phi + hv^+$, the spring equation can be rewritten as

$$\lambda^+ + (hk + b)\, v^+ = -k\phi,$$

which is further simplified by dividing both sides by $(hk + b)$, such that

$$v^+ + \frac{1}{hk + b}\lambda^+ = -\frac{k}{hk + b}\phi.$$

Finally, recall that $v^+ = \mathbf{J}\mathbf{u}^+$, and further simplification of the above equation yields:

$$\mathbf{J}\mathbf{u}^+ + \underbrace{\left(\frac{1}{h^2k + hb}\right)}_{\epsilon}\lambda^+ = \underbrace{\left(\frac{hk}{hk + b}\right)}_{\upsilon}\frac{-\phi}{h} . \tag{59}$$

Note that Equation 59 resembles Equation 4, but introduces a feedback term on the right-hand side of the equation. This term attempts to reduce the constraint violation ϕ by applying a constraint-space spring impulse in order to resolve the error by end of the time step. However, the portion of the constraint error being reduced at each step is modulated according to υ, which is commonly known as the *error reduction parameter* (ERP). Observe that setting $\upsilon = 1$ will encourage the feedback rule to produce a constraint velocity that reduces all of the constraint error in a single step. Also, observe that a portion of the implicit constraint force λ^+ is now being mixed with the kinematic constraint in Equation 59, and it is often referred to as the *constraint force mixing* (CFM) term.

The ϵ and υ parameters may be specified directly, which is intuitive from a numerical standpoint. Whereas adjusting k and b and then generating ϵ and υ from the spring coefficients is perhaps a more physically intuitive approach, since there is some notion that these behave like material properties of the contact. Additionally, parameters k and b may be tuned independent of the time step h.

The stabilization parameters can be tuned differently for each of the m constraints and assembled into the diagonal matrices:

$$\Sigma = \begin{bmatrix} \epsilon_1 & & & \\ & \epsilon_2 & & \\ & & \ddots & \\ & & & \epsilon_m \end{bmatrix}, \Upsilon = \begin{bmatrix} \upsilon_1 & & & \\ & \upsilon_2 & & \\ & & \ddots & \\ & & & \upsilon_m \end{bmatrix}.$$

Reassembling the linear system from Equation 8 and accounting for the new stabilization terms gives a multibody system with constraint stabilization:

$$\begin{bmatrix} \mathbf{M} & -\mathbf{J}^T \\ \mathbf{J} & \Sigma \end{bmatrix} \begin{bmatrix} \mathbf{u}^+ \\ \boldsymbol{\lambda}^+ \end{bmatrix} = \begin{bmatrix} \mathbf{M}\,\mathbf{u} + h\,\mathbf{f} \\ -\Upsilon \frac{\boldsymbol{\phi}}{h} \end{bmatrix}. \tag{60}$$

The system in Equation 60 includes not only a feedback term to reduce the positional errors for each constraint, which are stored in $\boldsymbol{\phi}^+ \in \mathbb{R}^m$, but also the diagonal matrix Σ in the lower right block. This has the added benefit of improving the conditioning of the matrix, which is especially beneficial during simulations involving complex contact where the linear system would otherwise become degenerate due to redundant rows in the Jacobian matrix. It can also be shown that the linear system in Equation 60 is positive definite, assuming that $\epsilon > 0$ for all constraints.

1.10 Soft vs. Rigid Body

Many of the methods we cover for solving the frictional contact problem can be extended from rigid bodies to soft bodies. For this introductory text on solvers we have taken the typical approach and presented the ideas using rigid body notation. Below we will first describe how equation and matrices of a full rigid body system look and afterwards we will describe the changes for a soft body system. The most important differences is that for soft bodies one typically use more implicit time discretizations and the number of variables are much larger.

1.10.1 Assembling the Matrices for a Rigid Body System. We will here for completeness explain how to construct the matrices $\mathbf{M}$ and $\mathbf{J}$ and the vector $\mathbf{f}$ that holds external and gyroscopic force terms when considering a rigid body system with multiple rigid bodies and multiple contacts. This section can be skipped if reader is already familiar with those aspects.

Let us consider the k^{th} contact point. The contact normal is given by $\hat{n}_k$ and the two orthonormal vectors spanning the contact plane are $\hat{t}_k$ and $\hat{b}_k$. The indices of the two bodies meeting at the contact is i and j, where we assume that $i < j$. We will adopt the convention that $\hat{n}_k$ is pointing from body i to body j. The vector arms from the center of mass of the bodies to the point of contact is given by $\mathbf{r}_{k_i}$ and $\mathbf{r}_{k_j}$ respectively. The relative contact point

velocity can now be written as follows

$$\begin{bmatrix} v_{\hat{n}_k} \\ v_{\hat{t}_k} \\ v_{\hat{b}_k} \\ v_{\tau_k} \end{bmatrix} = \begin{bmatrix} -\mathbf{C}_k^T & -\left(\mathbf{r}_{\boldsymbol{k_i}}{}^{\times}\mathbf{C}_k\right)^T & \mathbf{C}_k^T & \left(\mathbf{r}_{\boldsymbol{k_j}}{}^{\times}\mathbf{C}_k\right)^T \\ \mathbf{0}^T & -\hat{n}_k^T & \mathbf{0}^T & \hat{n}_k^T \end{bmatrix} \begin{bmatrix} \mathbf{v}_i \\ \omega_i \\ \mathbf{v}_j \\ \omega_j \end{bmatrix} \tag{61}$$

where

$$\mathbf{C}_k = \begin{bmatrix} \hat{n}_k & \hat{t}_k & \hat{b}_k \end{bmatrix} . \tag{62}$$

In the equations above we included the angular spin around the normal axis, v_{τ_k}, to demonstrate how this term can be included. If one only cares about planar friction then the bottom row of our equation can be dropped. Observe we can get the rolling spin around the tangent vectors included by replacing the $\hat{n}_k^T$ with $\mathbf{C}_k^T$ in the last row. Doing this the left-hand side contact velocity vector would gain two more components being the spin-velocities around $\hat{t}_k$ and $\hat{b}_k$.

The tangent plane directions $\hat{t}_k$ and $\hat{b}_k$ are often computed by picking $\hat{t}_k$ to be in the direction of sliding and letting $\hat{b}_k$ be orthogonal to $\hat{t}_k$. For isotropic friction modeling it does not matter much how the vectors are computed. In Section 6 we discuss other approaches for computing these vectors in the case of anisotropic friction.

Defining the block notation

$$\mathbf{v}_k = \begin{bmatrix} v_{\hat{n}_k} \\ v_{\hat{t}_k} \\ v_{\hat{b}_k} \\ v_{\tau_k} \end{bmatrix}, \mathbf{u}_i = \begin{bmatrix} \mathbf{v}_i \\ \omega_i \end{bmatrix}, \mathbf{u}_j = \begin{bmatrix} \mathbf{v}_j \\ \omega_j \end{bmatrix}, \tag{63}$$

and

$$\mathbf{J}_{k,i} = \begin{bmatrix} -\mathbf{C}_k^T & -\left(\mathbf{r}_{\boldsymbol{k_i}}{}^{\times}\mathbf{C}_k\right)^T \\ \mathbf{0}^T & -\hat{n}_k^T \end{bmatrix}, \mathbf{J}_{k,j} = \begin{bmatrix} \mathbf{C}_k^T & \left(\mathbf{r}_{\boldsymbol{k_j}}{}^{\times}\mathbf{C}_k\right)^T \\ \mathbf{0}^T & \hat{n}_k^T \end{bmatrix} \tag{64}$$

and assembling a full system of all K contacts and N bodies we have

$$\underbrace{\begin{bmatrix} \mathbf{v}_1 \\ \vdots \\ \mathbf{v}_k \\ \vdots \\ \mathbf{v}_K \end{bmatrix}}_{\mathbf{v}} = \underbrace{\begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \vdots \\ 0\cdots 0 & \mathbf{J}_{k,i} & 0\cdots 0 & \mathbf{J}_{k,j} & 0\cdots 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}}_{\mathbf{J}} \underbrace{\begin{bmatrix} \mathbf{u}_1 \\ \vdots \\ \mathbf{u}_i \\ \vdots \\ \mathbf{u}_j \\ \vdots \\ \mathbf{u}_N \end{bmatrix}}_{\mathbf{u}} . \tag{65}$$

The i^{th} rigid body has the center of mass position $\mathbf{x}_i$ and the orientation given by the unit quaternion $Q_i \equiv (\phi_i, \psi_i, \theta_i, \xi_i)$ together with the mass m_i and local body-frame inertia tensor ${}^b\mathbf{I}_i$. The local body frame inertia tensor must be updated to reflect the world-frame inertia tensor

$$ {}^w\mathbf{I}_i = \mathbf{R}_i^b \mathbf{I}_i \mathbf{R}_i^T \tag{66} $$

where $\mathbf{R}_i$ is the rotation matrix corresponding to Q_i. Defining the mass-block notation

$$ \mathbf{M}_i = \begin{bmatrix} m_i \mathbf{I}_{3\times 3} & \mathbf{0} \\ \mathbf{0} & {}^w\mathbf{I}_i \end{bmatrix}. \tag{67} $$

Now the assembled mass matrix reads

$$ \mathbf{M} = \begin{bmatrix} \mathbf{M}_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \mathbf{M}_N \end{bmatrix}. \tag{68} $$

Let $\mathbf{f}_{\text{ext}_i}$ and τ_{ext_i} be the force and torque accumulators of all external force types acting on the i^{th} body. Using the block-notation

$$ \mathbf{f}_i = \begin{bmatrix} \mathbf{f}_{\text{ext}_i} \\ \tau_{\text{ext}_i} - \omega_i \times {}^w\mathbf{I}_i \omega_i \end{bmatrix} \tag{69} $$

for the external and gyroscopic forces we assemble the $\mathbf{f}$ vector,

$$ \mathbf{f} = \begin{bmatrix} \mathbf{f}_1^T & \vdots & \mathbf{f}_N^T \end{bmatrix}^T. \tag{70} $$

Defining the blocked notation

$$ \mathbf{q}_i = \begin{bmatrix} \mathbf{x}_i^T & Q_i^T \end{bmatrix}^T \tag{71} $$

then the global assembled version reads

$$ \mathbf{q} = \begin{bmatrix} \mathbf{q}_1^T & \dots & \mathbf{q}_N^T \end{bmatrix}^T. \tag{72} $$

Notice that the dimension of $\mathbf{q}$ and $\mathbf{u}$ mismatch due to the use of a quaternion for body rotations, and this causes a problem for the generalized kinematic relation, $\dot{\mathbf{q}} = \mathbf{u}$. The problem can be solved by introducing the matrix

$$ \mathbf{H}_i = \frac{1}{2} \begin{bmatrix} -\psi_i & -\theta_i & -\xi_i \\ \phi_i & \xi_i & -\theta_i \\ -\xi_i & \phi_i & \psi_i \\ \theta_i & -\psi_i & \phi_i \end{bmatrix}, \tag{73} $$

and defining the block matrix

$$\mathbf{H} = \begin{bmatrix} \mathbf{I}_{3\times 3} & & & & \mathbf{0} \\ & \mathbf{H}_i & & & \\ & & \ddots & & \\ & & & \mathbf{I}_{3\times 3} & \\ \mathbf{0} & & & & \mathbf{H}_N \end{bmatrix}. \tag{74}$$

The discretized kinematic relation leading to the position update is now given by

$$\mathbf{q}^{+} = \mathbf{q} + \Delta t \mathbf{H}\mathbf{u}\,, \tag{75}$$

which must always be followed by a normalizing of all Q_i's to keep them a proper rotation.

1.10.2 Assembling the Matrices for a Soft Body System. For simulation of soft bodies the equations of motion changes slightly as we need to add elastic forces. Elastic forces can be computed using many different approaches ranging from particle systems to finite element methods. A rather large collection of works devoted to elastic body simulation exist in the field of computer graphics. Below we merely recap a few simple ideas that suffices for us to talk about contact forces for soft bodies and how this is different from the rigid bodies. For a more in depth treatment we refer the interested reader to the SIGGRAPH courses by Sifakis and Barbic [2012] and Kim and Eberle [2020]. For a soft deformable model, the finite element method results in the second order differential equation

$$\mathbf{M}\ddot{\mathbf{q}} + \mathbf{B}\dot{\mathbf{q}} + \mathbf{K}\mathbf{u} = \mathbf{f} + \mathbf{J}^T\boldsymbol{\lambda}. \tag{76}$$

Here $\mathbf{q}$ is the concatenation of the positions of all the nodes in the mesh, $\mathbf{u}$ is the displacement given by the difference from current deformed nodal positions $\mathbf{q}$ and undeformed positions $\mathbf{q}_0$, and $\mathbf{f}$ is the external forces acting on the soft body. The matrix $\mathbf{M}$ is the mass matrix of the soft body, $\mathbf{B}$ is the damping matrix and $\mathbf{K}$ is the stiffness matrix, and $\mathbf{J}$ is the contact Jacobian and $\boldsymbol{\lambda}$ contains the Lagrange multipliers for the contact forces. Observe that if we are in 3D and have N nodes in a linear tetrahedral mesh, then the dimensions of the introduced quantities are $\mathbf{q}, \mathbf{u}, \mathbf{f} \in \mathbb{R}^{3N}$ and $\mathbf{M}, \mathbf{B}, \mathbf{K} \in \mathbb{R}^{3N\times 3N}$. If there are K contacts, then we have $\mathbf{J} \in \mathbb{R}^{3K\times 3N}$ and $\boldsymbol{\lambda} \in \mathbb{R}^{3K}$.

We now already observe several differences compared to the rigid body case namely that there are no orientation or angular spin present in these equations, as such they are simpler as we do not need to deal with things such as quaternions. On the downside the dimensionality of the equation has exploded compared to the rigid body case. A small simple soft body can easily have more than 1000 nodes, hence the dimensionality of the equations is at least 2-3 orders of magnitude larger. Hence, solving the free motion of soft bodies is much more demanding than the rigid body case simply because there are more variables. Lastly we see a new force-type in the equations, the elastic forces. In our simple setup, we used the most simple type of elastic force, namely a linear isotropic elastic model. Proper elastic modeling is difficult and there exist many different elastic models, which one to pick and use is a whole subject in itself. The

elastic forces can be tricky to compute too, but the good news is that the soft elastic nature of the forces numerically dampen the response between nodes in the mesh. That is deformations in a soft body propagate like a wave through the body. This means that the effect of contact forces too are subject to this wave propagation nature. This is actually how real objects bounce. The nice thing compared to rigid bodies is that the numerics often gets better whereas in rigid bodies all forces in contact are instantly affecting each other, in soft bodies there is a delay, one can say the contact forces act more locally compared to rigid bodies where the effect is more global on the whole system.

We will now perform a typical time discretization using finite difference approximations. First we observe that time derivative of the velocity is the acceleration, $\dot{\mathbf{u}} = \ddot{\mathbf{q}}$, and so using first order Euler,

$$\dot{\mathbf{u}} \approx \frac{\mathbf{u}^+ - \mathbf{u}}{h}, \tag{77}$$

where $\mathbf{u}$ and $\mathbf{u}^+$ are the generalized velocities at time t and $t + h$, respectively. Similarly, integrating the generalized positions gives:

$$\mathbf{q}^+ \approx \mathbf{q} + h\mathbf{u}^+ . \tag{78}$$

Substituting the above finite difference approximations produces a velocity-level formulation of the dynamics equations, such that

$$\mathbf{M}\frac{\mathbf{u}^+ - \mathbf{u}}{h} + \mathbf{K}\left(\mathbf{q}^+ - \mathbf{q}_0\right) + \mathbf{B}\mathbf{u}^+ = \mathbf{f} + \mathbf{J}^T\boldsymbol{\lambda} . \tag{79}$$

Further manipulation, and converting forces to impulses by $\boldsymbol{\lambda} \equiv h\boldsymbol{\lambda}$, gives the form,

$$\mathbf{M}\mathbf{u}^+ - \mathbf{M}\mathbf{u} + h\mathbf{K}\left(\mathbf{q} + h\mathbf{u}^+ - \mathbf{q}_0\right) + h\mathbf{B}\mathbf{u}^+ = h\mathbf{f} + \mathbf{J}^T\boldsymbol{\lambda} . \tag{80}$$

Finally,

$$\mathbf{A}\,\mathbf{u}^+ = \mathbf{b} + \mathbf{J}^T\,\boldsymbol{\lambda}, \tag{81}$$

where

$$\mathbf{A} = \mathbf{M} + h\mathbf{B} + h^2\mathbf{K}, \tag{82a}$$

$$\mathbf{b} = \mathbf{M}\mathbf{u} + h\left(\mathbf{f} - \mathbf{K}\mathbf{q} + \mathbf{K}\mathbf{q}_0\right) . \tag{82b}$$

The $\mathbf{A}$ matrix is sparse block symmetric and positive definite matrix. Thus, if we ignore contact forces momentarily then the linear system can be solved effectively using a numerical method such as the conjugate gradient method. Having found $\mathbf{u}^+$ we may do the position update

$$\mathbf{q}^+ = \mathbf{q} + h\mathbf{u}^+. \tag{83}$$

Adding contact forces to this is straightforward done by adding non-penetration constraint and Coulomb friction law as we have outlined previously. For instance, when setting up the LCP model, we will have the system matrix defined as $\mathbf{W} \equiv \mathbf{J}\mathbf{A}^{-1}\mathbf{J}^T$, and the right-hand-side vector will be $\mathbf{w} \equiv \mathbf{J}\mathbf{A}^{-1}\mathbf{b}$. Historically the $\mathbf{W}$-matrix is called the Delassus operator. If we compare the equations to the rigid body counter part we see that the change is that $\mathbf{M}$ in the

rigid body case is replaced with $\mathbf{A}$ in the soft body case. What does this imply? Clearly $\mathbf{A}$ is of much higher dimension, and the stiffness matrix $\mathbf{K}$ may be computational heavy to actual assemble, further the inverse of $\mathbf{A}$ is wanted. One remedy is to use a lumped mass matrix and explicit time-discretization of the elastic forces in which case $\mathbf{K}$ drops out from $\mathbf{A}$ and $\mathbf{A}$ becomes a diagonal matrix. This however is often too limited as large time-step sizes causes instabilities in such a semi-implicit time-discretizaton approach. Section 4.2 elaborates deeper on other time-discretization approaches as well as a few of the numerical variations that make soft bodies contact a little different than rigid bodies contact.

So far, our treatment of the soft body simulation only contained a single soft body when adding multiple soft bodies to the system the equation of motion is replicated for each soft body. The tricky thing is that different soft bodies can have a different number of nodes and this means the block-size will vary when assembling all matrices into one system. As an example imagine we are given three soft bodies and their mass matrices are $\mathbf{M}_0$, $\mathbf{M}_1$, and $\mathbf{M}_2$ then the system mass matrix will be

$$\mathbf{M} \equiv \begin{bmatrix} \mathbf{M}_0 & 0 & 0 \\ 0 & \mathbf{M}_1 & 0 \\ 0 & 0 & \mathbf{M}_2 \end{bmatrix}. \tag{84}$$

Here the blocks will be of different size if the number of nodes in the soft bodies are different. Note that if consistent mass matrices are used the assembled mass matrix is not a diagonal matrix. Similar for the damping and stiffness matrices. The contact Jacobian matrix is slightly different if there are K contacts and the number of nodes in the soft bodies are N_0, N_1, and N_2 then $\mathbf{J} \in \mathbb{R}^{3K \times 3(N_0+N_1+N_2)}$. Each blocked row will have a form similar to Equation 20.

2 CONTACT GENERATION

Contact occurs when pairs of bodies touch inside a physics simulation. However, rather than simply touching, more likely is the scenario where the bodies actually intersect due to some overlap between their collision geometries. In this section, we discuss methods for generating contact points and normals for a variety of geometries.

The geometry used to determine whether there is a collision (or not) between two bodies can have a significant impact on the behavior of contact simulations. Often, performance is a principal consideration when selecting the type of shape used to represent the geometry of a simulation body, in which case simple shapes may be used to approximate the overall shape. We cover those in Section 2.1. However, accuracy and fidelity are also important considerations for certain applications, in which case geometrical representations with more detail are required, and for such cases we cover mesh-based representations and signed distances fields in Section 2.2 and Section 2.3 respectively.

The algorithms used to detect collisions between shape representations are specific to a pair of geometries. Throughout this section, we also provide algorithmic details about tests for various geometry combinations. In addition to determining whether or not a collision exists, the algorithms must also be extended to compute a contact normal and position. The contact position is a location that represents the overlapping volume of the two bodies, whereas the contact normal is the direction of a restorative force that is applied to separate the two bodies. In the case of the normal, it is typically useful to decide on a convention with respect to the order of bodies. For instance, we assume that contact normals are always directed from the first body toward the second body.

Essentially a contact point models proximity information between two objects. We label the objects A and B. Contact points are often used to model touching contact states as well as separation and penetrating states. The touching state is ideal for giving an intuitive description of the information associated with a contact point. Hence, we will use this state to introduce concepts. Conceptually a contact point provides three different kinds of information: a position, a normal, and a penetration (gap) measure. For continuous contact regions between two touching objects there are infinitely many points of contact. Therefore, a contact point can in general be considered as a sample point of those infinitely many points of contact. For mesh-based methods the intersection points between the local mesh features of the two objects are often used as the contact points.

Position: In a touching state a contact point has a reference to the two objects in contact, and specifies the actual points of the two objects in contact. We describe the common touching point, $\mathbf{p} \in \mathbb{R}^3$ of the two surface points with respect to two objects on the surface of each object, $\mathbf{p}_A, \mathbf{p}_B \in \mathbb{R}^3$. At the ideal touching state, one has $\mathbf{p} = \mathbf{p}_A = \mathbf{p}_B$. In case of separation or penetration this equality breaks and one may define $\mathbf{p} = \frac{1}{2}(\mathbf{p}_A + \mathbf{p}_B)$ or use some weighting of $\mathbf{p}_A$ or $\mathbf{p}_B$ based on volume or size of the objects. Positions are used as the point of action when

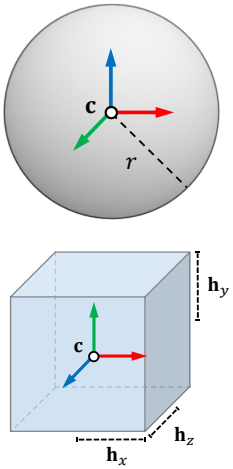
applying contact forces, and a consequence of using $\mathbf{p}_A$ and $\mathbf{p}_B$ is that ghost torques may be introduced if $\mathbf{p}_A \neq \mathbf{p}_B$.

Normal: In a touching state, the surfaces of two smooth objects will have unique parallel outward unit normals at any shared point on their respective surfaces. Let the two normals be $\hat{n}_A, \hat{n}_B$ that are oriented in opposite directions, such that $\hat{n}_A = -\hat{n}_B$. Often only one normal $\hat{n}$ is associated with a contact point. Typically, implementations use a convention to use either $\hat{n} = \hat{n}_A$ or $\hat{n} = \hat{n}_B$ as the normal associated with a contact point. For the case of penetration, the concept of a normal is perhaps less intuitive. Ideas such as using the minimum distance vector or minimum translational distance may instead be used to define $\hat{n}$. To make matters worse, for non-smooth surfaces, $\hat{n}_A$ and $\hat{n}_B$ become indeterminate. Even if they are well-defined, one may not have $\hat{n}_A = -\hat{n}_B$. Nevertheless, the normal is computed and used to apply non-interpenetration forces in the correct direction.

Penetration Measure: The penetration is defined to be zero for touching, positive for separation, and negative for intersection. This follows the same convention of the gap function ϕ introduced in Section 1. Conceptually, penetration is the distance one would need to move along $\hat{n}$ to bring the two objects into a touching state. The measure is often used to add stabilization terms to counter drift errors. Hence, it need not be a distance measure, but some monotone function that penalizes the constraint violation. For instance, volume overlap could alternatively be one such measure.

2.1 Analytic Shapes

Simple shapes may be used as an approximate representation for the surface of an object. While they are often used as elements in a hierarchical data structure, like bounding volume hierarchies, to perform cheap and conservative intersection tests, they are also commonly used as the fundamental geometry for many collision detection tasks. Implicit surface representations are common examples of collision proxies, such as spheres and planes. Whereas related surface representations may be used in other cases, such as capsules or boxes. In this section, we consider some of these simple shapes– spheres and boxes– and how to generate contacts using them.



Spheres are perhaps the most efficient 3D shapes for collision tests, both in terms of storage and computational efficiency. A sphere is defined by a center point $\mathbf{c} \in \mathbb{R}^3$ and radius $r \in \mathbb{R}$, as shown in the figure on the right.

Boxes are also popular due to their efficiency. Specifically, oriented bounding boxes (OBBs) that follow the local coordinate system of the object can be adjusted to give a good approximation of the visual geometry of an object. An OBB can be defined by center point $\mathbf{c} \in \mathbb{R}^3$, rotation matrix $\mathbf{R} \in SO(3)$ and half-width extents along the local axes $\mathbf{h} \in \mathbb{R}^3$, as shown in the figure on the right.

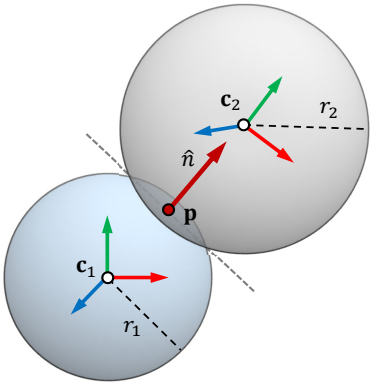


Fig. 10. A sphere-sphere intersection.

2.1.1 Sphere-Sphere Intersection. Assuming that the center position of two spheres share common reference frame, the intersection test involves simply computing the distance between the two centers positions, $\mathbf{c}_1$ and $\mathbf{c}_2$, and checking if the distance is less than the sum of the radii r_1 and r_2. In other words, if the gap function is negative or zero, such that

$$\phi = \|\mathbf{c}_2 - \mathbf{c}_1\| - (r_1 + r_2) \leq 0\,,$$

and if the above condition is true, a contact is generated. The contact normal for a sphere-sphere intersection is aligned with the vector between the two centers, and the unit length normal is computed as

$$\hat{n} = \frac{\mathbf{c}_2 - \mathbf{c}_1}{\|\mathbf{c}_2 - \mathbf{c}_1\|}\,.$$

Note that special consideration should be given to the degenerate case where $\mathbf{c}_1 = \mathbf{c}_2$, and in this case a good option is to simply specify an axis-aligned normal vector. The contact point is located anywhere along the line connecting the two centers. A common choice is the center of the overlapping region:

$$\mathbf{p} = \frac{1}{2}(\mathbf{c}_2 - r_2\hat{n}) + \frac{1}{2}(\mathbf{c}_1 + r_1\hat{n})$$

Other options for computing the contact point include using a radius-weighted sum of the distance between the centers, or a point on the surface of either sphere.

2.1.2 Sphere-OBB Intersection. The intersection test between a sphere and an OBB requires first transforming the center of the sphere into the local coordinate system of the box:

$$\mathbf{c}_{\text{local}} = \mathbf{R}_{\text{OBB}}^{-1}(\mathbf{c}_{\text{sphere}} - \mathbf{c}_{\text{OBB}})$$

Then, the closest point $\mathbf{g}$ on the box is found by considering the extents $\mathbf{h}$ in each axially aligned direction, and each coordinate $i \in \{x, y, z\}$ of the closest point is compute as

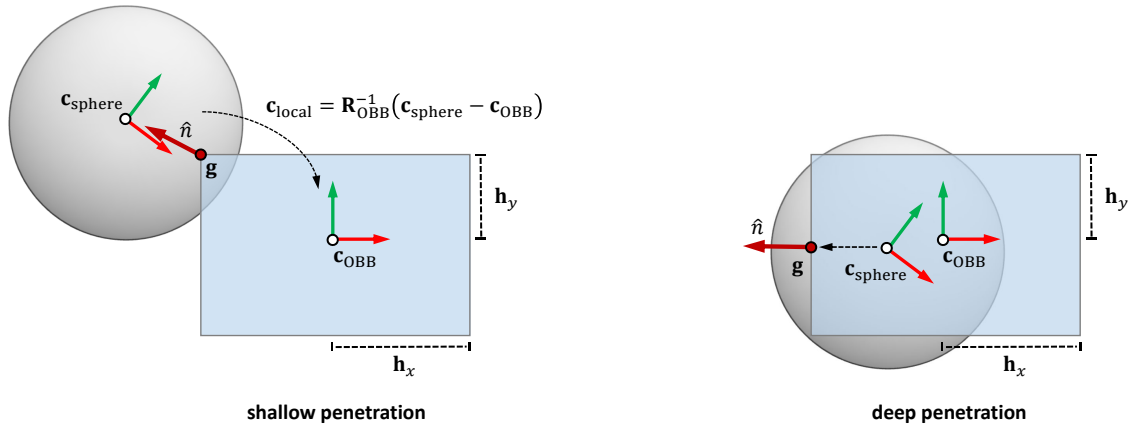


Fig. 11. Sphere-OBB intersection showing the shallow (left) and deep (right) penetration. Note that the z-dimension is not shown.

$$\mathbf{g}_i = \begin{cases} -\mathbf{h}_i & , -\mathbf{h}_i > \mathbf{c}_{\text{local},i} \\ \mathbf{h}_i & , \mathbf{h}_i < \mathbf{c}_{\text{local},i} \\ \mathbf{c}_{\text{local},i} & , \text{otherwise} \end{cases} \tag{85}$$

If the distance from the closest point to the sphere center is less than the radius of the sphere, such that $\|\mathbf{g} - \mathbf{c}_{\text{local}}\| < r$, then there is an intersection. However, special consideration must be given to the scenario where the sphere center is entirely inside the box, in which case $\mathbf{g} = \mathbf{c}_{\text{local}}$ based on Equation 85. Thus, there are two cases to consider when computing the contact normal: *shallow* and *deep* penetration.

Shallow penetration occurs if the transformed sphere center lies outside the box extents in at least one dimension. The penetration ϕ is computed as the difference between the distance $\|\mathbf{g} - \mathbf{c}_{\text{local}}\|$ and the sphere radius:

$$\phi = \|\mathbf{g} - \mathbf{c}_{\text{local}}\| - r$$

The normal may be simply computed as the unit vector that points from the sphere center to the closest point transformed in the global reference frame:

$$\hat{n} = \mathbf{R}_{\text{OBB}} \frac{\mathbf{g} - \mathbf{c}_{\text{local}}}{\|\mathbf{g} - \mathbf{c}_{\text{local}}\|}$$

Deep penetration is the case when the transformed sphere center lies entirely inside the box. This is somewhat of an extreme scenario for contact simulation, but nonetheless should be handled if robust collision is desired. The collision detection algorithm must determine which of the box faces is closest to $\mathbf{c}_{\text{local}}$. This is easily tested since the sphere center has been transformed to the local coordinate system of the box, and the test reduces to determining

which axis-aligned distance test returns the minimum penetration:

$$\phi = -\min\left(|\mathbf{h}_x - \mathbf{c}_{\text{local},x}|,\ |\mathbf{h}_y - \mathbf{c}_{\text{local},y}|,\ |\mathbf{h}_z - \mathbf{c}_{\text{local},z}| \right)$$

It is trivial to determine the closest face by considering the sign of the term that minimizes the above function. The point $\mathbf{g}$ is then computed by projecting the local sphere center onto the closest face. For example, in Figure 85, the closest point on the box surface is computed by moving the sphere center to the $-x$ face, such that

$$\mathbf{g} = \begin{bmatrix} -\mathbf{h}_x & \mathbf{c}_{\text{local},y} & \mathbf{c}_{\text{local},z} \end{bmatrix}^T .$$

Similarly, the normal of the closest box face also determines the contact normal. For example, in the case of the deep penetration example from Figure 85, the local contact normal is

$$\hat{n} = \begin{bmatrix} -1 & 0 & 0 \end{bmatrix}^T ,$$

and the world space normal is computed as $\mathbf{R}_{\text{OBB}}\hat{n}$. Finally, the contact point for both the shallow and deep penetration case may be located anywhere in the overlapping region between the OBB and the sphere. For simplicity, the closest point on the box is transformed to the global coordinate system:

$$\mathbf{p} = \mathbf{R}_{\text{OBB}}\mathbf{g} + \mathbf{c}_{\text{OBB}} .$$

2.1.3 OBB-OBB Intersection. Testing for intersection between two OBBs is a special case of testing for intersection between two convex hulls. There are several algorithms that can be considered here, such as the Gilbert–Johnson–Keerthi (GJK) distance algorithm [Gilbert et al. 1988] and the separating axis theorem (SAT) [Boyd and Vandenberghe 2004]. The latter is often preferred since the SAT allows efficient identification of the contact surfaces, as well as the penetration depth. Although, GJK may be combine with the expanding polytope algorithm (EPA) [Van Den Bergen 2003] to compute the overlap.

The first step is to determine if an intersection exists between the boxes. The SAT states that two convex objects do not intersect if a plane, onto which the geometry of each convex shape is projected, can be found where they do not overlap. Recall that the orientation of a plane is determined by a normal vector that is perpendicular to the plane. For the OBB-OBB intersection test, the following cases are considered for the separating plane orientation:

- The 3 face normals from box A ;
- The 3 face normals from box B ;
- The 9 normals formed by computing the cross-product of all edge pairs combining box A with box B.

Given two OBBs A and B, the tests involve projecting the vector between the box centers, $\mathbf{d}_{AB} = \mathbf{c}_B - \mathbf{c}_A$, onto the separating axis of the plane, $\hat{l}$. Then, we check if the sum of the projected box extents exceeds the length of $\mathbf{d}_{AB}$. Due to symmetry, additional tests can be

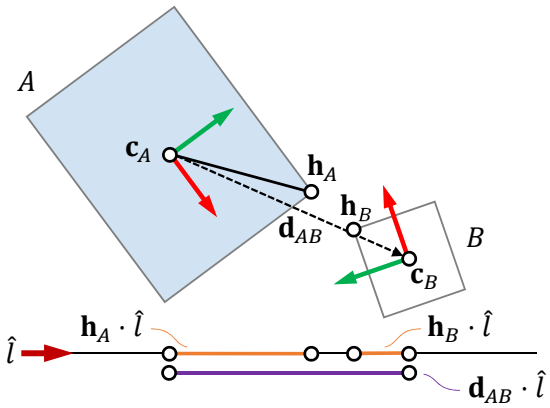


Fig. 12. A separating axis test applied to two boxes in 2D. The vector between the box centers, $\mathbf{d}_{AB}$, is projected onto the axis $\hat{l}$, and similarly for the box extents $\mathbf{h}_A$ and $\mathbf{h}_B$. Since the sum of the absolute values of the projected extents is less than $|\mathbf{d}_{AB} \cdot \hat{l}|$, there is a separation and hence no collision.

avoided by using the absolute value of the projections, which results in the separation equation

$$s = |\mathbf{d}_{AB} \cdot \hat{l}| - \left(|\mathbf{h}_A \cdot \hat{l}| + |\mathbf{h}_B \cdot \hat{l}| \right). \tag{86}$$

If an axis can be found such that $s > 0$, then there is no intersection and thus there is no need to continue. Otherwise, if $s \leq 0$ for all SAT tests, then an overlap exists and a contact will be generated.

The next step in OBB-OBB intersection requires determining which of the tests resulted in the shallowest penetration. For this purpose, specific edge pairs and faces giving a minimum separation are carefully tracked during the overlap tests. Once the test case has been identified, there are two cases to consider for computing the contact normal and point: i) an edge-edge intersection, or ii) a face intersection.

Edge-edge intersection occurs if the shallowest separation is due to a separating plane computed from an edge-edge pair. In this case, the contact normal is computed as the cross-product of the edges. That is, for edge $\mathbf{e}_\mathrm{A}$ from body A and edge $\mathbf{e}_\mathrm{B}$ from body B, the contact normal is computed as

$$\hat{n} = \frac{\mathbf{e}_A \times \mathbf{e}_B}{\|\mathbf{e}_\mathrm{A} \times \mathbf{e}_\mathrm{B}\|}. \tag{87}$$

Note that here we use that convention that edges are vectors between edge vertices. The normal is computed assuming all edge vertices are expressed in the global coordinate system, and otherwise the normal must be rotated from the local coordinate system. Only a single contact point is generated for an edge intersection. The contact point is computed by finding the closest points on each edge, such that

$$\mathbf{p}_A, \mathbf{p}_B \leftarrow \textsc{ClosestPoints}\left(\mathbf{e}_A, \mathbf{e}_B\right).$$

Finally, the penetration is then computed as the distance between the closest edge points:

$$\phi = -\|\mathbf{p}_A - \mathbf{p}_B\|$$

Alternatively, the contact point may be computed as the midpoint of the overlapping region:

$$\mathbf{p} = \frac{1}{2}(\mathbf{p}_A + \mathbf{p}_B)$$

Face intersection occurs between a face of one box and a combination of geometrical features from the other box (vertices, edges, or faces). However, rather than a single contact point, a face intersection can generate up to eight contacts depending on the configuration of the bodies. The first step in this case is to determine which of the faces, from either box, resulted in the shallowest penetration. This can be done efficiently by carefully tracking the separation values during the SAT tests and choosing the face that maximizes s, assuming that all tests return a negative separation. The responsible face is denoted as the *reference face*, and so it remains to determine the *incident face* on the other box that intersects it. Without loss of generality, let us assume that the reference face belongs to box A. The incident face f is found by searching the faces of box B for the one that most opposes the reference face. This is achieved by finding the face on box B whose normal $\hat{n}_f$ minimizes the dot product with the reference face normal, $\hat{n}_{\text{ref}}$, such that

$$f = \arg\min_{i \in F_{\text{B}}} \hat{n}_i \cdot \hat{n}_{\text{ref}},$$

where F_{B} are the faces of the box from body B. The incident face is then clipped to the sides of the reference plane using a polygon clipping algorithm (such as the Sutherland-Hodgman algorithm [Sutherland and Hodgman 1974]). For this purpose, a transformation of box B into the local coordinate system of box A is recommended, since in this reference frame the clipping planes are all axis aligned. Each clipped edge vertex on the incident face that lies below the reference face plane generates a contact. A 2D example is shown in Figure 13.

The normal for each generated contact j is the normal of the reference face:

$$\hat{n}_j = \hat{n}_{\text{ref}}$$

The contact point is found by projecting the clipped edge vertices onto the reference plane, but only for those edges that intersect the plane. Recall that if clipping occurs in the local coordinate frame of the reference box, then projection simply requires changing one of the coordinates of the clipped edge vertex to match the position of the reference face. For example, if the reference face normal is $+y$ from box A, then for clipped edge vertex $\bar{\mathbf{x}}$, the vertex projected onto the reference face has position

$$\mathbf{p} = \begin{bmatrix} \bar{\mathbf{x}}_x & \mathbf{h}_{\text{A},y} & \bar{\mathbf{x}}_z \end{bmatrix}^T,$$

where the y coordinate is assigned $\mathbf{h}_y$ from the half-width extend vector of box A. The penetration is then computed as the projection distance, which for the $+y$ face is:

$$\phi = -|\bar{\mathbf{x}}_y - \mathbf{h}_y|.$$

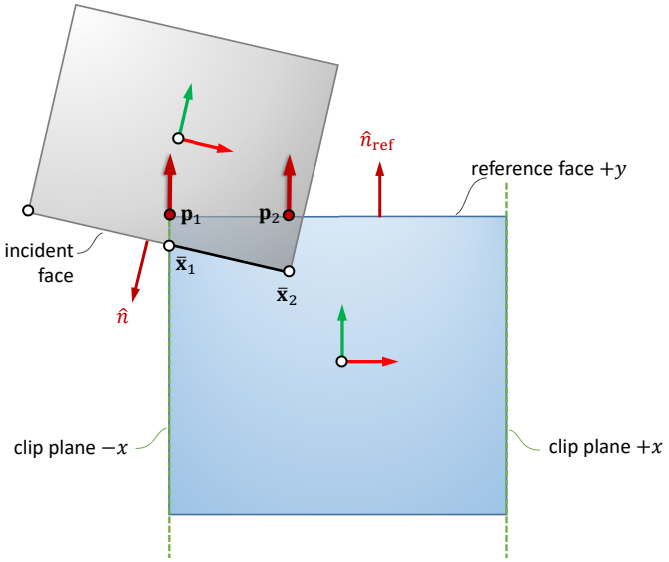


Fig. 13. Generation of contacts for OBB-OBB face intersection. This 2D example shows a collision between an edge of the grey box and the $+y$ face of the blue box. The edge vertices are clipped to the extents of the reference plane, and the clipped vertices are then projected to the reference plane to generate the contact points $\mathbf{p}_1$ and $\mathbf{p}_2$.

2.2 Mesh-based Representations

In computer graphics, objects are often represented by polygonal meshes and in particular triangle meshes are very popular for representing the shape of objects. This section briefly presents how to compute contact points for such representations.

In practical implementations, mesh-based contact generation is done in several stages. In a first stage, a broad phase collision detection method determines triangles that are sufficiently close and which pair-wise triangles need further processing. Often, bounding volume hierarchies or spatial hashing are the common tools applied in this stage. For the remainder of our discussion on mesh-based contact generation, we assume that these initial stages have been completed and instead focus on how the actual contact points are generated.

For the mesh-based contact point generation the positions and normals are defined directly from mesh features such as vertices (V), edges (E) or faces (F) of the mesh surfaces. For instance, $\mathbf{p}_A$ and $\mathbf{p}_B$ can be computed as intersection points of mesh features, or from closest points between mesh features. As we saw with box-box collisions, normals can be computed from a face normal or as a cross product of two edge vectors. Hence, the types of mesh features are often associated with the contact point and used to classify the type of the contact point. For instance, one may write a contact as (V, F) or (E, E) to identify a vertex-face generated contact point or edge-edge generated contact point, respectively.

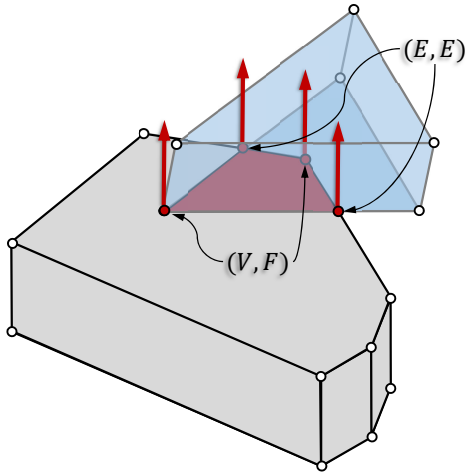


Fig. 14. Corner points of polygonal contact areas can be represented by pairs for mesh features. The vertex-face (V, F) and edge-edge (E, E) combinations forms a sufficient set of pair-types.

The illustration in Figure 14 on the right shows the concept of feature pairs. The blue contact points are defined by intersection points of mesh vertices, edges and faces from the two objects $\mathcal{A}$ and $\mathcal{B}$. The drawing has two contact points of type (V, F) and two contact points of type (E, E). There is a total of six possible types. They are rarely all used as some of them can be considered sub-types of each other. For instance, a (F, F) type can be broken down into multiple (V, F) type contact points. Associating the mesh features with a contact point has the benefit of being able to track a contact point over time simply by using labels of features and objects. Feature labels are convenient to test for redundancy without having to use floating point comparisons.

Figure 14 illustrates the ideal case where there is no penetration or separation between the two objects; they are simply touching. However, due to the nature of the discrete time stepping scheme we introduced earlier in Section 1, it is expected that objects will be slightly separated or that penetrations will exist. Hence, it becomes tricky to define what constitutes a contact point. In many practical implementations, it is common to simply take mean positions of closest points between vertex and face pairs or between edge and edge pairs. The distance between closest points is then used as a measure of penetration.

2.2.1 Continuous Collision Detection of Mesh Features. Rather than dealing with the difficulties that arise from using a discrete collision detection scheme, a more intelligent approach is to use *continuous collision detection* (CCD). This type of method processes every collision event that might occur for a given time span (i.e., a time-step). There are two flavors of continuous collision detection: (i) look forward in time, from the current instance of invocation to the

next expected instance, or (ii) look backward in time, from the current instance of invocation to the previous instance.

The initial stages of CCD often use a time-swept volume around the geometric features of the mesh-based objects. If the volumes intersect, then a more detailed computation is initiated to find the time of first contact, at which point one can simply compute normals and contact points as outlined above.

The CCD can be used in the discrete time-stepping method in many ways. The smallest computed time-of-impact can be used as a limit on how big a time-step one can use. Alternatively, one can take the CCD computed normals and points and warp them to the start of the time-step. In the case of the latter, a heuristic is required to measure the penetration depth of an earlier instant.

When looking backward in time we are actually seeking the first point of contact. This point in space is attractive for simulation methods, since it implies the point in time to which we should "rewind" the simulation in order to avoid penetration. Let us dig a bit deeper into how to predict this point using geometric features of the mesh.

Assuming that the time-step is small, it is justifiable that the motion between two instances of time may be considered linear. However, the errors based on this assumption increase with the size of the time-step, or $O(h)$. From a convergence theory point of view, we can simply let the time-step go to zero and the approximation would be exact. In practice, this is impossible, but the error can be made so small that it is negligible compared to errors coming from numerical inaccuracies and round off. Other assumptions about the motion can sometimes be made, such that it is a screw motion, which is commonly done for rigid bodies and can allow for larger time-steps in practice. However, we limit the assumption to linear motion for small time-steps since the method we outline should be sufficiently general to work for both rigid and soft bodies.

We can compute the space swept by a single triangle over the time-step. Consider that the three vertices of a triangle move with velocities $\mathbf{u}_0$, $\mathbf{u}_1$, and $\mathbf{u}_2$. Then, for a forward prediction scheme and based on the linear motion assumption, the triangle will have candidate positions at the next time step given by

$$\mathbf{x}_0' = \mathbf{x}_0 + h\mathbf{u}_0, \tag{88a}$$

$$\mathbf{x}_1' = \mathbf{x}_1 + h\mathbf{u}_1, \tag{88b}$$

$$\mathbf{x}_2' = \mathbf{x}_2 + h\mathbf{u}_2\,. \tag{88c}$$

However, for a backward prediction scheme, the candidate positions would be

$$\mathbf{x}_0' = \mathbf{x}_0 - h\mathbf{u}_0, \tag{89a}$$

$$\mathbf{x}_1' = \mathbf{x}_1 - h\mathbf{u}_1, \tag{89b}$$

$$\mathbf{x}_2' = \mathbf{x}_2 - h\mathbf{u}_2\,. \tag{89c}$$

In the case of backward prediction, we know the exact candidate position without having to compute them as above. Instead, the linear velocity may be computed as

$$\mathbf{u}_0 = \frac{\mathbf{x}_0 - \mathbf{x}_0'}{h}, \tag{90a}$$

$$\mathbf{u}_1 = \frac{\mathbf{x}_1 - \mathbf{x}_1'}{h}, \tag{90b}$$

$$\mathbf{u}_2 = \frac{\mathbf{x}_2 - \mathbf{x}_2'}{h}\,. \tag{90c}$$

A simple preliminary test would be to create two axis aligned bounding boxes (AABBs) that enclose the triangles– one using the current positions and the other using the candidate positions– and test them for overlap. Only if the bounding boxes overlap can there be a contact that occurred between the mesh features during the interval h.

The next step is to examine all possible contact types for the triangle pair. Specifically, (V, F) and (E, E) contacts. In some approaches, like the ones based on bounding volume hierarchies, the triangle pairs are split into sub-types. Whereas for other approaches, like spatial hashing, they work directly with the mesh feature pairs.

Due to our assumption of linear motion over the time interval, h, the time of contact is characterized by either: (i) the vertex lying in the face plane in the case of a (V, F) contact, or (ii) by two edges being co-planar in the case of an (E, E) contact. Regardless of the case, we seek a point in time where four points all lie in the same plane.

2.2.2 Continuous Collision Detection of Face-Vertex Pair. The idea is to use three points to compute a plane normal, and then use one of the three points to find a distance between the plane and the origin. Finally, the fourth point is used for a point in the plane test. For convenience, in the case of the (V, F), let us label the points of the triangle in counterclockwise order: $\mathbf{x}_1$, $\mathbf{x}_2$, and $\mathbf{x}_3$ and the point of the vertex $\mathbf{x}_4$. In the case of an (E, E) type, we label the end-points of the first edge $\mathbf{x}_1$ and $\mathbf{x}_2$, and the end-points of the second edge $\mathbf{x}_3$ and $\mathbf{x}_4$. The corresponding velocities are in both cases labeled $\mathbf{u}_1$, $\mathbf{u}_2$, $\mathbf{u}_3$, and $\mathbf{u}_4$. Now we'll set up two vectors $\mathbf{x}_2 - \mathbf{x}_1$ and $\mathbf{x}_3 - \mathbf{x}_1$, and take the cross product of these two to obtain a plane normal

$$\hat{n} = (\mathbf{x}_2 - \mathbf{x}_1) \times (\mathbf{x}_3 - \mathbf{x}_1)\,. \tag{91}$$

The point $\mathbf{x}_1$ must lie in the plane, so the distance, d, to the origin is given by

$$d = \hat{n} \cdot \mathbf{x}_1. \tag{92}$$

In order for the last point to lie in the plane, then its distance to the plane must be zero, that is,

$$\hat{n} \cdot \mathbf{x}_4 - d = 0\,. \tag{93}$$

Substitution leads to

$$((\mathbf{x}_2 - \mathbf{x}_1) \times (\mathbf{x}_3 - \mathbf{x}_1)) \cdot (\mathbf{x}_4 - \mathbf{x}_1) = 0. \tag{94}$$

Whenever this equation is fulfilled, the four points will lie in the same plane. The last thing to consider is the motion of the points whose positions change with time, such that

$$((\mathbf{x}_2(t) - \mathbf{x}_1(t)) \times (\mathbf{x}_3(t) - \mathbf{x}_1(t))) \cdot (\mathbf{x}_4(t) - \mathbf{x}_1(t)) = 0\,. \tag{95}$$

Here, the objective is to determine the smallest non-negative value $t < h$ that makes the above equation true. From the linear motion assumption, the trajectory of each point may be written as

$$\mathbf{x}_1(t) = \mathbf{x}_1 + \mathbf{u}_1 t, \tag{96a}$$
$$\mathbf{x}_2(t) = \mathbf{x}_2 + \mathbf{u}_2 t, \tag{96b}$$
$$\mathbf{x}_3(t) = \mathbf{x}_3 + \mathbf{u}_3 t, \tag{96c}$$
$$\mathbf{x}_4(t) = \mathbf{x}_4 + \mathbf{u}_4 t\,, \tag{96d}$$

and by substitution we obtain

$$(((\mathbf{x}_2 - \mathbf{x}_1) + (\mathbf{u}_2 - \mathbf{u}_1)\,t) \times ((\mathbf{x}_3 - \mathbf{x}_1) + (\mathbf{u}_3 - \mathbf{u}_1)\,t)) \cdot ((\mathbf{x}_4 - \mathbf{x}_1) + (\mathbf{u}_4 - \mathbf{u}_1)\,t) = 0\,. \tag{97}$$

This is a cubic polynomial in t with an analytical solution. Once the three roots are found, the positions of the vertices can also be found at the specific values of t.

Having found the points in time where the four points are co-planar, we will finally compute the candidate for the contact point $\mathbf{p} = \mathbf{x}_4 + \mathbf{u}_4 t$, if this point is inside the triangle face at the time of contact then a contact point is reported with position $\mathbf{p}$, normal $\mathbf{n} = (\mathbf{x}_2(t) - \mathbf{x}_1(t)) \times (\mathbf{x}_3(t) - \mathbf{x}_1(t))$ and penetration depth of $\phi = 0$. In most simulation approaches the state will be reset to the earliest point in time where a collision is found hence the penetration depth will always be zero.

2.2.3 Continuous Collision Detection of Edge-Edge Pair. As in the (V, F) case, the (E, E) case requires finding the roots of a polynomial equation, and then examining the geometries in ascending temporal order. For simplicity, let $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$, and $\mathbf{x}_4$ denote the edge geometry positions at the instant in time corresponding to a root. First, we test whether the edges are parallel. This is the case when

$$(\mathbf{x}_2 - \mathbf{x}_1) \times (\mathbf{x}_4 - \mathbf{x}_3) = 0\,. \tag{98}$$

In practice, an equality test will not work, so we use a threshold test instead. If the test succeeds, then a dimension reduction technique can be used by projecting the vertices of one edge onto the line running through the edge. However, one could also simply drop this case, since if a touching contact exists, it could just as well be represented by two (V, F) type contacts. Therefore, we will only consider the case where the two edges are touching at exactly one point.

Parameterization of the two edges with the a and b parameters, yields

$$\mathbf{x}_1 + a\left(\mathbf{x}_2 - \mathbf{x}_1\right), \tag{99a}$$

$$\mathbf{x}_3 + b\left(\mathbf{x}_4 - \mathbf{x}_3\right). \tag{99b}$$

The touching point between the two lines must also be the closest point, and the closest point is characterized by the minimum distance, so we seek the values of a and b that minimizes

$$\left\| \left(\mathbf{x}_1 + a\left(\mathbf{x}_2 - \mathbf{x}_1\right)\right) - \left(\mathbf{x}_3 + b\left(\mathbf{x}_4 - \mathbf{x}_3\right)\right) \right\|_2. \tag{100}$$

Taking the derivative with respect to a and b yields the so-called normal equations:

$$\begin{bmatrix} \left(\mathbf{x}_2 - \mathbf{x}_1\right) \cdot \left(\mathbf{x}_2 - \mathbf{x}_1\right) & -\left(\mathbf{x}_2 - \mathbf{x}_1\right) \cdot \left(\mathbf{x}_4 - \mathbf{x}_3\right) \\ -\left(\mathbf{x}_2 - \mathbf{x}_1\right) \cdot \left(\mathbf{x}_4 - \mathbf{x}_3\right) & \left(\mathbf{x}_4 - \mathbf{x}_3\right) \cdot \left(\mathbf{x}_4 - \mathbf{x}_3\right) \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \left(\mathbf{x}_2 - \mathbf{x}_1\right) \cdot \left(\mathbf{x}_3 - \mathbf{x}_1\right) \\ -\left(\mathbf{x}_4 - \mathbf{x}_3\right) \cdot \left(\mathbf{x}_3 - \mathbf{x}_1\right) \end{bmatrix}. \tag{101}$$

Solving for a and b computes the closest points between the two lines running through the two edges. If a and b both are both in the range $[0, 1]$, then a contact point can be reported at location $\mathbf{p} = \left(\mathbf{x}_1 - \mathbf{x}_3\right) + a\left(\mathbf{x}_2 - \mathbf{x}_1\right) - b\left(\mathbf{x}_4 - \mathbf{x}_3\right)$. The normal will be given by the edge cross products similar to Section 2.1.3. Except here we are using CCD so the penetration depth is trivially zero.

2.2.4 Multivariate Style of Continuous Collision Detection. Alternatively, the CCD functional can be expressed in a multivariate form [Brochu et al. 2012; Wang et al. 2021]. For the (V, F) case, this is

$$\left((1 - u - v)\mathbf{x}_1(t) + u\mathbf{x}_2(t) + v\mathbf{x}_3(t)\right) - \mathbf{x}_4(t) = \mathbf{0}, \tag{102}$$

where u and v are the Barycentric coordinates of the triangle and a domain of $t \in [0, h]$ and $u \geq 0$, $v \geq 0$, and $u + v \leq 1$. Similarly, for the (E, E) case, the multivariate form is

$$\left(\mathbf{x}_1(t) + a(\mathbf{x}_2(t) - \mathbf{x}_1(t))\right) - \left(\mathbf{x}_3(t) + b(\mathbf{x}_4(t) - \mathbf{x}_3(t))\right) = \mathbf{0}, \tag{103}$$

with a domain of $t \in [0, h]$ and $a, b \in [0, 1]$. In both cases, these functions map from $\mathbb{R}^3 \mapsto \mathbb{R}^3$ and involve root-finding with a system of three quadratic equations. Geometrically, this is equivalent to determining if the origin is contained inside the co-domain of the function.

The advantage of this form is that it eliminates the need to do post check on the computed solution. These checks are particularly problematic in degenerate cases where the univariate formulation has infinite roots. The disadvantage of this form is that it requires finding the roots of a multivariate equation, which is more challenging and time consuming than solving a univariate equation.

2.2.5 Tight-Inclusion. Wang et al. [2021] introduce a method for solving the system of equations using inclusion-based root-finding [Redon et al. 2002; Snyder 1992]. From a one dimensional interval $i = [a, b]$, we can define an n-dimensional interval as $I = i_1 \times \cdots \times i_n$, where i_k

are intervals. The width of an interval is defined as $w([a,b]) = b - a$. Similarly, the width of an n-dimensional interval is

$$w(I) = \max_{k=\{1,\ldots,n\}} w(i_k).$$

Given an m-dimensional interval D and a continuous function $g\colon \mathbb{R}^m \mapsto \mathbb{R}^n$, an inclusion function for g, written $\square g$, is a function such that

$$\forall x \in D \quad g(x) \in \square g(D)\,.$$

In other words, $\square g(D)$ is a n-dimensional interval bounding the range of g evaluated over an m-dimensional interval D bounding its domain. We call the inclusion function $\square g$ of a continuous function g convergent if, for an interval X, we have

$$w(X) \to 0 \implies w\big(\square g(X)\big) \to 0\,.$$

Data: I_0, g, δ
Result: res
res $\leftarrow \emptyset$;
$S \leftarrow \{I_0\}$;
while $S \neq \emptyset$ **do**
  $I \leftarrow \text{pop}(S)$;
  $I_g \leftarrow \square g(I)$;
  if $0 \in I_g$ **then**
    if $w(I) < \delta$ **then**
      res $\leftarrow$ res $\cup\, I$;
    else
      $I_1, I_2 \leftarrow \text{split}(I)$;
      $S \leftarrow S \cup \{I_1, I_2\}$;
    end
  end
end

Algorithm 1: Bisection Inclusion-based Root-finding

Using an convergent inclusion function, Snyder [1992] show how to define a simple inclusion-based root-finding method using bisection as outlined in Algorithm 1. This amounts to evaluating the inclusion function and determining if the origin is contained in the inclusion function. If it is either split the domain interval and repeating the process on each sub interval or terminating when the domain is sufficiently small. If the origin is not contained and there are no other sub-intervals to evaluate then we can be sure the function does not have a root in the initial domain provided.

In general, building these inclusion functions can be done by using interval arithmetic [Snyder 1992]. For our particular functions (Equation 102 and Equation 103), Wang et al. [2021] propose

a closed form inclusion function. Which is to construct a bounding box around the corner vertices of the codomain.

2.2.6 Implementation Notes. The above methods assume the computation is done under infinite precision (i.e., real numbers) in order to provide an exact time of impact. However, when computed using floating-point numbers, numerical error and round-off can result in missing true collisions (false negative) and detecting false collisions (false positive). To address these inaccuracies, both Brochu et al. [2012] and Tang et al. [2014] propose "exact" methods for CCD. That is, they claim their methods are robust to numerical error and produce zero false positives and false negatives.

Brochu et al. [2012] propose a geometric approach to solving the problem. At a high level, they take the multivariate formulations (Equation 102 and Equation 103) and, in order to determine if the origin is contained within the co-domain, they shoot rays from the origin outwards. Wang et al. [2022] improved upon this by both fixing some degenerate-cases are handled and proposing a approach that is able to use floating-point computation directly instead of having to rely on expensive

Tang et al. [2014] instead propose an algebraic method in which they decompose the cubic polynomial given by the univariate formulations (Equation 97 and Equation 98) into quadratic and linear functions. They then use sign classification of the coefficient of these functions to determine if there exists a root in the time interval.

Unfortunately, as Wang et al. [2021] showed, each of these methods can produce false positives and false negatives. While not providing an exact method, Wang et al. [2021] (as we outlined above) propose a *conservative* method that avoids false negatives at the cost of potential false positives. This area of exact CCD remains an open problem.

2.3 Signed Distance Fields

Signed distance fields (SDFs) represent the shape of an object by an implicit function $s(\mathbf{x}) : \mathbb{R}^3 \rightarrow \mathbb{R}$. Unlike mesh-based representations, which use explicit geometry, SDFs give the signed distance of some point $\mathbf{x}$ from the surface of an object. The sign (positive or negative) indicates whether the point lies outside or inside of the object. Specifically, points interior to the shape have $s(\mathbf{x}) < 0$, whereas exterior points have $s(\mathbf{x}) > 0$, and points on the surface satisfy $s(\mathbf{x}) = 0$. For example, the SDF of a circle is shown on the left side of Figure 15, and different colors are used to draw regions with positive and negative distances and the zero-level isosurface.

In this section, we present some details about performing collision detection and generating contacts using SDFs. Specifically, we show how polygonal meshes can be tested for intersection using SDFs. This offers an alternative to the mesh-mesh based collision detection outlined in Section 2.2, since it is straightforward to generate an SDF from a polygon mesh that can then be used for efficient collision detection. However, we do not cover how to generate SDFs, and instead refer the reader to related work on this topic [Jones et al. 2006].

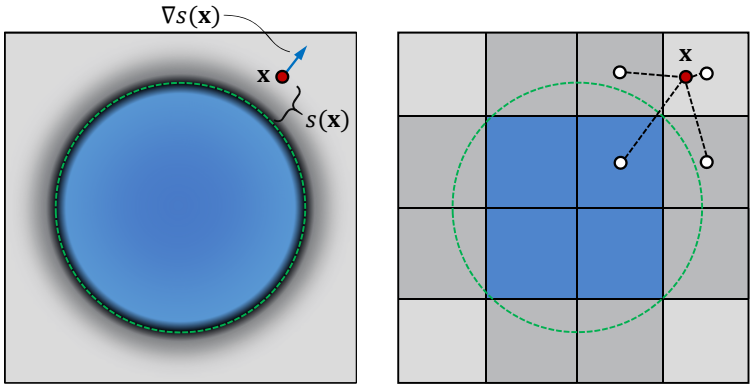


Fig. 15. Left: a 2D example showing the SDF of a circle. Locations with positive distances are drawn in light grey, whereas negative distances are drawn in light blue; the zero-level isosurface is draw in black. Right: the same SDF stored on a 4×4 grid. The signed distance $s(\mathbf{x})$ and gradient $\nabla s(\mathbf{x})$ may be computed from adjacent grid cells (i.e., using bilinear interpolation). The grid resolution can be increased to improve the approximated shape and recover finer scale details, but at the cost of increased complexity.

In addition to computing a scalar field of signed distances, is it also common to compute its gradient, $\nabla s(\mathbf{x}) \in \mathbb{R}^3$. Observe that the gradient function returns a 3D vector that gives direction of the closest point on the surface from $\mathbf{x}$. A desirable property of the gradient function is that the returned vector is unit length, such that

$$\|\nabla s(\mathbf{x})\| = 1 \,.$$

Therefore, $\nabla s(\mathbf{x})$ gives the vector field of outward facing unit normal vectors. However, this property of the field may not always be true, for instance if the closet point is not unique (e.g., the center of a sphere), but in practice the gradient function can be constructed so as to preserve this property.

2.3.1 SDF-Point Intersection. The canonical intersection test using an SDF is with a point. The first step is to determine if $s(\mathbf{x}) \leq 0$ is true, which means that either $\mathbf{x}$ is an interior point or touching the surface. If true, the gap function ϕ is simply the signed distance provided by the field function:

$$\phi = s(\mathbf{x}) \tag{104}$$

Next, let us consider how to compute the contact normal. Generally speaking, the normal is given by the forcing direction that will most quickly move $\mathbf{x}$ outside the field. This is in fact the gradient evaluated at $\mathbf{x}$, and so the contact normal may be computed as:

$$\hat{n} = \nabla s(\mathbf{x})^T \tag{105}$$

Note again that special care should be taken to orient the normal based on the convention used by the collision system (e.g., from the first body toward the second body).

Finally, there are two obvious choices for computing the contact position. One option is to situate the contact point $\mathbf{p}$ at the surface of the SDF, which is computed by projecting $\mathbf{x}$ along the gradient:

$$\mathbf{p} = \mathbf{x} - \nabla s(\mathbf{x})^T s(\mathbf{x}) \tag{106}$$

Conversely, the query point may be used, such that

$$\mathbf{p} = \mathbf{x}\,. \tag{107}$$

2.3.2 SDF-Mesh Intersection. Generating a contact for the SDF and point intersection provides a useful building block for collision with more complex geometries, such as polygonal meshes. Collision detection between meshes can be computationally costly, as we saw in Section 2.2. However, a common use case of SDFs is accelerating mesh-mesh collision detection by computing a signed distance field for each mesh and then performing approximate intersection tests using the SDF of one object versus the mesh of the other. Luckily, there are robust methods for building an SDF from arbitrary polygonal soups [Xu and Barbič 2014].

In Section 2.3.1, we saw that it is straightforward to generate contacts for the case of a point intersecting an SDF. This naturally leads to an intersection test with a polygonal mesh, whereby each vertex $\mathbf{x} \in V$ is tested. Let us assume that we want to test a vertex from the mesh of body B with the signed distance field of body A. The first step is to transform vertex $\mathbf{v}$ into the local coordinate system of body A, since it is often more efficient to transform the mesh vertices into the field coordinate frame, rather than the other way around. A local vertex position is computed, such that

$$\mathbf{x}_{\text{local}} = \mathbf{R}_{\text{A}}^{-1} \left(\mathbf{R}_{\text{B}} \left(\mathbf{x} + \mathbf{c}_{\text{B}} \right) - \mathbf{c}_{\text{A}} \right) , \tag{108}$$

where $\mathbf{c}_{\text{A}} \in \mathbb{R}^3$, $\mathbf{R}_{\text{A}} \in SO(3)$ and $\mathbf{c}_{\text{B}} \in \mathbb{R}^3$, $\mathbf{R}_{\text{B}} \in SO(3)$ are the positions and rotation matrices of bodies A and B respectively. Then, if $s(\mathbf{x}_{\text{local}}) < 0$, an intersection exists and Equation 104-Equation 107 may be used to generate the depth, contact normal and position. This process is summarized in Algorithm 2.

However, testing only against the vertices of the polygonal mesh can lead to missed collisions and other artifacts. For instance, this is likely if the mesh geometry is coarse and the SDF surface contains thin features. These issues can be mitigated by adding more vertices to the mesh, although theoretically this will never fully eliminate penetration artifacts and may also impact performance.

Recently, Macklin et al. [2020a] proposed a refinement technique to compute exact collision between SDFs and polygonal meshes. Rather than considering each vertex of the mesh, each face is considered. Assuming a triangle mesh, each face is defined by vertices $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3 \in \mathbb{R}^3$. A point on the interior or boundary of the face can be described by its barycentric coordinates

Input: mesh F, V, SDF s, gradient field ∇s
Result: List of contacts C

$\forall \mathbf{x} \in V$
compute $\mathbf{x}_{\text{local}}$ by transforming $\mathbf{x}$ (Equation 108)
if $s(\mathbf{x}_{local}) < 0$ **then**
    $\phi = s(\mathbf{x}_{\text{local}})$
    $\hat{n} = \nabla s(\mathbf{x}_{\text{local}})^T$
    $\mathbf{p} = \mathbf{x}$
    transform $\hat{n}$ and $\mathbf{p}$ to world coordinates
    $\mathrm{C} \leftarrow \mathrm{C} \cup \{\hat{n}, \mathbf{p}, \phi\}$
end

Algorithm 2: SDF-Mesh contact generation

u, v, w, such that

$$\mathbf{x}(u, v, w) = u\mathbf{x}_1 + v\mathbf{x}_2 + w\mathbf{x}_3\,, \tag{109a}$$

$$\text{subject to } u, v, w \geq 0\,, \tag{109b}$$

$$u + v + w = 1\,. \tag{109c}$$

The closest point on the face can then be found by a minimization over the barycentric coordinates:

$$\arg\min_{u,v,w} \; s(u\mathbf{x}_1 + v\mathbf{x}_2 + w\mathbf{x}_3) \tag{110}$$

The minimization in Equation 110 is a non-linear optimization problem, and thus requires an iterative approach. For instance, a projected gradient descent method. However, this requires computing the derivative of the distance field at each candidate solution with respect to the barycentric coordinates, which is

$$\mathbf{d} = \begin{bmatrix} \frac{\partial s}{\partial u} \\ \frac{\partial s}{\partial v} \\ \frac{\partial s}{\partial w} \end{bmatrix} = \begin{bmatrix} \nabla s(\mathbf{x})\mathbf{x}_1 \\ \nabla s(\mathbf{x})\mathbf{x}_2 \\ \nabla s(\mathbf{x})\mathbf{x}_3 \end{bmatrix}.$$

At the kth iteration of the gradient descent algorithm, the candidate solution to Equation 110 is then updated by a fixed-point iteration, such that

$$\mathbf{x}^{(k+1)} \leftarrow \textsc{ProjBarycentric}(\mathbf{x}^{(k)}) - \alpha\mathbf{d}\,.$$

Here, α is a fixed step size and ProjBarycentric projects the solution u, v, w to the manifold defined by Equation 109b-Equation 109c. After the solver terminates, if $s(\mathbf{x}(u, v, w)) < 0$ then a contact is generated.

2.3.3 Implementation Notes. In practice, an SDF is often represented by a discrete data structure, such as a dense or sparse voxel grid. Each grid cell (i, j, k) stores the signed distance, and thus $s(\mathbf{x})$ is computed by first identifying the grid cell containing $\mathbf{x}$ and then performing an interpolation using the neighboring grid cells (see the right side of Figure 15). In the case of the gradient, $\nabla s(\mathbf{x})$, it may be computed on-demand using finite differences, by analytic gradients, or precomputed and stored alongside distance values on the grid.

The grid cell size will have an effect on the quality of the collision detection, and often the grid cell size is determined based on the smallest features of the source geometry. However, this may result in heavy memory consumption if a uniform grid is used to store complex surfaces. Frisken et al. [2000] proposed an adaptive SDF to address this problem, where an octree subdivision is used to recursively refine the grid until the interpolated field values closely match the original surface. Similarly, Koschier et al. [2016] improved the accuracy of SDFs for complex geometry by adaptively refining the resolution of the grid, as well as the polynomial degree used to represent local field details.

It is trivial to extend the contact generation techniques presented in this section to analytical shapes, such as spheres. However, in the case that the field is stored using a discrete grid, it is important that the grid is extended by a margin around the zero-level isosurface of the source geometry to avoid discontinuities and ensure that valid field data exists for possible query points.

3 NUMERICAL METHODS

In this section, we present a collection of numerical methods for solving the frictional contact problems. These can be roughly classified into two general approaches: i) pivoting methods and ii) iterative methods. Pivoting methods focus on determining a labeling of variables based on the complementarity conditions. Once the labeling is known, an exact solution to the complementarity problem may be computed if a direct solver is used to solve the resulting subproblem. Conversely, iterative methods make incremental progress toward the exact solution, with better approximations being obtained at each iteration. Additionally, there are hybrid approaches that combine these schemes.

The fundamental goal of applying these numerical methods is to solve for the constraint impulses λ, while also ensuring that the feasibility and complementarity conditions of the various contact models are met. However, some of the frictional contact models from Section 1 require computing other parameters that are important for producing the correct behavior, such as the slack β parameter used by the polyhedral LCP. Therefore, we depart from earlier convention and now use $\mathbf{x}$ to denote the vector of unknowns that is computed by each numerical method, rather than just the constraint impulses $\boldsymbol{\lambda}$.

We have chosen to cover the topic of proximal operators as these are a quite general and powerful framework for expression the contact physics as well as with ease provide a simple way to derive numerical methods. Our treatment is extensive and show many facets of how to use proximal operators in a simulator.

Table 2 gives a high level classification road map of the methods we cover in this chapter. We cover methods for a wide range of models. One would often have to first decide upon which model to use in a simulator before carefully picking the numerical method. When choosing the numerical method it is often trade offs in terms of performance, accuracy and robustness that must be considered. The application context often dictates which trade offs that can be acceptable. For instance, in a gaming context accuracy may not be necessary, but high performance and robustness would be extremely important, in digital prototyping high fidelity may be more of a concern and hence accuracy may out weight performance considerations. Many methods and ideas can be used in combination as well to create hybrid methods. We will not cover this aspect in these course notes. In general there is not a single one method that is better than all the rest. The right choice really depends on the modeling needs and the context at hand.

3.1 Pivoting Methods

Pivoting methods exploit the combinatorial nature of complementarity problems to find a labeling of the variables based on the complementarity and feasibility conditions imposed by the model. With this labeling, which is sometimes referred to as the *index set*, it is possible to identify constraint variables as being *free* or *tight*, and the latter case is further decomposed into *tight lower* and *tight upper* variables. This labeling permits a partitioning of a linear system

Table 2. This table presents an overview of the main numerical methods covered in this chapter as well as a summary of some their traits.

Model	Methods	Benefits	Disadvantages	Sec.
(B)LCP	Incremental Pivoting, Block Principal Pivoting.	High accuracy and supports large mass ratios.	Computational expensive.	3.1, 3.1.2, 3.1.3.
(B)LCP	Splitting Methods: Projected Jacobi, Gauss-Seidel (PGS), Successive-Over-Relaxation (PSOR).	Fast performance and very robust.	Inaccurate and poor large mass ratio handling, linear convergence rate.	3.2, 3.2.1.
(B)LCP	Staggering.	Converts LCP into two simpler coupled quadratic programming (QP) problems.	No convergence proof exist.	3.2.5.
(B)LCP	Subspace-Minimization.	Improved accuracy of active contacts.	Solving linear subsystem can be expensive.	3.2.6.
(B)LCP	Non-Smooth Nonlinear Conjugate Gradient.	Improved accuracy at two times cost of PGS and better handling of large mass ratios.	May need many iterations before benefit kicks in, and may not improve over PGS in all cases.	3.2.7.
NCP	Non-Smooth Newton Method.	Quadratic convergence rate and support for nonlinear friction models.	Assembling and solving a linear sub-system in every Newton iteration can be computational expensive.	3.3.
Proximal operator	PROX Gauss-Seidel and PROX Jacobi methods.	Supports exact regularization, strong convergence guarantee, and supports nonlinear friction models. Can achieve super-linear convergence rate.	Difficult to select regularization parameter values.	3.4.

based on the index set. For instance,

$$\begin{bmatrix} \mathbf{A}_{\mathsf{F},\mathsf{F}} & \mathbf{A}_{\mathsf{F},\mathsf{L}} & \mathbf{A}_{\mathsf{F},\mathsf{U}} \\ \mathbf{A}_{\mathsf{L},\mathsf{F}} & \mathbf{A}_{\mathsf{L},\mathsf{L}} & \mathbf{A}_{\mathsf{L},\mathsf{U}} \\ \mathbf{A}_{\mathsf{U},\mathsf{F}} & \mathbf{A}_{\mathsf{U},\mathsf{L}} & \mathbf{A}_{\mathsf{U},\mathsf{U}} \end{bmatrix} \begin{bmatrix} \mathbf{x}_{\mathsf{F}} \\ \mathbf{x}_{\mathsf{L}} \\ \mathbf{x}_{\mathsf{U}} \end{bmatrix} + \begin{bmatrix} \mathbf{b}_{\mathsf{F}} \\ \mathbf{b}_{\mathsf{L}} \\ \mathbf{b}_{\mathsf{U}} \end{bmatrix} = \begin{bmatrix} \mathbf{v}_{\mathsf{F}} \\ \mathbf{v}_{\mathsf{L}} \\ \mathbf{v}_{\mathsf{U}} \end{bmatrix}, \tag{111}$$

where F is the set of free variables, L is the set of tight lower variables, and U is the set of tight upper variables. Each sub-block of $\mathbf{A}$ in Equation 111 thus corresponds to selecting rows and columns of the lead matrix that correspond to the rows and columns of each set. For instance, $\mathbf{A}_{\mathsf{F},\mathsf{L}}$ contains row indices from F and column indices from L. The rows of vectors $\mathbf{b}$, $\mathbf{x}$, and $\mathbf{v}$ may be similarly partitioned according to the index set labels.

For contact problems, the index set of each variable is determined by inspection of each v_i and x_i. Variables in $i \in \mathsf{F}$ are those for which the value of $\mathbf{x}_i$ is not restricted by the feasibility condition of the constraint (e.g., the lower and upper bounds). Whereas tight variables $i \in \mathsf{L} \cup \mathsf{U}$ have a value which is determined by the feasibility conditions.[1]

To better understand how this partitioning works, let us consider the BLCP formulation, where $\mathbf{x} = \boldsymbol{\lambda}$ are the constraint impulses. From Equation 41, we can define the index sets based on the feasibility and complementarity conditions of the BLCP based on three cases:

$$\{\forall i \in \mathsf{F} \ : \ \lambda_i^{\text{lo}} < \lambda_i < \lambda_i^{\text{hi}} \text{ and } v_i = 0\}, \tag{112a}$$

$$\{\forall i \in \mathsf{L} \ : \ \lambda_i = \lambda_i^{\text{lo}} \text{ and } v_i > 0\}, \tag{112b}$$

$$\{\forall i \in \mathsf{U} \ : \ \lambda_i = \lambda_i^{\text{hi}} \text{ and } v_i < 0\}. \tag{112c}$$

For non-interpenetration constraints, which are only lower bounded, this labeling implies that

$$i \in \mathsf{F} \implies \lambda_i > 0 \text{ and } v_i = 0, \tag{113a}$$

$$i \in \mathsf{L} \implies \lambda_i = 0 \text{ and } v_i > 0. \tag{113b}$$

Whereas for frictional constraints with box limits, the free and tight labels imply that

$$\{i \in \mathsf{F} \implies -\mu\lambda_{\hat{n}} < \lambda_i < \mu\lambda_{\hat{n}} \text{ and } v_i = 0\}, \tag{114a}$$

$$\{i \in \mathsf{L} \implies \lambda_i = -\mu\lambda_{\hat{n}} \text{ and } v_i > 0\}, \tag{114b}$$

$$\{i \in \mathsf{U} \implies \lambda_i = +\mu\lambda_{\hat{n}} \text{ and } v_i < 0\}, \tag{114c}$$

where $\lambda_{\hat{n}}$ is the non-interpenetration impulse corresponding to frictional impulse λ_i.

Extending the conditions in Equations 112a-112c to the general case, we observe that the residual velocity is zero for all variables in the free set. That is,

$$\forall i \in \mathsf{F} : v_i = 0.$$

[1]The variables our contact models are mostly limited by inequality conditions. So, "tight" describes a variable that is directly against the hyperplane defining an inequality constraint, whereas "free" variables are not and they have some room on all sides.

Furthermore, we note that the value of tight variables is determined by a boundary condition (i.e., they are known). This allows us to simplify Equation 111 to a version that involves only the unknown free variables:

$$\mathbf{A}_{\mathsf{F},\mathsf{F}}\mathbf{x}_{\mathsf{F}} = -\mathbf{b}_{\mathsf{F}} - \mathbf{A}_{\mathsf{F},\mathsf{L}}\mathbf{x}_{\mathsf{L}}^{\text{lo}} - \mathbf{A}_{\mathsf{F},\mathsf{U}}\mathbf{x}_{\mathsf{U}}^{\text{hi}} \tag{115}$$

Hence, computing $\mathbf{x}_{\mathsf{F}}$ simply requires solving an unconstrained linear system. The objective of pivoting methods is therefore to correctly determine the index set of constraint variables that solves the complementarity problem. Essentially, a solution is found by "guessing" an index set for the constraint variables and then confirming that a candidate solution is correct with respect to the complementarity and feasibility conditions. We note that a brute force approach for determining the correct index set would be inefficient since it requires guessing 2^m possibilities for an m variables system. Rather, most pivoting methods use a systematic approach to change the index set of individual variables. In the next sections, we will explore some popular algorithms that use various strategies to pivot variables between the free and tight sets to find a solution.

3.1.1 Incremental Pivoting. Lemke's algorithm [Lloyd 2005] is a well-known pivoting method that incrementally pivots variables between the tight and free set in order to find a basis that solves the LCP. The simplex method proposed by Cottle and Dantzig [Cottle 1968] has likewise been used to find the index set. Baraff [Baraff 1994] used an incremental pivoting approach based on Dantzig's algorithm to solve for contact forces. Their method incrementally computes a solution by making change to the index sets while keeping the complementarity constraints as invariants. We consider the case of solving a traditional LCP with the index sets following Equation 113.

The algorithm begins with sets F and L empty; set U is ignored. It is presumed that, at each pivoting step, the complementarity conditions of all previously processed variables are maintained. The algorithm proceeds at each step by selecting the candidate variable with index j from the unprocessed set P that minimizes $v_j \in \mathbb{R}$, since this corresponds to an index with the most violated complementarity condition. At the k^{th} iteration, the following partitioning of the system is used:

$$\begin{bmatrix} \mathbf{v}_{\mathsf{F}}^k \\ \mathbf{v}_{\mathsf{L}}^k \\ v_j^k \end{bmatrix} = \begin{bmatrix} \mathbf{A}_{\mathsf{F},\mathsf{F}} & \mathbf{A}_{\mathsf{F},\mathsf{L}} & \mathbf{A}_{\mathsf{F},j} \\ \mathbf{A}_{\mathsf{F},\mathsf{L}}^T & \mathbf{A}_{\mathsf{L},\mathsf{L}} & \mathbf{A}_{\mathsf{L},j} \\ \mathbf{A}_{\mathsf{F},j}^T & \mathbf{A}_{\mathsf{L},j}^T & \mathbf{A}_{j,j} \end{bmatrix} \begin{bmatrix} \mathbf{x}_{\mathsf{F}}^k \\ \mathbf{x}_{\mathsf{L}}^k \\ 0 \end{bmatrix} + \begin{bmatrix} \mathbf{b}_{\mathsf{F}} \\ \mathbf{b}_{\mathsf{L}} \\ b_j \end{bmatrix}.$$

Initially, it is assumed that $x_j = 0$. Knowing that $\mathbf{x}^k$ of the processed indices respects the complementarity conditions, we focus on computing an update of the variables based on a

change of the unprocessed variable Δx_j, such that

$$\begin{bmatrix} \Delta \mathbf{v}_\mathsf{F} \\ \Delta \mathbf{v}_\mathsf{L} \\ \Delta v_j \end{bmatrix} = \begin{bmatrix} \mathbf{A}_{\mathsf{F},\mathsf{F}} & \mathbf{A}_{\mathsf{F},\mathsf{L}} & \mathbf{A}_{\mathsf{F},j} \\ \mathbf{A}_{\mathsf{F},\mathsf{L}}^T & \mathbf{A}_{\mathsf{L},\mathsf{L}} & \mathbf{A}_{\mathsf{L},j} \\ \mathbf{A}_{\mathsf{F},j}^T & \mathbf{A}_{\mathsf{L},j}^T & \mathbf{A}_{j,j} \end{bmatrix} \begin{bmatrix} \Delta \mathbf{x}_\mathsf{F} \\ \Delta \mathbf{x}_\mathsf{L} \\ \Delta x_j \end{bmatrix} .$$

The method takes advantage of the invariants $\mathbf{v}_\mathsf{F} = 0$ and $\mathbf{x}_\mathsf{L} = 0$, which logically leads to the assumption that $\Delta \mathbf{v}_\mathsf{F} = 0$ and $\Delta \mathbf{x}_\mathsf{L} = 0$. For a unit increase $\Delta x_j = 1$, we can write the linear system as:

$$\begin{bmatrix} \mathbf{0} \\ \Delta \mathbf{v}_\mathsf{L} \\ \Delta v_j \end{bmatrix} = \begin{bmatrix} \mathbf{A}_{\mathsf{F},\mathsf{F}} & \mathbf{A}_{\mathsf{F},\mathsf{L}} & \mathbf{A}_{\mathsf{F},j} \\ \mathbf{A}_{\mathsf{F},\mathsf{L}}^T & \mathbf{A}_{\mathsf{L},\mathsf{L}} & \mathbf{A}_{\mathsf{L},j} \\ \mathbf{A}_{\mathsf{F},j}^T & \mathbf{A}_{\mathsf{L},j}^T & \mathbf{A}_{j,j} \end{bmatrix} \begin{bmatrix} \Delta \mathbf{x}_\mathsf{F} \\ \mathbf{0} \\ 1 \end{bmatrix} . \tag{116}$$

The first block row of Equation 116 allows us to compute changes in the free variables by solving

$$\mathbf{A}_{\mathsf{F},\mathsf{F}} \Delta \mathbf{x}_\mathsf{F} = -\mathbf{A}_{\mathsf{F},j} .$$

The change in residual velocity of tight variables $\mathbf{v}_\mathsf{L}$ is then given by

$$\Delta \mathbf{v}_\mathsf{L} = \mathbf{A}_{\mathsf{F},\mathsf{L}}^T \Delta \mathbf{x}_\mathsf{F} + \mathbf{A}_{\mathsf{L},j} ,$$

and likewise

$$\Delta v_j = \mathbf{A}_{\mathsf{F},j}^T \Delta \mathbf{x}_\mathsf{F} + \mathbf{A}_{j,j} .$$

However, we note that our assumptions are valid only if the index set of variables does not change. For some $i \in \mathsf{F}$ where $\Delta x_i < 0$, it is possible that x_i will be driven to zero, in which case i should be pivoted from $\mathsf{F} \rightarrow \mathsf{L}$. Similarly, for some $i \in \mathsf{L}$ where $\Delta v_i < 0$, it is possible the residual velocity v_i is driven to zero, in which case the variable i should be pivoted from $\mathsf{L} \rightarrow \mathsf{F}$.

Essentially, the idea here is to increase x_j^{k+1} as much as possible without breaking the complementarity or feasibility conditions. At each iteration, it is therefore necessary to compute a maximum step size α up to the point that the index set of one of the variables changes (pivots).

Starting with the unprocessed variable j, the largest step we can take without forcing x_j^{k+1} negative is

$$\alpha_j = \left(\frac{-v_j^k}{\Delta v_j} \right) . \tag{117}$$

For each $i \in \mathsf{F}$, if $\Delta x_i < 0$, the minimum step size that can be taken without forcing x_i^{k+1} negative is

$$\alpha_\mathsf{F} = \min_{i \in \mathsf{F} \wedge \Delta x_i < 0} \left(\frac{-x_i^k}{\Delta x_i} \right) . \tag{118}$$

Finally, for each $i \in \mathsf{L}$, if $\Delta v_i < 0$, the minimum step size that can be taken without forcing v_i^{k+1} negative is

$$\alpha_{\mathsf{L}} = \min_{i \in \mathsf{L} \wedge \Delta v_i < 0} \left(\frac{-v_i^k}{\Delta v_i} \right). \tag{119}$$

The step size is computed as $\alpha = \min\left(\alpha_{\mathsf{F}}, \alpha_{\mathsf{L}}, \alpha_j\right)$. The constraint impulses are then increased by $\alpha \Delta \mathbf{x} = \alpha \begin{bmatrix} \Delta \mathbf{x}_{\mathsf{F}} & \mathbf{0} & 1 \end{bmatrix}^T$, which causes v to change by $\alpha \Delta v = \mathbf{A}(\alpha \Delta \mathbf{x})$. If the blocking constraint index is not j, this causes a change in the index set and the variable is pivoted to maintain the complementarity conditions. The process of updating the constraint variables and increasing Δx_j continues until v_j is driven to zero. Pseudo-code for the incremental pivoting algorithm is provided in Algorithm 3.

3.1.2 Principal Pivoting Methods. Unlike Baraff's incremental pivoting approach, other pivoting algorithms initialize variables with a specific label. The Bard-type method proposed by Murty [Murty 1974; Murty and Yu 1988] is one such algorithm, and it uses principal pivots at each iteration to change the index set whenever a variable at index i violates the feasibility or complementarity conditions.

The algorithm begins with an initial "guess" for the index sets, and at each step the principal sub-problem in Equation 115 is solved in order to determine $\mathbf{x}_{\mathsf{F}}$, and thus $\mathbf{v}_{\mathsf{L}}$ and $\mathbf{v}_{\mathsf{U}}$ too. If a variable violates its feasibility or complimentarity conditions, then the algorithm stops since the current index set is a solution to the LCP. Otherwise, Murty's algorithm chooses an infeasible variable with the smallest index i and pivots it into the complementary set. That is, variables are pivoted according to the rules:

$$\begin{cases} \mathsf{F} \leftarrow \mathsf{F} - \{i\} \quad \mathsf{L} = \mathsf{L} \cup \{i\}, & \text{if } i \in \mathsf{F} \text{ and } x_i \leq x_i^{\text{lo}} \\ \mathsf{F} \leftarrow \mathsf{F} - \{i\} \quad \mathsf{U} = \mathsf{U} \cup \{i\}, & \text{if } i \in \mathsf{F} \text{ and } x_i \geq x_i^{\text{hi}} \\ \mathsf{U} = \mathsf{U} - \{i\} \quad \mathsf{F} \leftarrow \mathsf{F} \cup \{i\}, & \text{if } i \in \mathsf{U} \text{ and } v_i \geq 0 \\ \mathsf{L} = \mathsf{L} - \{i\} \quad \mathsf{F} \leftarrow \mathsf{F} \cup \{i\}, & \text{if } i \in \mathsf{L} \text{ and } v_i \leq 0 \end{cases}. \tag{120}$$

It can be proven that if the lead matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is a P-matrix then the system will find a solution in n pivots. However, for degenerate systems, it can be shown that the pivoting algorithm may lead to cycling, which occurs whenever the same index set is encountered twice by the pivoting algorithm. Robust implementations of principal pivoting algorithms often require some way to detect and break cycles.

3.1.3 Block Principal Pivoting. Both the incremental pivoting and the Bard-type algorithm proceed at each step by updates that are based on a single variable. While single pivoting schemes have good convergence guarantees, computation times are protracted due to only changing a single variable at each step. However, in practice, it is often possible to pivot more than one variable to achieve faster convergence.

Data: $\mathbf{A}$: The LCP coefficient matrix, $\mathbf{b}$: The right hand side vector.
Result: $\mathbf{x}$: The LCP solution, F,L: The index sets of the free and tight variables.

$\mathbf{x} = \mathbf{0}$
$\mathbf{v} = \mathbf{b}$
initialize $\mathsf{F} = \mathsf{L} = \emptyset$
while $\exists j$ such that $v_j < 0$ **do**
  repeat
    solve $\mathbf{A}_{\mathsf{F},\mathsf{F}} \Delta \mathbf{x}_{\mathsf{F}} = -\mathbf{A}_{\mathsf{F},j}$
    $\Delta \mathbf{v}_{\mathsf{L}} = \mathbf{A}^T_{\mathsf{F},\mathsf{L}} \Delta \mathbf{x}_{\mathsf{F}} + \mathbf{A}_{\mathsf{L},j}$
    $\Delta v_j = \mathbf{A}^T_{\mathsf{F},j} \Delta \mathbf{x}_{\mathsf{F}} + \mathbf{A}_{j,j}$
    compute α according to Equation 117-Equation 119
    store blocking constraint index l
    $\mathbf{x}_{\mathsf{F}} \leftarrow \mathbf{x}_{\mathsf{F}} + \alpha \Delta \mathbf{x}_{\mathsf{F}}$
    $\mathbf{v}_{\mathsf{L}} \leftarrow \mathbf{v}_{\mathsf{L}} + \alpha \Delta \mathbf{v}_{\mathsf{L}}$
    $\lambda_j = \alpha$
    if $l \in \mathsf{F}$ **then**
      $\mathsf{F} \leftarrow \mathsf{F} - \{l\}$
      $\mathsf{L} \leftarrow \mathsf{L} \cup \{l\}$
    else if $l \in \mathsf{L}$ **then**
      $\mathsf{L} \leftarrow \mathsf{L} - \{l\}$
      $\mathsf{F} \leftarrow \mathsf{F} \cup \{l\}$
    else
      $\mathsf{F} \leftarrow \mathsf{F} \cup \{l\}$
  until $l = j$;
end

Algorithm 3: Incremental pivoting for an LCP.

Judice and Pires [Júdice and Pires 1994] proposed a block version of Murty's algorithm whereby all variables in violation of the feasibility or complementarity conditions are pivoted together at each step, rather than just the least indexed variable. Therefore, at each algorithm step, a pass over all variables check if any violate the condition, and if so the pivoting rules in Equation 120 are applied to all variables in violation simultaneously. The algorithm terminates with success if the index sets between two consecutive pivoting steps do not change. Pseudocode for the block principal pivoting algorithm is provided in Algorithm 4.

In practice, the BPP algorithm can converge to the correct index set after only a small number of iterations. However, cycling in the index set can occur even when $\mathbf{A}$ is positive definite. In such cases, one strategy proposed by Judice and Pires [Júdice and Pires 1994] is to temporarily

Data: N: The total number of pivoting steps to perform, $\mathbf{A}$: The coefficient matrix, $\mathbf{b}$: The right hand side vector.
Result: $\mathbf{x}$: The BLCP solution for all n variables, F,L,U: The index sets of the free and tight variables.

initialize $\mathsf{F} = \{1, \ldots, n\}, \mathsf{L} = \emptyset, \mathsf{U} = \emptyset$
$k = 1$
while ($k \leq N$ and F, L, U changed) **do**
  $\forall\, i \in \mathsf{L} \;:\; x_i \leftarrow x_i^{\text{lo}}$
  $\forall\, i \in \mathsf{U} \;:\; x_i \leftarrow x_i^{\text{hi}}$
  solve $\mathbf{A}_{\mathsf{F},\mathsf{F}}\mathbf{x}_{\mathsf{F}} = -\mathbf{b}_{\mathsf{F}} - \mathbf{A}_{\mathsf{F},\mathsf{L}}\mathbf{x}_{\mathsf{L}}^{\text{lo}} - \mathbf{A}_{\mathsf{F},\mathsf{U}}\mathbf{x}_{\mathsf{U}}^{\text{hi}}$
  $\mathbf{v} = \mathbf{A}\mathbf{x} + \mathbf{b}$
  update F, L, U according to Equation 120
  $k = k + 1$
end

Algorithm 4: Block principal pivoting (BPP) for BLCPs.

revert to a Murty single pivoting scheme by choosing an infeasible variable with the least index. However, in general, robust implementations of pivoting algorithm require some sort of cycle breaking strategy.

3.1.4 Implementation Notes. Observe that both the incremental pivoting and the block principal pivoting methods require a solution involving the linear system $\mathbf{A}_{\mathsf{F},\mathsf{F}}\mathbf{x}_{\mathsf{F}}$. We assume that $\mathbf{A}$ is symmetric positive definite (PD), and therefore $\mathbf{A}_{\mathsf{FF}}^{-1}$ exists. However, Baraff reported that even when $\mathbf{A}$ is positive semi-definite (PSD), which is often the case in practice due to redundant contact constraints, then $\mathbf{A}_{\mathsf{F},\mathsf{F}}^{-1}$ could still be computed. Applying the constraint stabilization techniques presented in Section 1.9 can improve the conditioning of $\mathbf{A}_{\mathsf{F},\mathsf{F}}$ and alleviate problems related to degenerate matrices.

In practice, forming and solving the linear system uses a Cholesky decomposition. Baraff noted that for complex systems, an incremental factorization is preferred for efficiency, although an implementation is potentially non-trivial. Enzenhoefer et al. [2019] proposed an efficient version of Judice's block pivoting algorithm that re-uses an initial factorization of $\mathbf{A}_{\mathsf{F},\mathsf{F}}\mathbf{x}_{\mathsf{F}}$ and downdates the factorization based on indices in $\mathsf{L} \cup \mathsf{U}$. For complex simulations, they observed a 1.5 to 3 times reduction in the computation time of the BPP algorithm when using a downdated factorization, with the caveat that the approach is more efficient only when a nominal percentage of variables is pivoted. Furthermore, parallelization of pivoting methods is challenging, especially if a factorization is used to solve the principal sub-problem. However, there is some work that has addressed this problem. For instance, Peiret et al. [2019] introduced

a parallelization technique for pivoting solvers using a domain decomposition technique, and their method is capable of producing exact solutions for the contact BLCP.

3.2 Fixed-point methods

In this section we will focus on a popular class of methods for solving contact problems that are commonly referred to as fixed-point methods. Specifically, we focus on the projected Gauss-Seidel (PGS) algorithm and its variants. Furthermore, this section focuses entirely on contact models that originate from the complementary problem formulations introduced in Sections 1.6-1.8. First, we introduce the idea of splitting up the contact problem in order to get a group of more manageable problems, which we can then solve iteratively. This naturally leads to a frictional contact formulation that can be solved using the PGS method. Then, in Section 3.2.4, the idea of splitting in a recursive fashion is used to derive the blocked Gauss-Seidel method. The staggering method is presented in Section 3.2.5. Blocking and staggering are general concepts that can be applied in other problem domains, as they serve as a general scheme for combining different numerical solvers into one overall solver. The PGS type of methods are interesting building blocks for many other types of numerical methods such as sub-space minimization [Silcowitz et al. 2010b] and non-smooth non-linear conjugate gradient (NNCG) method [Silcowitz et al. 2010a]. We cover these higher-level solvers once we have created a strong foundation for the PGS type of methods.

3.2.1 Splitting Methods. We begin with the general concept of splitting methods. This class of methods is well known in numerical optimization, and includes methods such as Gauss-Seidel, Jacobi and successive over-relaxation (SOR). Splitting methods are iterative methods, which unlike the pivoting methods described in Section 3.1, are practically incapable of computing the exact solution to the LCP. Iterative methods approximate the solution, but do so in a much more efficient way, both computationally and storage wise. The traditional form of the LCP is

$$\mathbf{A}\mathbf{x} + \mathbf{b} \geq \mathbf{0}\,, \tag{121a}$$

$$\mathbf{x} \geq \mathbf{0}\,, \tag{121b}$$

$$\mathbf{x}^T\left(\mathbf{A}\mathbf{x} + \mathbf{b}\right) = 0\,, \tag{121c}$$

which corresponds to the LCP introduced using the polyhedral cone in Section 1.6. Splitting methods start with the decomposition of the lead matrix such that $\mathbf{A} = \mathbf{M} - \mathbf{N}$. Note that $\mathbf{M}$ here is not the mass matrix, and we leave $\mathbf{M}$ and $\mathbf{N}$ unspecified for now. Using the new definition of $\mathbf{A}$, Equation 121 becomes

$$\mathbf{M}\mathbf{x} - \mathbf{N}\mathbf{x} + \mathbf{b} \geq \mathbf{0}\,, \tag{122a}$$

$$\mathbf{x} \geq \mathbf{0}\,, \tag{122b}$$

$$\mathbf{x}^T(\mathbf{M}\mathbf{x} - \mathbf{N}\mathbf{x} + \mathbf{b}) = 0\,. \tag{122c}$$

Let us next assume that we will use an iterative method where $\mathbf{x}^k \longrightarrow \mathbf{x}^{k+1}$ for $k \longrightarrow \infty$. This assumes there is an accumulation or limiting point for the system. We can then rewrite Equation 121 using the decomposition and in terms of the current and next iterate:

$$\mathbf{M}\mathbf{x}^{k+1} - \mathbf{N}\mathbf{x}^k + \mathbf{b} \geq \mathbf{0}\,, \tag{123a}$$

$$\mathbf{x}^{k+1} \geq \mathbf{0}\,, \tag{123b}$$

$$\left(\mathbf{x}^{k+1}\right)^T \left(\mathbf{M}\mathbf{x}^{k+1} - \mathbf{N}\mathbf{x}^k + \mathbf{b}\right) = 0\ . \tag{123c}$$

In the k^{th} iteration of the iterative method, we let $\mathbf{c}^k = \mathbf{b} - \mathbf{N}\mathbf{x}^k$, and the LCP Equation 123 becomes

$$\mathbf{M}\mathbf{x}^{k+1} + \mathbf{c}^k \geq \mathbf{0}\,, \tag{124a}$$

$$\mathbf{x}^{k+1} \geq \mathbf{0}\,, \tag{124b}$$

$$(\mathbf{x}^{k+1})^T(\mathbf{M}\mathbf{x}^{k+1} + \mathbf{c}^k) = 0\,. \tag{124c}$$

This is a fixed-point formulation and, for a suitable choice of $\mathbf{M}$ and $\mathbf{N}$, we hope that the complementarity sub-problem Equation 124 might be easier to solve than the original problem Equation 121. Imagine, for instance, letting $\mathbf{M}$ be the diagonal of $\mathbf{A}$. This choice decouples all variables and we have a sub-problem of n independent 1D LCPs. The general splitting method can be summarized as follows:

Step 1 Initialization: set $k = 0$ and choose an arbitrary non-negative $\mathbf{x}^0 \geq \mathbf{0}$.
Step 2 Given $\mathbf{x}^k \geq \mathbf{0}$ solve the LCP Equation 124.
Step 3 If $\mathbf{x}^{k+1}$ satisfies the termination criteria then stop, otherwise set $k \leftarrow k + 1$ and go to Step 2.

The splitting is often chosen such that $\mathbf{M}$ is a Q-matrix, meaning that $\mathbf{M}$ belongs to the class of matrices where the corresponding LCP has a solution. Clearly, if $\mathbf{x}^{k+1}$ is a solution to Equation 123 and we have $\mathbf{x}^{k+1} = \mathbf{x}^k$, then by substitution into the sub-problem given by Equation 123 we see that $\mathbf{x}^{k+1}$ is a solution to the original problem Equation 121.

We use the minimum map reformulation to realize the LCP sub-problem in Equation 123c, such that

$$\min(\mathbf{x}^{k+1}, \mathbf{M}\mathbf{x}^{k+1} + \mathbf{b} - \mathbf{N}\mathbf{x}^k) = \mathbf{0}\,, \tag{125}$$

and subtracting $\mathbf{x}^{k+1}$ and multiply by minus one, we get

$$\max(\mathbf{0}, -\mathbf{M}\mathbf{x}^{k+1} - \mathbf{b} + \mathbf{N}\mathbf{x}^k + \mathbf{x}^{k+1}) = \mathbf{x}^{k+1}\,. \tag{126}$$

Here, we re-discover a fixed-point formulation. Now, let us perform a case-by-case analysis of the i^{th} component. If we assume

$$\left(\mathbf{x}^{k+1} - \mathbf{M}\,\mathbf{x}^{k+1} - \mathbf{b} + \mathbf{N}\mathbf{x}^k\right)_i < 0\,, \tag{127}$$

Data: N: Number of sweeps to perform, $\mathbf{x}$: Initial iterate, $\mathbf{M}$: First part of the matrix splitting, $\mathbf{N}$: Second part of the matrix splitting, $\mathbf{b}$: The LCP right hand side vector.
Result: $\mathbf{x}$: The numerical solution of the LCP.

```
for k ← 1 to N do
    z^k ← M^{-1} (N x^k + b);
    x^{k+1} ← max (0, z^k);
end
```

Algorithm 5: Splitting method. Generic projected method given matrix splitting $\mathbf{A} = \mathbf{M} - \mathbf{N}$. Notice that the key to an efficient method is to find an in-expensive to compute the effect of the matrix multiplication by $\mathbf{M}^{-1}$.

then we must have $\mathbf{x}_i^{k+1} = 0$. Otherwise, our assumption is false and we must have

$$\left(\mathbf{x}^{k+1} - \mathbf{M}\,\mathbf{x}^{k+1} - \mathbf{b} + \mathbf{N}\mathbf{x}^{k}\right)_i = \mathbf{x}_i^{k+1}\,. \tag{128}$$

That is,

$$(\mathbf{M}\,\mathbf{x}^{k+1})_i = -\mathbf{c}_i^k. \tag{129}$$

Here we have defined $\mathbf{c}^k = \mathbf{b} - \mathbf{N}\mathbf{x}^k$. For a suitable choice of $\mathbf{M}$ we can rewrite $(\mathbf{M}\mathbf{x}^{k+1})_i = -\mathbf{c}_i^k$ as

$$\mathbf{x}_i^{k+1} = -(\mathbf{M}^{-1}\mathbf{c}^k)_i. \tag{130}$$

Not all splittings will make this rewrite possible. A trivial example that allows this is to let $\mathbf{M}$ to be the diagonal of $\mathbf{A}$. Other common choices that allow this rewrite are listed in Table 3. A simple rearrangement of terms gives an expression to update $\mathbf{x}$ at each iterate, such that

$$\left(\mathbf{M}^{-1}\left(\mathbf{N}\mathbf{x}^{k} - \mathbf{b}\right)\right)_i = \mathbf{x}_i^{k+1}\,, \tag{131}$$

and combining it all we have derived the closed form solution for the complementary sub-problem:

$$\max\left(\mathbf{0}, \left(\mathbf{M}^{-1}\left(\mathbf{N}\mathbf{x}^{k} - \mathbf{b}\right)\right)\right) = \mathbf{x}^{k+1}\,. \tag{132}$$

Iterative schemes like these are often termed *projection methods*. The reason for this is that if we introduce the vector $\mathbf{z}^k = \mathbf{M}^{-1}\left(\mathbf{N}\mathbf{x}^k - \mathbf{b}\right)$ then

$$\mathbf{x}^{k+1} = \max\left(\mathbf{0}, \mathbf{z}^k\right)\,. \tag{133}$$

That is, the $k+1$ iterate is obtained by projecting the vector $\mathbf{z}^k$ to be positive. The pseudo-code for splitting methods with projection is given in Algorithm 5 and this framework applies to a variety of algorithms which are distinguished according to their splitting. Table 3 lists some of the popular splittings used by specific variants of the algorithm.

Table 3. Splittings for the three methods: Jacobi, projected Gauss-Seidel (PGS) and projected successive over-relaxation (PSOR). The matrix $\mathbf{D}$ is the diagonal part of the original $\mathbf{A}$ matrix. The matrix $\mathbf{U}$ is the strict upper part of the original $\mathbf{A}$ matrix. The matrix $\mathbf{L}$ is the strict lower part of the original $\mathbf{A}$ matrix. The relaxation parameter should be chosen such that $0 < \omega < 2$.

Method	$\mathbf{M}$	$-\mathbf{N}$
Jacobi	$\mathbf{D}$	$\mathbf{L}+\mathbf{U}$
PGS	$\mathbf{L}+\mathbf{D}$	$\mathbf{U}$
PSOR	$\mathbf{D}+\omega\mathbf{L}$	$(1-\omega)\,\mathbf{D}-\omega\mathbf{U}$

It should be noted that $\mathbf{A}$ must at least have nonzero diagonal values for these splittings to work. As far as we know there exist no convergence proofs in the general case of $\mathbf{A}$ being arbitrary. However, given appropriate assumptions on $\mathbf{A}$ such as being a contraction mapping or symmetric, global convergence can be proven [Cottle et al. 1992; Murty and Yu 1988]. The projected Successive Over Relaxation (PSOR) method is interesting as it has the free parameter ω that helps control the convergence rate. Further, we see that the projected Gauss–Seidel (PGS) method pops out as the special case for $\omega = 1$. In practice one would tune the value of ω.

To see how to solve Equation 133 in an actual implementation, let us look at the specific case of PSOR splitting as given in the Table 3. The principle is a for-loop which sweeps over the vector components and updates the $\mathbf{x}$ vector in place. When we use the PSOR splitting, it is useful to note that

$$\begin{aligned}\mathbf{M}-\mathbf{N} &= \mathbf{D}+\omega\mathbf{L}-((1-\omega)\mathbf{D}-\omega\mathbf{U})\,, &(134)\\ &= \omega(\mathbf{L}+\mathbf{D}+\mathbf{U})\,, &(135)\\ &= \omega\mathbf{A}\,. &(136)\end{aligned}$$

This is equivalent to a relaxation of the LCP by replacing $\mathbf{A}\,\mathbf{x}+\mathbf{b} \geq \mathbf{0}$ with $\omega\,(\mathbf{A}\,\mathbf{x}+\mathbf{b}) \geq \mathbf{0}$. For now we assume ω is a positive real number. The relaxed LCP is then,

$$\begin{aligned}\omega\ (\mathbf{A}\mathbf{x}+\mathbf{b}) &\geq \mathbf{0}\,, &(137a)\\ \mathbf{x} &\geq \mathbf{0}\,, &(137b)\\ \mathbf{x}^T\omega\,(\mathbf{A}\mathbf{x}+\mathbf{b}) &= 0\,. &(137c)\end{aligned}$$

The factor ω can be interpreted as a scaling of the term $\mathbf{A}\,\mathbf{x}+\mathbf{b}$. We now rewind the derivation of the projection method to Equation 129. For the specific case of PSOR this will result in

$$\left((\omega\,\mathbf{L}+\mathbf{D})\ \mathbf{x}^{k+1}\right)_i = \left(((1-\omega)\,\mathbf{D}-\omega\mathbf{U})\ \mathbf{x}^k-\omega\mathbf{b}\right)_i \tag{138}$$

Next we replace matrix multiplications with index notation

$$\omega \sum_{j=1}^{n} \mathbf{L}_{ij}\mathbf{x}_j^{k+1} + \sum_{j=1}^{n} \mathbf{D}_{ij}\mathbf{x}_j^{k+1} = (1-\omega) \sum_{j=1}^{n} \mathbf{D}_{ij}\,\mathbf{x}_j^{k} - \omega \sum_{j=1}^{n} \mathbf{U}_{ij}\,\mathbf{x}_j^{k} - \omega \mathbf{b}_i\,. \tag{139}$$

We can use the fill pattern of the matrices $\mathbf{L}$, $\mathbf{D}$, and $\mathbf{U}$ to optimize the summation operations,

$$\omega \sum_{j=1}^{i-1} \mathbf{L}_{ij}\mathbf{x}_j^{k+1} + \mathbf{D}_{ii}\mathbf{x}_i^{k+1} = (1-\omega)\,\mathbf{D}_{ii}\,\mathbf{x}_i^{k} - \omega \sum_{j=i+1}^{n} \mathbf{U}_{ij}\,\mathbf{x}_j^{k} - \omega \mathbf{b}_i\,. \tag{140}$$

Then we isolate $\mathbf{x}_i^{k+1}$ on the left hand side of the equation

$$\mathbf{x}_i^{k+1} = \frac{-\omega \mathbf{b}_i - \omega \sum_{j=1}^{i-1} \mathbf{L}_{ij}\mathbf{x}_j^{k+1} - \omega \mathbf{D}_{ii}\,\mathbf{x}_i^{k} - \omega \sum_{j=i+1}^{n} \mathbf{U}_{ij}\,\mathbf{x}_j^{k} + \mathbf{D}_{ii}\,\mathbf{x}_i^{k}}{\mathbf{D}_{ii}}\,. \tag{141}$$

Reducing this slightly, we get

$$\mathbf{x}_i^{k+1} = \mathbf{x}_i^{k} - \omega \frac{\mathbf{b}_i + \sum_{j=1}^{i-1} \mathbf{L}_{ij}\mathbf{x}_j^{k+1} + \mathbf{D}_{ii}\,\mathbf{x}_i^{k} + \sum_{j=i+1}^{n} \mathbf{U}_{ij}\,\mathbf{x}_j^{k}}{\mathbf{D}_{ii}}\,. \tag{142}$$

We assume a sweep over the indices i^{th} in increasing order. Due to this particular sweep order we will have solved for all $\mathbf{x}_j^{k+1}$ with $j < i$ when we start updating the i^{th} index. Hence there are no unknowns on the right hand side of Equation 142. We can exploit this knowledge to allow in-place updating of the $\mathbf{x}$-iterate,

$$\mathbf{x}_i \leftarrow \mathbf{x}_i - \omega \frac{\mathbf{b}_i + \sum_{j=1}^{i-1} \mathbf{L}_{ij}\mathbf{x}_j + \mathbf{D}_{ii}\,\mathbf{x}_i + \sum_{j=i+1}^{n} \mathbf{U}_{ij}\,\mathbf{x}_j}{\mathbf{D}_{ii}}\,. \tag{143}$$

Noting that

$$\sum_{j=1}^{i-1} \mathbf{L}_{ij}\mathbf{x}_j + \mathbf{D}_{ii}\,\mathbf{x}_i + \sum_{j=i+1}^{n} \mathbf{U}_{ij}\,\mathbf{x}_j = \sum_{j=1}^{n} \mathbf{L}_{ij}\mathbf{x}_j + \sum_{j=1}^{n} \mathbf{D}_{ii}\,\mathbf{x}_j + \sum_{j=1}^{n} \mathbf{U}_{ij}\,\mathbf{x}_j \tag{144}$$

$$= \sum_{j=1}^{n} \mathbf{A}_{ij}\mathbf{x}_j\,, \tag{145}$$

and defining $\mathbf{r} = \mathbf{b} + \mathbf{A}\mathbf{x}$ our derivation is reduced to

$$\mathbf{x}_i \leftarrow \mathbf{x}_i - \omega \frac{\mathbf{r}_i}{\mathbf{D}_{ii}}\,. \tag{146}$$

All that remains is to substitute the expression into the re-written minimal map reformulation from Equation 133. We obtain the final update scheme,

$$\mathbf{x}_i \leftarrow \max\left(0\,, \mathbf{x}_i - \omega \frac{\mathbf{r}_i}{\mathbf{D}_{ii}}\right) \quad \text{for } i = 1 \text{ to } n\,. \tag{147}$$

Data: N: The total number of sweeps to perform, ω: SOR relaxation parameter value, $\mathbf{x}$: Initial iterative, $\mathbf{A}$: The LCP coefficient matrix, $\mathbf{b}$: The right hand side vector.
Result: $\mathbf{x}$: The LCP numerical solution.

```
for k ← 1 to N do
    foreach i do
        r_i ← A_{i*} x + b_i;
        x_i ← max(0, x_i − ω r_i / A_ii);
    end
end
```

Algorithm 6: Projected successive over-relaxation (PSOR). Observe that this method essential builds on sparse blocked matrix products. This can be exploited in actual implementations.

Notice that the sweep order of the i^{th} index is important. A pseudo code version of this practical approach can be found in Algorithm 6. Note that PGS is a special case of where $\omega = 1$.

On a side note, observe that if the reverse sweep order is wanted such that $i = n$ to 1, then this is possible too. The original fixed-point formulation Equation 123 will then be

$$\mathbf{M}\mathbf{x}^{k} - \mathbf{N}\mathbf{x}^{k+1} + \omega\,\mathbf{b} \geq \mathbf{0}\,, \tag{148a}$$

$$\mathbf{x}^{k+1} \geq \mathbf{0}\,, \tag{148b}$$

$$\left(\mathbf{x}^{k+1}\right)^{T} \omega\left(\mathbf{M}\mathbf{x}^{k} - \mathbf{N}\mathbf{x}^{k+1} + \mathbf{b}\right) = 0\,. \tag{148c}$$

All steps in the derivations are similar for this version of the fixed-point problem. However, instead of forward PSOR update rule Equation 147 we now have a backward update rule

$$\mathbf{x}_i \leftarrow \max\left(0\,, \mathbf{x}_i - \omega\frac{\mathbf{r}_i}{\mathbf{D}_{ii}}\right) \quad \text{for } i = n \text{ to } 1\,. \tag{149}$$

In some applications, it can be convenient to let a full forward sweep be followed by a full backward sweep. This variant of the projection method is known as a symmetric projection method. The idea of symmetry applies to both PSOR and PGS, but is pointless for Jacobi. We note that in some applications the sweep order may cause side-effects and the symmetric variant can potentially alleviate this order-dependency issue to some extent.

In the splitting algorithms derived so far, we have applied a simple maximum iteration count to guard against infinite looping. For many real-time applications such a termination criteria will be sufficient. However, for more accurate applications it could result in inaccurate results.

It is quite easy to extend the algorithm to use merit functions for both absolute and relative convergence testing or detecting divergence. Convergence of projection methods requires that the update rule – such as Equation 147 – is a contraction mapping. As an example, we can use

the infinity norm of $\mathbf{x}$ as a merit function. The infinity norm merit function is defined as

$$\theta_{\infty}(\mathbf{x}) \equiv \|\mathbf{x}\|_{\infty} = \arg\max_{i} |\mathbf{x}_i| . \tag{150}$$

This particular norm is very attractive, as it is easily calculated during the sweeping of the indices, shown in Algorithm 7.

Data: N: The total number of sweeps to perform, $\mathbf{x}$: Initial solution estimate, $\mathbf{A}$: The coefficient matrix, $\mathbf{b}$: The right hand side vector, $\varepsilon_{\text{relative}} > 0$: relative convergence threshold.
Result: $\mathbf{x}$: The LCP numerical solution.

```
δ ← ∞;                                  /* Merit-value of current iterate */
γ ← ∞;                                  /* Merit-value of previous iterate */
for k ← 1 to N do
    γ ← δ;
    δ ← 0;
    foreach i do
        x_i ← update scheme;
        δ ← max(δ, x_i);
    end
    if δ > γ then
        return divergence
    end
    if (δ−γ)/γ < ε_relative then
        return relative convergence
    end
end
```

Algorithm 7: Projection method testing for contraction and divergence using infinity norm of iterate as merit function. Notice how divergence and relative convergence can be easily tested for.

Another possibility for convergence testing is to use a measurement for the complementary condition as a merit function:

$$\theta_{\text{compl.}}(\mathbf{x}) \equiv \left|\mathbf{x}^T (\mathbf{A}\mathbf{x} + \mathbf{b})\right| . \tag{151}$$

In the case of $\mathbf{x} = \mathbf{0}$ it is important to also make sure that the constraint $\mathbf{A}\mathbf{x} + \mathbf{b} \geq 0$ is not violated.

Any of the complementary reformulations can in principle be used as a merit function, but be careful not to choose a merit function which dominates the computational cost of the

iteration of the projection methods itself. The cost of a projection method iteration is $O(nk)$, where $k \leq n$ is the maximum number of non-zeros in any given row of $\mathbf{A}$.

3.2.2 Extending to the BLCP. As we saw in Section 1.7 the frictional problem could be recast from an LCP model into a boxed LCP model (BLCP). Hence, we will extend the splitting ideas from solving an LCP model to solving a BLCP model as well. Our starting point for deriving a splitting method for the BLCP is the minimum map reformulation. From this we can write the i^{th} component as follows

$$\min(u_i - \mathbf{x}_i, \max(l_i - \mathbf{x}_i, -\mathbf{y}_i)) = 0\,. \tag{152}$$

By adding $\mathbf{x}_i$ we get a fixed point formulation

$$\min(u_i, \max(l_i, \mathbf{x}_i - (\mathbf{A}\mathbf{x} + \mathbf{b})_i)) = \mathbf{x}_i\,. \tag{153}$$

What follows now, is quite similar to what we derived in the LCP case. Once again, we introduce the splitting $\mathbf{A} = \mathbf{M} - \mathbf{N}$ and the iteration index k. Then we define $\mathbf{c}^k = \mathbf{b} - \mathbf{N}\mathbf{x}^k$. Using this we have

$$\min(u_i, \max(l_i, (\mathbf{x}^{k+1} - \mathbf{M}\mathbf{x}^{k+1} - \mathbf{c}^k)_i)) = \mathbf{x}_i^{k+1}\,. \tag{154}$$

When $\mathbf{x}^k$ converges, then Equation 154 is equivalent to Equation 152. Next we perform a case-by-case analysis. Three cases are possible,

$$(\mathbf{x}^{k+1} - \mathbf{M}\mathbf{x}^{k+1} - \mathbf{c}^k)_i < l_i \Rightarrow \mathbf{x}_i^{k+1} = l_i\,, \tag{155a}$$

$$(\mathbf{x}^{k+1} - \mathbf{M}\mathbf{x}^{k+1} - \mathbf{c}^k)_i > u_i \Rightarrow \mathbf{x}_i^{k+1} = u_i\,, \tag{155b}$$

$$l_i \leq (\mathbf{x}^{k+1} - \mathbf{M}\mathbf{x}^{k+1} - \mathbf{c}^k)_i \leq u_i \Rightarrow\ \mathbf{x}_i^{k+1} = (\mathbf{x}^{k+1} - \mathbf{M}\mathbf{x}^{k+1} - \mathbf{c}^k)_i\,. \tag{155c}$$

Case Equation 155c reduces to,

$$(\mathbf{M}\mathbf{x}^{k+1})_i = -\mathbf{c}_i^k\,, \tag{156}$$

which for a suitable choice of $\mathbf{M}$ and back substitution of $\mathbf{c}^k$ gives,

$$\mathbf{x}_i^{k+1} = (\mathbf{M}^{-1}(\mathbf{N}\mathbf{x}^k - \mathbf{b}))_i\,. \tag{157}$$

Thus, our iterative splitting scheme becomes,

$$\min(u_i, \max(l_i, (\mathbf{M}^{-1}(\mathbf{N}\mathbf{x}^k - \mathbf{b}))_i)) = \mathbf{x}_i^{k+1}\,. \tag{158}$$

Now let $\mathbf{x}' = \mathbf{M}^{-1}(\mathbf{N}\mathbf{x}^k - \mathbf{b})$ then we have the projection method

$$\mathbf{x}^{k+1} = \min(\mathbf{u}, \max(\mathbf{l}, \mathbf{x}'))\,, \tag{159}$$

where the $(k+1)^{\text{th}}$ iterate obtained by projecting the vector $\mathbf{x}'$ onto the box given by $\mathbf{l}$ and $\mathbf{u}$. Valid splittings of $\mathbf{A}$ are the same as in Table 3. By comparing Equation 158 to Equation 133, we notice the the relationship between LCP and BLCP for splitting methods.

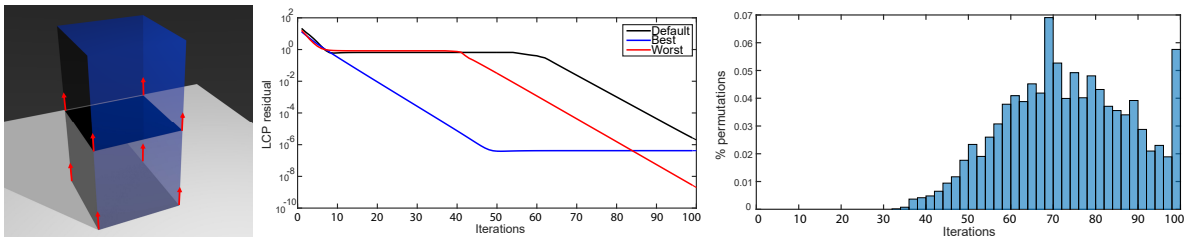


Fig. 16. Left: A stack of two boxes with 8 non-interpenetration constraints. Middle: The LCP residual versus the iteration count when solving non-interpenetration constraint of the two box stack example. The convergence of the best, worst, and default orderings are shown. Right: The distribution of iterations required to achieve 10^{-4} residual. From Andrews et al. [2017].

3.2.3 Ordering of Variables. The convergence of PGS and related fixed-point algorithms depends on the order in which variables $\mathbf{x}_{1\ldots n}$ are updated. This can easily be seen by inspecting Equation 143, where the update of variable $\mathbf{x}_i$ depends on the values of the previous $1 \ldots (i-1)$ variables and hence the order in which they are updated.

Fratarcangeli and Pellacini [2015] used an algorithm based on sequential vertex coloring to partition constraints of a particle simulation into a k-partite graph. They observed that the residual error in Gauss-Seidel iterations depends on the order in which the equations are solved, and that it can possibly introduce bias.

Andrews et al. [2017] evaluated convergence behavior of the PGS algorithm for simple contact problems, where all permutations for updating individual $\mathbf{x}_i$ values were evaluated. For example, with the two box stack shown in Figure 16, there are 8 non-interpenetration constraints, which gives 40320 permutations of the constraint variables. Their work shows that the best ordering can converge to a solution in less than half the iterations compared to the worst ordering. Furthermore, a wide range of convergence behavior is observed when evaluating the number of iterations that are required to achieve a certain accuracy.

In [Poulsen et al. 2010], a PGS variant of the splitting algorithm in Equation 158 is applied to a BLCP model. In this work, $\mathbf{l}$ and $\mathbf{u}$ are affine functions of $\mathbf{x}$. The algorithm framework we outlined can easily deal with this, simply by updating the $\mathbf{l}$ and $\mathbf{u}$ vectors whenever a change is made to $\mathbf{x}$. Figure 17 shows typical convergence plots from Poulsen et al. [2010] using different heuristics for permuting the order inside the PGS loop. Notice the non-monotone behavior of the greedy strategy, which strongly suggests the missing convergence guarantees of the BLCP contact model from Section 1.7.

3.2.4 The Blocked Gauss-Seidel Method. The matrix splitting approach we have used to derive the presented Gauss-Seidel methods, imply that they can not be used for the LCP contact model presented in Section 1.5 due to a zero diagonal values and nonsymmetry of $\mathbf{A}$. However, the splitting idea can be applied in a blocked version. This results in a numerical method that is very easy to implement and still preserves the good numerical properties of the PGS method.

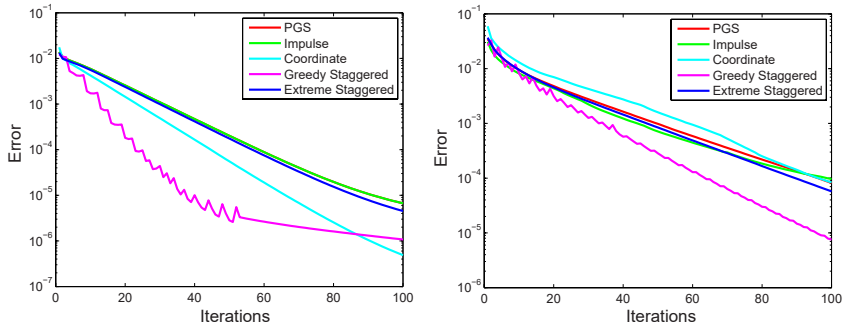


Fig. 17. Convergence rates taken from Poulsen et al. [2010]. On left is a box stack and on right a card house. Different heuristics are applied for permuting the order of variables inside the PGS loop. In this specific test example on left the Impulse curve is on top of the PGS curve. This specific study is for dense structured rigid body scenes, such as brick walls. Notice the non-monotone behavior.

To illustrate the blocking idea, we will – without loss of generalization – use the multiple contact problem as an example. Here, a block may be defined as all variables connected to a single contact point. When using a four-sided friction pyramid to model the friction constraints, the i^{th} block of $\mathbf{x}$ will consist of the normal impulse $\lambda_{\hat{n},i}$, four friction impulses $\lambda_{\hat{t}_1,i}$, $\lambda_{\hat{t}_2,i}$, $\lambda_{\hat{t}_3,i}$, $\lambda_{\hat{t}_4,i}$ and one slack variable β_i. The structure of the i^{th} block of $\mathbf{x}$ will then be defined as $\left[\mathbf{x}\right]_i = \begin{bmatrix}\lambda_{\hat{n},i} & \lambda_{\hat{t}_1,i} & \cdots & \beta_i\end{bmatrix}^T$, observe that we overload the square bracket notation to indicate blocks. Similarly, $\left[\mathbf{A}\right]_{ij}$ is the block of $\mathbf{A}$ corresponding to the i^{th} and j^{th} contact point variables. Thus, the blocked LCP can be written

$$\left[\mathbf{v}\right]_i = \sum_j \left[\mathbf{A}\right]_{ij} \left[\mathbf{x}\right]_j + \left[\mathbf{b}\right]_i \geq \mathbf{0} \quad \forall i\,, \tag{160a}$$

$$\left[\mathbf{x}\right]_i \geq \mathbf{0} \quad \forall i\,, \tag{160b}$$

$$\left[\mathbf{v}\right]_i^T \left[\mathbf{x}\right]_i = 0 \quad \forall i\,. \tag{160c}$$

In practice, the blocks can be defined however suits the problem or solver best.

We next apply the Gauss-Seidel splitting to the blocked LCP. The result is a *blocked Gauss-Seidel* (BGS) method (see Algorithm 8). The intuition behind this numerical method is that all contact point variables other than the i^{th} block are momentarily frozen while solving for the variables of the block. The BGS approach is also known as a *s*weeping process or as the non-smooth contact dynamics (NSCD) method [Jean 1999; Moreau 1999].

Data: N: Maximum number of sweeps/iterations, $\mathbf{x}$: Initial starting iterate, $\mathbf{A}$: The coefficient matrix of the LCP, $\mathbf{b}$: The right hand side vector of the LCP.
Result: $\mathbf{x}$: The solution for the LCP.

```
1 for k ← 1 to N do
2     foreach block i do
3         [b]'_i ← [b]_i − Σ_{j≠i} [A]_ij [x]_j;
4         solve sub-LCP ([A]_ii, [x]_i, [b]'_i) ;
5     end
6 end
```

Algorithm 8: Blocked Gauss-Seidel (BGS) method. Notice that the blocking is a very general mechanism. One may define blocks in any way and dimensions that one sees fit. Another trait from blocking is that any kind of sub-solver can be used for solving the blocked sub-problem.

The sub-block LCP at line 4 can be solved using any LCP solver. Usually yet another splitting is applied, dividing the sub-block LCP into a normal impulse sub-block and a frictional sub-block. The normal sub-block is a 1D problem and can be solved by a projection. For instance, when using the four-sided friction pyramid, the frictional sub-block is a 5D problem. The frictional sub-block of $\mathbf{A}$ is not entirely neat as we have zero diagonal terms and non-symmetry. However, the low dimensionality allows efficient direct enumeration. Another fix is to ignore the principle of maximum dissipation – effectively changing the contact model – which reduces the number of variables such that we have to solve a 2D problem with a symmetric PSD frictional sub-block matrix.

The blocked Gauss-Seidel method offers many possibilities. In Section 3.2.5 we divide a LCP into two sub-blocks, one with all normal variables only and the other containing the rests. If the LCP includes joints, we could have a sub-block for all the joint variables. As we will illustrate here, being clever in making such partitions can be used to solve sub-problems efficiently.

One such example is the joint sub-block of the LCP, this is known to be equivalent to a symmetric positive semi-definite linear system. Thus, we can use a preconditioned conjugate gradient (PCG) solver to solve for joint impulses rather than a PGS method. Because PCG has the same per-iteration cost as PGS, but a better convergence rate, the result is much less joint drifting errors at the same cost as PGS. If the number of joints is sufficiently small, we could even use an incomplete Cholesky factorization to solve for joint impulses, resulting in very accurate solutions.

Taking the concept of blocking a step further, BGS can be used to partitioning a configuration into sub-blocks where specialized solvers can be applied for each sub-block. This is termed *hierarchical solvers* in the graphics and gaming community. Instead of using a Gauss-Seidel

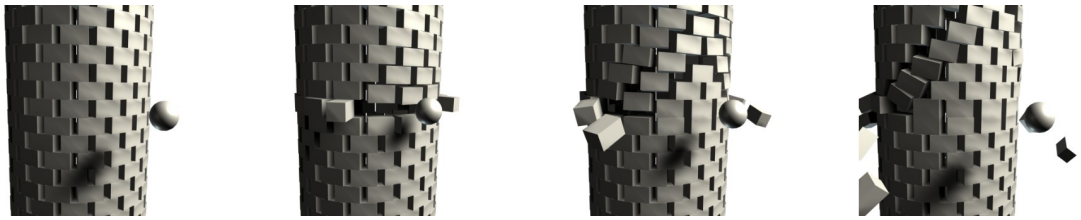


Fig. 18. Simulation of a canon ball hitting a tower. The simulation uses a velocity based shock-propagation, fixed time stepping method [Erleben 2007]. This is an example of a blocking method using gravity to create layers of blocks of a boxed linear complementarity problem (BLCP). Incredible details and non-viscous contact interaction can be obtained with this method.

splitting, we could use a blocked Jacobi method or a blocked red and black Gauss-Seidel method. This can be beneficial as domain decomposition techniques for distributed or parallel system solvers.

The blocking concepts are rather generic and can be applied to the more general class of BLCP problems too. We leave it to the reader to explore those ideas further.

Erleben [2007] realize a shock-propagation time-stepping method that can be explained as a specific blocking method that divides the contact points in a simulation into disjoint spatial chunks based on layers along the direction of gravity. This allows for fast propagation of shocks in the up-down gravitational direction. Figure 18 shows an example demonstrating the high fidelity visual details that a proper blocking strategy can create.

Erleben [2005] uses a blocked PGS for a BLCP type of problem with variable lower and upper bounds. Figure 19 shows simulation results obtained with this method and Figure 20 illustrates typical convergence rates obtained during a single time-step. Notice that for these specific simulations, the method still obtains a linear convergence rate, although the convergence constant varies a lot across simulations.

3.2.5 Staggering. We will now combine the ideas of splitting the LCP and using QP reformulations. This is collectively referred to as staggering [Kaufman et al. 2008; Lötstedt 1984]. We illustrate the idea for a multiple contact problem. We partition the LCP variables into three index sets: one corresponding to normal impulses $\mathcal{N}$, one to friction impulses $\mathcal{F}$, and the last one is slack variables β. Applying our partition would require us to solve the two coupled LCPs,

$$\mathbf{0} \leq \mathbf{A}_{\mathcal{N}\mathcal{N}}\mathbf{x}_{\mathcal{N}} + (\mathbf{b}_{\mathcal{N}} + \mathbf{A}_{\mathcal{N}\mathcal{F}}\mathbf{x}_{\mathcal{F}}) \quad \perp \quad \mathbf{x}_{\mathcal{N}} \geq \mathbf{0} \tag{161}$$

and

$$\mathbf{0} \leq \begin{bmatrix} \mathbf{A}_{\mathcal{F}\mathcal{F}} & \mathbf{e} \\ -\mathbf{e}^T & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x}_{\mathcal{F}} \\ \beta \end{bmatrix} + \begin{bmatrix} \mathbf{b}_{\mathcal{F}} + \mathbf{A}_{\mathcal{F}\mathcal{N}}\mathbf{x}_{\mathcal{N}} \\ \mu\mathbf{x}_{\mathcal{N}} \end{bmatrix} \quad \perp \quad \begin{bmatrix} \mathbf{x}_{\mathcal{F}} \\ \beta \end{bmatrix} \geq \mathbf{0}\,. \tag{162}$$

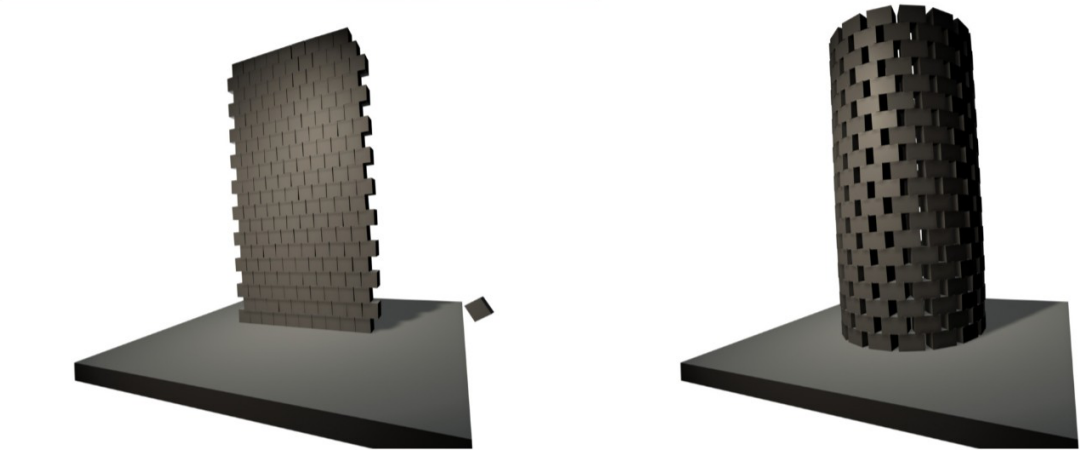
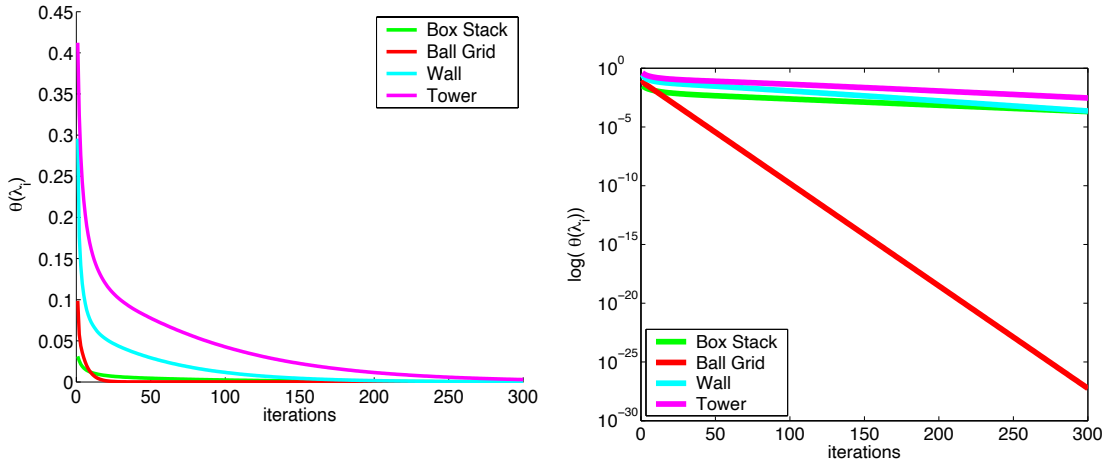


Fig. 19. Simulations of wall and a tower using a blocked Projected Gauss-Seidel (PGS) type of method for boxed linear complementarity problem (BLCP) with variable lower and upper bounds [Erleben 2005].



(a) Convergence (b) Log Convergence

Fig. 20. Convergence rate plots of a single time-step for each of the configurations shown in Figure 19.

In a staggered approach, we first solve the normal force LCP and then the friction force LCP. This is repeated iteratively until a fixed-point is reached. This is in essence a specialized blocked Gauss-Seidel splitting method.

Using the splitting of Equation 161 and Equation 162, we note that the normal force problem has a symmetric positive semi-definite coefficient matrix $\mathbf{A}_{\mathcal{N}\mathcal{N}}$ making QP reformulations possible, whereas the frictional problem has a non-symmetric matrix. However, because the

friction LCP is equivalent to the first order optimality conditions of the QP problem

$$\mathbf{x}^*_{\mathcal{F}} = \arg\min \ \frac{1}{2}\mathbf{x}^T_{\mathcal{F}}\mathbf{A}_{\mathcal{F}\mathcal{F}}\mathbf{x}_{\mathcal{F}} + \mathbf{c}^T_{\mathcal{F}}\mathbf{x}_{\mathcal{F}} \tag{163}$$

subject to

$$\mathbf{x}_{\mathcal{F}} \geq \mathbf{0} \quad \text{and} \quad c_{\mathcal{N}} - \mathbf{e}^T\mathbf{x}_{\mathcal{F}} \geq 0\,, \tag{164}$$

where $c_{\mathcal{N}} = \mu\mathbf{x}_{\mathcal{N}}$ and $\mathbf{c}_{\mathcal{F}} = \mathbf{b}_{\mathcal{F}} + \mathbf{B}_{\mathcal{F}\mathcal{N}}\mathbf{x}_{\mathcal{N}}$, any convex QP method can be used to solve for the normal and friction forces. Thus, we are guaranteed to find a solution for each sub-problem. Whether the sequence of QP sub-problems converge to a fixed point is not obvious. There exist many variations over this staggering scheme [Lacoursiere and Linde 2011].

Silcowitz et al. [2009] apply a kind of semi-staggering method for BLCPs. They found that using this simple semi-staggering approach to warm start a Newton method shows how dramatically staggering can affect the convergence behavior.

3.2.6 The Projected Gauss-Seidel Subspace Minimization Method. We extend the general PGS method, by tailoring it to the generalized boxed LCP problem class which allows for variable lower and upper bounds. The method we derive originates from work on solving the contact force problem [Silcowitz et al. 2010b]. The projected Gauss-Seidel with Subspace Minimization (PGS-SM) method is an iterative method, and each iteration consists of two phases:

(1) The first phase estimates a set of active constraints F using the standard PGS method.
(2) The second phase solves accurately for the active constraints, potentially further reducing F for the next iteration.

Phase I. This phase consists of running a standard PGS method to solve for $\mathbf{x}$. When the PGS algorithm terminates, we know that $\mathbf{x}$ is feasible, although not necessarily the correct solution. However, $\mathbf{v}$ may be infeasible due to the projection on $\mathbf{x}$ made by the PGS method. Recall the definitions given by Equation 112a-112c, which are practical for labeling and eliminating some of the unknowns based on analysis of the feasibility and complementarity conditions. In fact, the pivoting based solver exploited this aspect of the index set definitions. A similar approach is taken by the second phase of the PGS-SM method, which solves for a a subset of variables.

Phase II. The linear system may be partitioned as in Equation 111 into three distinct sets: free and tight lower and upper bounded. To solve this system for the unknowns $\mathbf{v}_{\mathsf{L}}$, $\mathbf{v}_{\mathsf{U}}$ and $\mathbf{x}_{\mathsf{F}}$, we first compute $\mathbf{x}_{\mathsf{F}}$ by solving

$$\mathbf{A}_{\mathsf{F},\mathsf{F}}\mathbf{x}_{\mathsf{F}} = -\left(\mathbf{b}_{\mathsf{F}} + \mathbf{A}_{\mathsf{F},\mathsf{L}}\boldsymbol{\lambda}^{\mathrm{lo}}{}_{\mathsf{L}} + \mathbf{A}_{\mathsf{F},\mathsf{U}}\boldsymbol{\lambda}^{\mathrm{hi}}{}_{\mathsf{U}}\right)\,. \tag{165}$$

Here, $\mathbf{A}_{\mathsf{F},\mathsf{F}}$ is a symmetric principal sub-matrix of $\mathbf{A}$. Knowing $\mathbf{x}_{\mathsf{F}}$, we can easily compute $\mathbf{v}_{\mathsf{L}}$ and $\mathbf{v}_{\mathsf{U}}$,

$$\mathbf{v}_{\mathsf{L}} = \mathbf{A}_{\mathsf{L}\mathsf{F}}\mathbf{x}_{\mathsf{F}} + \mathbf{A}_{\mathsf{L}\mathsf{L}}\boldsymbol{\lambda}^{\mathrm{lo}}{}_{\mathsf{L}} + \mathbf{A}_{\mathsf{L}\mathsf{U}}\boldsymbol{\lambda}^{\mathrm{hi}}{}_{\mathsf{U}} + \mathbf{b}_{\mathsf{L}}, \tag{166a}$$

$$\mathbf{v}_{\mathsf{U}} = \mathbf{A}_{\mathsf{U}\mathsf{F}}\mathbf{x}_{\mathsf{F}} + \mathbf{A}_{\mathsf{U}\mathsf{L}}\boldsymbol{\lambda}^{\mathrm{lo}}{}_{\mathsf{L}} + \mathbf{A}_{\mathsf{U}\mathsf{U}}\boldsymbol{\lambda}^{\mathrm{hi}}{}_{\mathsf{U}} + \mathbf{b}_{\mathsf{U}}. \tag{166b}$$

Now, we check that the feasibility conditions are satisfied: $\mathbf{v}_{\mathsf{L}} < 0$, $\mathbf{v}_{\mathsf{U}} > 0$ and $\boldsymbol{\lambda}^{\mathrm{lo}}{}_{\mathsf{F}} \leq \mathbf{x}_{\mathsf{F}} \leq \boldsymbol{\lambda}^{\mathrm{hi}}{}_{\mathsf{F}}$. If all constraints are satisfied, we have reached a solution. Rather than testing the constraints explicitly, a projection is performed on the reduced problem:

$$\mathbf{x}_{\mathsf{F}} \leftarrow \min(\mathbf{x}_{\mathsf{F}}^{\mathrm{hi}}, \max(\mathbf{x}_{\mathsf{F}}^{\mathrm{lo}}, \mathbf{x}_{\mathsf{F}})) \,. \tag{167}$$

We assemble the full solution vector $\mathbf{x} \leftarrow \begin{bmatrix} \mathbf{x}_{\mathsf{F}}^T & \mathbf{l}_{\mathsf{L}}^T & \mathbf{u}_{\mathsf{U}}^T \end{bmatrix}^T$ before re-estimating the index sets for the next iteration. The projection on the reduced problem will either leave the active set unchanged or reduce it further. See Algorithm 9 for the full pseudo code of the PGS-SM method.

Notice that Algorithm 9 does not specify which termination criteria to use . A particularly useful termination criterion for the PGS-SM method could be to monitor if the set F has changed from the previous iteration,

$$\mathsf{F}^{k+1} = \mathsf{F}^{k} \,. \tag{168}$$

Figure 21 shows different simulation results obtained with the PGS-SM method [Silcowitz et al. 2010b]. As shown the PGS-SM method behaves rather well for small configurations and configurations with joints. For larger configurations, we obtain convergence results similar to the PGS method.

3.2.7 The Non-Smooth Nonlinear Conjugate Gradient Method. Silcowitz et al. [2010a] shows that the PGS iteration can be written in generic form using Equation 158 as

$$\mathbf{x}^{k+1} = \min(\underbrace{\mathbf{u}(\mathbf{x}^k)}_{\mathbf{T}_U\mathbf{x}^k+\mathbf{t}_U}, \max(\underbrace{\mathbf{l}(\mathbf{x}^k)}_{\mathbf{T}_L\mathbf{x}^k+\mathbf{t}_L}, \underbrace{-(\mathbf{D}+\mathbf{L})^{-1}(\mathbf{U}\mathbf{x}^k+\mathbf{b})}_{\mathbf{T}\mathbf{x}^k+\mathbf{t}})) \tag{169}$$

where the lower and upper bound functions $\mathbf{l}, \mathbf{u} : \mathbb{R}^n \mapsto \mathbb{R}^n$ are affine functions. The $\mathbf{T}_L$ and $\mathbf{T}_U$ matrices express the linear relations between the tangential friction forces and their associated normal forces. The $\mathbf{t}_L$ and $\mathbf{t}_U$ vectors can be used to express fixed bound constraints, such as a normal force constraint. Thus, the PGS iteration can be perceived as a selector function of three affine functions.

Assuming a converging sequence $\mathbf{x}^k \rightarrow \mathbf{x}^*$ for $k \rightarrow \infty$ the solution of PGS can be written as the fixed point formulation,

$$\mathbf{x}^* = \underbrace{\min(\mathbf{T}_U\mathbf{x}^* + \mathbf{t}_U, \max(\mathbf{T}_L\mathbf{x}^* + \mathbf{t}_L, \mathbf{T}\mathbf{x}^* + \mathbf{t}))}_{\simeq \mathbf{G}\mathbf{x}^*+\mathbf{g}} \,. \tag{170}$$

The right hand side of Equation 170 can be conceptually considered as the evaluation of an affine function, $\mathbf{G}\mathbf{x}^* + \mathbf{g}$. This is true if the active set of constraints is known in advance. Therefore, we have

$$0 = (\mathbf{G} - \mathbf{I})\mathbf{x}^* + \mathbf{g} \,. \tag{171}$$

Data: k_{pgs}: Number of PGS sweeps per outer iteration, k_{sm}: Number of subspace solves per outer iteration, $\mathbf{x}$: The initial iterate, $\mathbf{A}$: The coefficient matrix of the BLCP, $\mathbf{b}$: The right hand side vector the BLCP.
Result: $\mathbf{x}$: The numerical solution for the BLCP.

while *not converged* **do**
  $\mathbf{x} \leftarrow$ run PGS for at least k_{pgs} iterations;
  if *termination criteria reached* **then**
    return $\mathbf{x}$;
  end
  for $k \leftarrow 1$ **to** k_{sm} **do**
    $\mathsf{L} \equiv \{i | \mathbf{x}_i = \mathbf{l}_i\}$;
    $\mathsf{U} \equiv \{i | \mathbf{x}_i = \mathbf{u}_i\}$;
    $\mathsf{F} \equiv \{i | \mathbf{l}_i < \mathbf{x}_i < \mathbf{u}_i\}$;
    solve: $\mathbf{A}_{\mathsf{F},\mathsf{F}}\mathbf{x}_{\mathsf{F}} = -\left(\mathbf{b}_{\mathsf{F}} + \mathbf{A}_{\mathsf{F},\mathsf{L}}\mathbf{l}_{\mathsf{L}} + \mathbf{A}_{\mathsf{F},\mathsf{U}}\mathbf{u}_{\mathsf{U}}\right)$;
    $\mathbf{v}_{\mathsf{L}} \leftarrow \mathbf{A}_{\mathsf{L},\mathsf{F}}\mathbf{x}_{\mathsf{F}} + \mathbf{A}_{\mathsf{L},\mathsf{L}}\mathbf{l}_{\mathsf{L}} + \mathbf{A}_{\mathsf{L},\mathsf{U}}\mathbf{u}_{\mathsf{U}} + \mathbf{b}_{\mathsf{L}}$;
    $\mathbf{v}_{\mathsf{U}} \leftarrow \mathbf{A}_{\mathsf{U},\mathsf{F}}\mathbf{x}_{\mathsf{F}} + \mathbf{A}_{\mathsf{U},\mathsf{L}}\mathbf{l}_{\mathsf{L}} + \mathbf{A}_{\mathsf{U},\mathsf{U}}\mathbf{u}_{\mathsf{U}} + \mathbf{b}_{\mathsf{U}}$;
    update: $(\mathbf{l}, \mathbf{u})$;
    $\mathbf{x}_{\mathsf{F}} \leftarrow \min(\mathbf{u}_{\mathsf{F}}, \max(\mathbf{l}_{\mathsf{F}}, \mathbf{x}_{\mathsf{F}}))$;
    $\mathbf{x} \leftarrow \begin{bmatrix} \mathbf{x}_{\mathsf{F}}^T & \mathbf{l}_{\mathsf{L}}^T & \mathbf{u}_{\mathsf{U}}^T \end{bmatrix}^T$;
    if *termination criteria reached* **then**
      return $\mathbf{x}$
    end
  end
end

Algorithm 9: Projected Gauss-Seidel subspace minimization (PGS-SM) method. Observe that is can be seen as a PGS solver followed by an active-set strategy solving a sequence of linear systems while efficiently updating the active index set. PGS becomes a prior for making a good initial estimate of the active set.

Observe, explicit assembly is not needed for any of the matrices, instead the PGS method can be used to implicitly evaluate the residual of any given iteration, $\mathbf{r}^k = (\mathbf{G} - \mathbf{I})\mathbf{x}^k + \mathbf{g}$. Thus, if we write one iteration of the standard PGS method as

$$\mathbf{x}^{k+1} = \mathbf{PGS}(\mathbf{x}^k) \tag{172}$$

then $\mathbf{r}^k = \mathbf{x}^{k+1} - \mathbf{x}^k = \mathbf{PGS}(x^k) - \mathbf{x}^k$. This can be thought of as the gradient of a non-smooth nonlinear quasi-quadratic function $f(\mathbf{x}^k) \approx \frac{1}{2}\|\mathbf{r}^k\|^2$. We are essentially seeking a

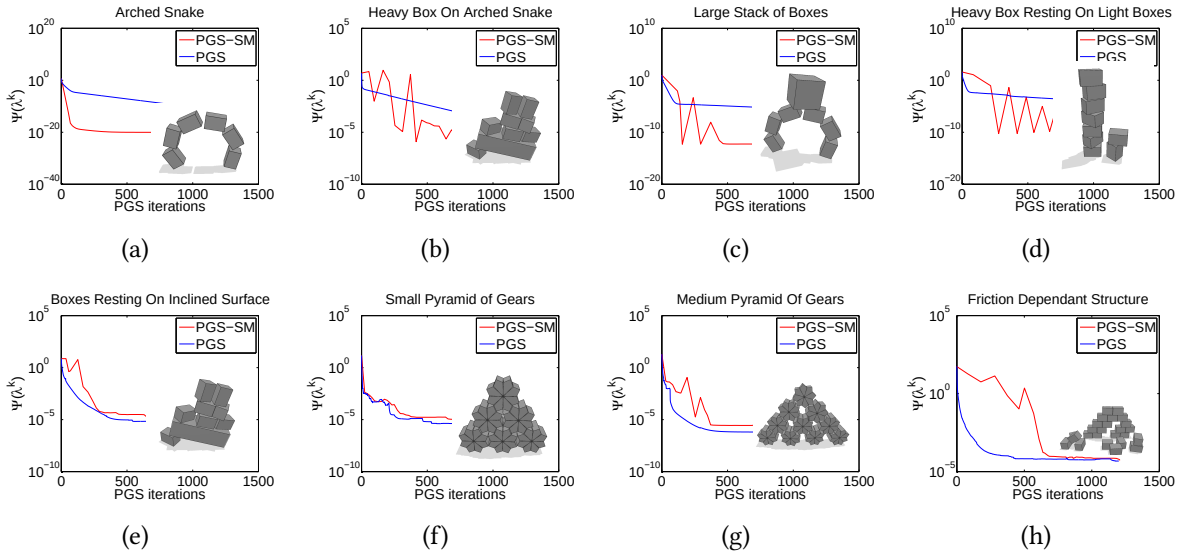


Fig. 21. Test cases for the comparison of PGS and PGS–SM methods with corresponding convergence plots. Observe the jaggedness in the PGS–SM plots in (b), (c), (d), and (g). The spikes indicates that the PGS–SM method guessed a wrong active set. This can cause the merit function to rise abruptly. The ψ function is the Fischer–Burmeister function from Silcowitz et al. [2009]. The x-axis is measured in units of one PGS iteration to make comparison easier.

local minimizer of f, however, we only know its gradient $\nabla f(\mathbf{x}^k) = \mathbf{r}^k$. The Fletcher-Reeves nonlinear conjugate gradient method is perfect for this [Nocedal and Wright 2006].

In each iteration of the conjugate gradient method we perform the update,

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \tau^k \mathbf{p}^k \tag{173}$$

where $\mathbf{p}^k$ is the search direction and τ^k can be found using a line search method. Next a new search direction is computed by

$$\beta^{k+1} = \frac{\left\|\nabla f^{k+1}\right\|^2}{\left\|\nabla f^k\right\|^2}, \tag{174a}$$

$$\mathbf{p}^{k+1} = \beta^{k+1}\mathbf{p}^k - \nabla f^{k+1}. \tag{174b}$$

In an an interactive context, the line search on $f(\mathbf{x}^{k+1})$ is dropped. Here the full step length, $\tau = 1$, is used and the method is restarted whenever $\|\nabla f^{k+1}\|^2 > \|\nabla f^k\|^2$.

When computing $\mathbf{PGS}(\mathbf{x}^k)$, it is practical to do so in-place, meaning that a PGS step is taken implicitly on $\mathbf{x}^k$. Therefore, the update of $\mathbf{x}^k$ is done separately in two places in each iteration. The full algorithm is stated in Algorithm 10.

Data: $\mathbf{x}$: The initial iterate, $\mathbf{A}$: The coefficient matrix of the LCP , $\mathbf{b}$: The right hand side of the LCP.
Result: $\mathbf{x}$: The numerical solution for the LCP.

$\mathbf{x}^1 \leftarrow \mathbf{PGS}(\mathbf{x}^0)$;
$\nabla f^0 \leftarrow -(\mathbf{x}^1 - \mathbf{x}^0)$;
$\mathbf{p}^0 \leftarrow -\nabla f^0$;
$k \leftarrow 1$;
while *not converged* **do**
  $\mathbf{x}^{k+1} \leftarrow \mathbf{PGS}(\mathbf{x}^k)$;
  $\nabla f^k \leftarrow -(\mathbf{x}^{k+1} - \mathbf{x}^k)$;
  $\beta^k \leftarrow \left\|\nabla f^k\right\|^2 / \left\|\nabla f^{k-1}\right\|^2$;
  if $\beta^k > 1$ **then**
    $\mathbf{p}^k \leftarrow 0$; /* restart */
  else
    $\mathbf{x}^{k+1} \leftarrow \mathbf{x}^{k+1} + \beta^k \mathbf{p}^{k-1}$;
    $\mathbf{p}^k \leftarrow \beta^k \mathbf{p}^{k-1} - \nabla f^k$;
  end
  $k \leftarrow k + 1$;
end

Algorithm 10: Nonsmooth nonlinear conjugate gradient (NNCG) method. Observe how a single PGS sweep is used to compute the gradient ∇f of some imaginary objective function f that NNCG method tries to minimize. It is surprisingly that we do not need to be able to explicitly create f.

An implementation of the non-smooth nonlinear conjugate gradient (NNCG) method can be found in [Silcowitz-Hansen 2010] and [Coumans 2005]. Detailed convergence studys may be found in [Silcowitz et al. 2010a]. Figure 22 shows corresponding convergence rates for some simulation results, taken from [Silcowitz et al. 2010a]. Figure 22 Notice the dramatic change in convergence behavior. The improved accuracy means the NNCG method is more equipped to deal with for instance large mass ratios. Usually PGS performs slow on such problems and NNCG will out-perform it. Further, NNCG is overall computationally cheaper than applying a direct method which otherwise would be able to handle large mass ratios.

3.3 Non-Smooth Newton Methods

Let us just remember the classical Newton method for solving a generic root finding problem $\mathbf{F}(\mathbf{x}) = 0$, that is solve for $\mathbf{x}$ such that the vector function $\mathbf{F}$ is zero. The Newton method is

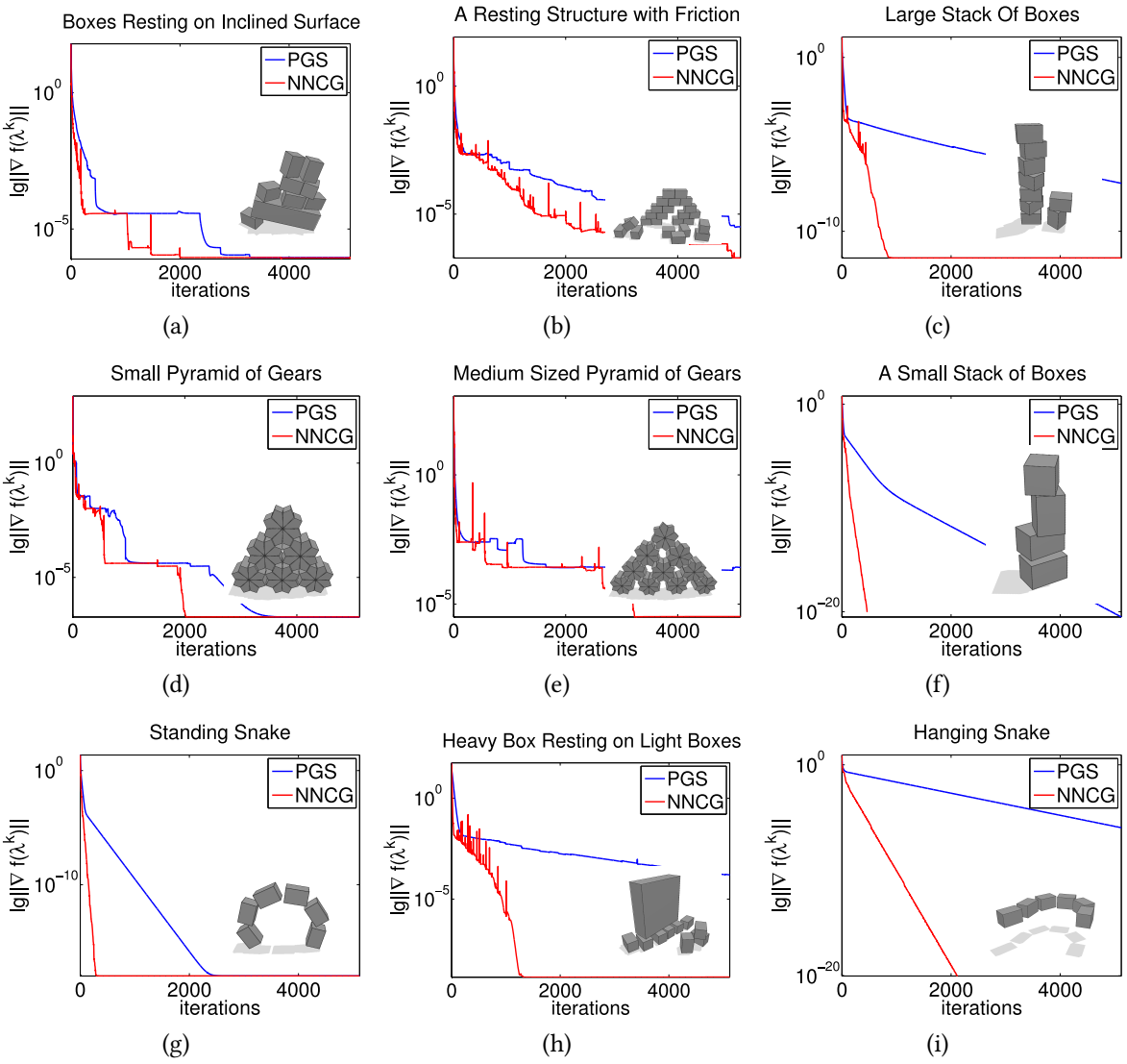


Fig. 22. Test results comparing the nonlinear conjugate gradient method (NNCG) with projected Gauss-Seidel (PGS) method with up to 5000 iterations. The NNCG method clearly converges faster, and often to a higher accuracy than that of the PGS method. Notice the superior rate of convergence in (f)-(i).

classically derived from a first order Taylor approximation around the k^{th} iterate,

$$\mathbf{F}(\mathbf{x}^k + \Delta\mathbf{x}) \approx \mathbf{F}(\mathbf{x}^k) + \frac{\partial \mathbf{F}(\mathbf{x}^k)}{\partial \mathbf{x}} \Delta\mathbf{x}\,. \tag{175}$$

Setting the approximation equal to zero we have the Newton equation,

$$\frac{\partial \mathbf{F}(\mathbf{x}^k)}{\partial \mathbf{x}} \Delta \mathbf{x} = -\mathbf{F}(\mathbf{x}^k)\,. \tag{176}$$

The $\Delta \mathbf{x}$ is called the search direction or Newton direction. We solve the Newton equation for finding the search direction. The search direction is then used in the Newton update rule given by,

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \Delta \mathbf{x}\,. \tag{177}$$

The Newton method can be globalized by adding a line-search method to the update rule. This essentially consist of multiplying the search direction by a scalar line-step parameter. Line-search strategies are sometimes avoided in interactive and real-time simulations as they can be expensive, but back-tracking strategies are quite popular when a line-search is added. In these notes we will not go into line-search methods, but instead focus on how to reformulate the contact and friction problem into a root finding problem. In the text below we will start by building up the mathematical parts that allow us to define a $\mathbf{F}$-function in the context of contact problems and provide the building blocks for later showing how to assemble $\frac{\partial \mathbf{F}(\mathbf{x}^k)}{\partial \mathbf{x}}$ and solve for $\Delta \mathbf{x}$.

One successful approach by Ferris and Munson [1999] is to reformulate a complementarity problem in terms of a NCP function whose roots satisfy the original complementarity conditions. In other words, functions where the following equivalence holds:

$$\psi(a, b) = 0 \iff 0 \leq a \perp b \geq 0\,. \tag{178}$$

Combined with an appropriate time-discretization, such a NCP function turns our contact problem into a root finding one. In general the functions ψ are non-smooth, but allow us to apply a wide range of numerical root finding methods [Munson et al. 2001].

Let us review some of the most commonly used NCP functions and show how to rephrase the contact problem into an equivalent root finding problem.

3.3.1 Minimum-Map Formulation. The first NCP function we will consider is the *minimum-map* defined as

$$\psi^{\text{mm}}(a, b) \equiv \min(a, b). \tag{179}$$

The equivalence of this function to the original NCP can be verified by examining the values associated with each conditional case for finding a root for the equation $\psi^{\text{mm}} = 0$. We now consider how this reformulation applies to unilateral contact constraints. Recall that the complementarity condition associated with a contact constraint $\phi(\mathbf{q})$ and its associated Lagrange multiplier $\lambda_{\hat{n}}$ is

$$0 \leq \phi(\mathbf{q}) \perp \lambda_{\hat{n}} \geq 0\,. \tag{180}$$

We can write this in the equivalent minimum-map form of the contact constraint as

$$\psi_{\hat{n}}(\mathbf{q}, \lambda_{\hat{n}}) \equiv \psi^{\text{mm}}(\phi(\mathbf{q}), \lambda_{\hat{n}}) = 0\,, \tag{181}$$

which has the following derivatives,

$$\frac{\partial \psi_{\hat{n}}}{\partial \mathbf{q}} = \begin{cases} \nabla\phi(\mathbf{q}), & \phi(\mathbf{q}) \leq \lambda_{\hat{n}} \\ \mathbf{0}, & \text{otherwise} \end{cases}, \tag{182}$$

$$\frac{\partial \psi_{\hat{n}}}{\partial \lambda_{\hat{n}}} = \begin{cases} 0, & \phi(\mathbf{q}) \leq \lambda_{\hat{n}} \\ 1, & \text{otherwise} \end{cases}. \tag{183}$$

From these cases we can see that the minimum-map gives rise to an active-set style method where a contact is considered active if $\phi(\mathbf{q}) \leq \lambda_{\hat{n}}$. Active contacts are treated as equality constraints, while for inactive contacts the minimum-map enforces that the constraint's Lagrange multiplier is zero.

3.3.2 Fischer-Burmeister Formulation. An alternative NCP function is given by Fischer [1992], who observe the roots of the following equation satisfy complementarity:

$$\psi^{\text{fb}}(a, b) = a + b - \sqrt{a^2 + b^2} = 0\,. \tag{184}$$

This is the Fischer-Burmeister function, and it is interesting because, unlike the minimum-map, it is smooth everywhere apart from the point $(a, b) = (0, 0)$. For $(a, b) \neq (0, 0)$ the partial derivatives of the Fisher-Burmeister function are given by:

$$\alpha(a, b) = \frac{\partial \psi^{\text{fb}}}{\partial a} = 1 - \frac{a}{\sqrt{a^2 + b^2}}\,, \tag{185}$$

$$\beta(a, b) = \frac{\partial \psi^{\text{fb}}}{\partial b} = 1 - \frac{b}{\sqrt{a^2 + b^2}}\,. \tag{186}$$

At the point $(a, b) = (0, 0)$ the derivative is set-valued. For Newton methods it suffices to choose any value from this sub-gradient. Erleben et al. [2011] compared how the choice of derivative at the non-smooth point affects convergence for LCP problems and found no overall best strategy. Thus, for simplicity we make the arbitrary choice of

$$\alpha(0, 0) = 0\,, \tag{187}$$

$$\beta(0, 0) = 1\,. \tag{188}$$

For a contact constraint ϕ, with Lagrange multiplier $\lambda_{\hat{n}}$ we may then write our contact constraint alternatively as,

$$\psi_{\hat{n}}(\mathbf{q}, \lambda_{\hat{n}}) \equiv \psi^{\text{fb}}(\phi(\mathbf{q}), \lambda_{\hat{n}}) = 0\,, \tag{189}$$

with derivatives given by

$$\frac{\partial \psi_{\hat{n}}}{\partial \mathbf{q}} = \alpha(\phi, \lambda_{\hat{n}}) \nabla \phi \,, \tag{190}$$

$$\frac{\partial \psi_{\hat{n}}}{\partial \lambda_{\hat{n}}} = \beta(\phi, \lambda_{\hat{n}}) \,. \tag{191}$$

3.3.3 Friction. Coulomb's law can be derived from a principle of maximal dissipation that requires the frictional forces remove the maximum amount of energy from the system while having their magnitude bounded by the normal force:

$$\underset{\boldsymbol{\lambda}_{\hat{t}}}{\arg\min} \quad \mathbf{v}_{\hat{t}}{}^{T} \boldsymbol{\lambda}_{\hat{t}} \quad \text{subject to} \quad \|\boldsymbol{\lambda}_{\hat{t}}\| \leq \mu \lambda_{\hat{n}} \,. \tag{192}$$

This minimization defines an admissible cone that the total contact force must lie in. The Lagrangian associated with this minimization is

$$\mathcal{L}(\boldsymbol{\lambda}_{\hat{t}}, \lambda_{\hat{n}}) \equiv \mathbf{v}_{\hat{t}}{}^{T} \boldsymbol{\lambda}_{\hat{t}} + \gamma(\|\boldsymbol{\lambda}_{\hat{t}}\| - \mu \lambda_{\hat{n}}) \,. \tag{193}$$

where γ is a slack variable used to enforce the Coulomb constraint that the friction force magnitude is bounded by μ times the normal force magnitude. When $\mu \lambda_{\hat{n}} > 0$ the problem satisfies Slater's condition [Boyd and Vandenberghe 2004] and we can use the first-order Karush-Kuhn-Tucker (KKT) conditions, given by

$$\mathbf{s}(\mathbf{u}, \boldsymbol{\lambda}_{\hat{t}}, \gamma) \equiv \mathbf{J}_{\hat{t}} \mathbf{u} + \gamma \frac{\partial \|\boldsymbol{\lambda}_{\hat{t}}\|}{\partial \boldsymbol{\lambda}_{\hat{t}}} = \mathbf{0} \tag{194}$$

$$0 \leq \gamma \;\perp\; \mu \lambda_{\hat{n}} - \|\boldsymbol{\lambda}_{\hat{t}}\| \geq 0 \,, \tag{195}$$

where γ is a slack variable that governs stick/slip behavior, and we used the block notation for the mapping $\mathbf{v} = \mathbf{J}\mathbf{u}$ given by

$$\mathbf{v} = \begin{bmatrix} v_{\hat{n}} \\ \mathbf{v}_{\hat{t}} \end{bmatrix} = \underbrace{\begin{bmatrix} \mathbf{J}_{\hat{n}} \\ \mathbf{J}_{\hat{t}} \end{bmatrix}}_{\mathbf{J}} \mathbf{u} \,. \tag{196}$$

The complementarity condition may be directly written using any NCP function as,

$$\psi_{\hat{t}} \equiv \psi(\gamma, \mu \lambda_{\hat{n}} - \|\boldsymbol{\lambda}_{\hat{t}}\|) = 0 \,. \tag{197}$$

3.3.4 Newton's Method. We may now define generically for any chosen NCP function,

$$\mathbf{x} \equiv \begin{bmatrix} \mathbf{u} \\ \lambda_{\hat{n}} \\ \boldsymbol{\lambda}_{\hat{t}} \\ \gamma \end{bmatrix} \quad \text{and} \quad \mathbf{F}(\mathbf{x}) \equiv \begin{bmatrix} \mathbf{M}\mathbf{u} - \mathbf{J}^{T}\lambda + \mathbf{f} \\ \psi_{\hat{n}}(\phi(\mathbf{q}), \lambda_{\hat{n}}) \\ \mathbf{s}(\mathbf{u}, \boldsymbol{\lambda}_{\hat{t}}, \gamma) \\ \psi_{\hat{t}}(\gamma, \mu \lambda_{\hat{n}} - \|\boldsymbol{\lambda}_{\hat{t}}\|) \end{bmatrix} = \mathbf{0} \,, \tag{198}$$

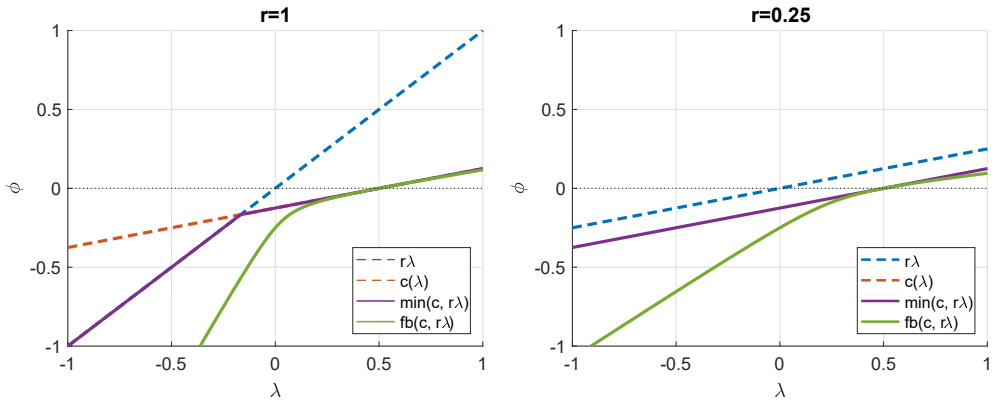


Fig. 23. Left: The minimum-map of a unilateral constraint has a kink in it at the cross over point. Right: Choosing an appropriate r-factor can remove the discontinuity by forcing both terms to be parallel. For a single constraint this results in a straight-line error function that can be solved in one step regardless of starting point. In the case of Fischer-Burmeister (green) the error function's curvature is reduced. For illustration purposes we have shown the constraint function $c = \frac{1}{4}\lambda - \frac{1}{8} > 0$, which has a unique solution at $\lambda = \frac{1}{2}$.

Observe the top-row in $\mathbf{F}$ is the time-discretized Newton Euler equations. One may replace the normal reformulation with one that is based on the separation velocity $v_{\hat{n}}$ rather than the normal constraint ϕ. In which case we have $\psi_{\hat{n}}(\dot{\phi}, \lambda_{\hat{n}}) = \psi_{\hat{n}}(v_{\hat{n}}, \lambda_{\hat{n}}) = \psi_{\hat{n}}(\mathbf{J}_{\hat{n}}\,\mathbf{u}, \lambda_{\hat{n}})$. We can solve for the roots of $\psi_{\hat{n}}$ and $\psi_{\hat{t}}$ using Newton's methods by linearizing the equation $\mathbf{F} = 0$ in terms of velocities and the Lagrange multipliers to obtain the following system,

$$\begin{bmatrix} \mathbf{M} & -\mathbf{J}_{\hat{n}}{}^T & -\mathbf{J}_{\hat{t}}{}^T & \mathbf{0} \\ \mathbf{J}_{\hat{n}} & \frac{\partial \psi_{\hat{n}}}{\partial \lambda_{\hat{n}}} & \mathbf{0} & \mathbf{0} \\ \mathbf{J}_{\hat{t}} & \mathbf{0} & \frac{\partial \mathbf{s}}{\partial \boldsymbol{\lambda}_{\hat{t}}} & \frac{\partial \mathbf{s}}{\partial \gamma} \\ \mathbf{0} & \frac{\partial \psi_{\hat{t}}}{\partial \lambda_{\hat{n}}} & \frac{\partial \psi_{\hat{t}}}{\partial \boldsymbol{\lambda}_{\hat{t}}} & \frac{\partial \psi_{\hat{t}}}{\partial \gamma} \end{bmatrix} \begin{bmatrix} \Delta \mathbf{u} \\ \Delta \lambda_{\hat{n}} \\ \Delta \boldsymbol{\lambda}_{\hat{t}} \\ \Delta \gamma \end{bmatrix} = - \begin{bmatrix} \mathbf{f} \\ \psi_{\hat{n}} \\ \mathbf{s} \\ \psi_{\hat{t}} \end{bmatrix}. \tag{199}$$

Here we have considered a single contact constraint for simplicity. The extension to a system of contacts is straightforward. However, it is clear that, due to the frictional constraints, Equation 199 is non-symmetric, which restricts the numerical methods we can use to solve it. Macklin et al. [2019] proposed a fixed-point iteration to eliminate β and obtain a symmetric system that can be solved with iterative Krylov methods.

3.3.5 Preconditioning. Similar to prox formulations, the NCP functions presented in this section have a free parameter r that may be used to rescale the functions. Specifically, a solution to $\psi(a, b) = 0$, is also a solution to the scaled problem, $\psi(a, rb) = 0$, where r may be chosen as any positive constant. In practice this parameter is important for the robustness of Newton methods, and we visualize its effect on the NCP functions in Figure 23.

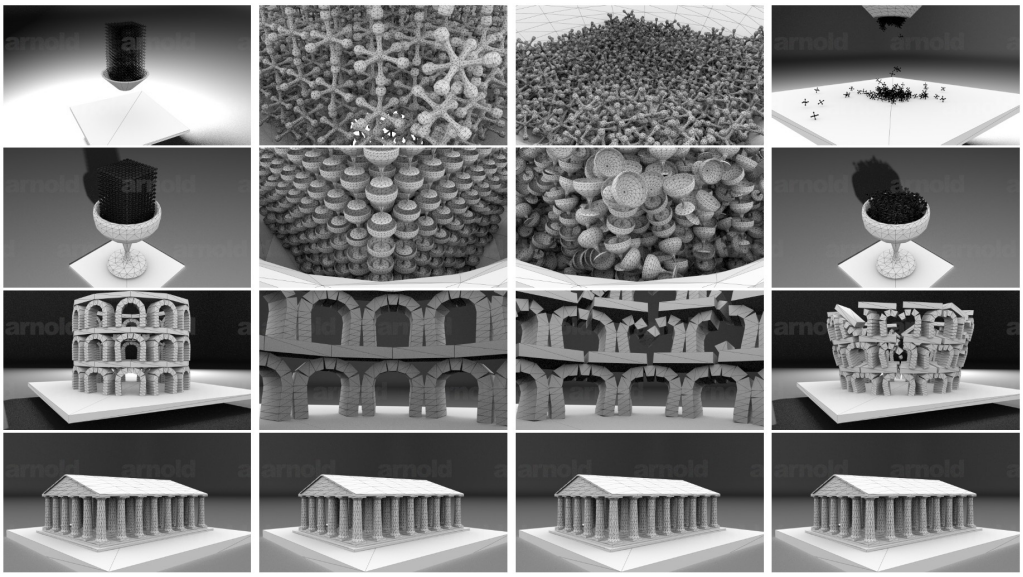


Fig. 24. Examples of rigid body simulation using PROX schemes. Top row shows digital prototyping application studying jamming due to friction properties, upper-middle: packing into a container, middle-lower: building collapse due to poor design. Bottom demonstrates structured stacking.

3.4 Proximal Operators

Iterative methods are popular for solving contact force problems in rigid body dynamics. We have extensively covered many of the LCP variations of these type of methods. The ideas do extend into other types of models. One such model is based on the proximal operator. It is a much more general model that allow one to express more nonlinear and non smooth properties in a compact way than the LCP approach often would allow one to do. In this section we extend many previous ideas covered in previous sections into this more general abstract setting building on proximal operators. We provide a mathematical foundation for computer graphics researchers for iterative (PROX) schemes based on proximal operators. Our presentation follows the outline of Erleben [2017]. We derive a class of iterative Jacobi and blocked Gauss-Seidel variants that theoretically proven always converge and provides flexible plug and play framework for exploring different friction laws as we describe in Section 6. PROX methods are both fast, scale to handle a large number of rigid bodies, deal with non-convex shapes, accurately model the physics, and be robust as well as predictable in case of user interactions as illustrated in Figure 24.

The PGS type of methods have become quite a de-facto standard for interactive entertainment. The robustness of PGS and PSOR variants are well-known as is their poor convergence behavior [Andrews et al. 2017; Erleben 2007]. We provide a derivation of such iterative schemes using proximal operators. We name these schemes PROX methods to avoid confusion with the

traditional PGS variants as described in Section 3.2. We will take a different path based on proximal operators than the usual PGS-type derivation such as the one done by Silcowitz et al. [2010a].

Interestingly, the usual PGS complementarity based friction box-model variants are special cases of a PROX method with a fixed constant r-factor (to be explained later). This is the reason why PGS can diverge and why the PROX method setting is theoretically guaranteed to always converge [Foerg et al. 2006; Parikh and Boyd 2014]. Further, PGS is tied to the box-model for performance reasons as it provides a small tight blocked memory footprint. PROX methods are in their very derivation not limited to such friction models. PROX schemes provide one with both a theoretical guarantee of convergence and a flexible modeling of any convex multi-set friction law. For completeness we show the model can express Newton style impact laws, and post-stabilization. The proximal operator model is derived directly from physical principles and is in our opinion a very powerful approach. One benefit is that the iterative scheme for computing a solution comes for free, more or less, directly from the model. No additional discretization is needed. Another benefit of this model is that it generalizes to general limit surfaces [Goyal et al. 1989] and can include torsional friction and Coulomb friction [Leine and Glocker 2003].

3.4.1 The Proximal Operator Model. We will present a contact force model for a single contact point. The notation makes use of a proximal operator, $\text{prox}_C(\mathbf{z})$. The proximal point of a convex set C to a point $\mathbf{z}$ is the point in C that minimizes the distance to $\mathbf{z}$,

$$\text{prox}_C(\mathbf{z}) \equiv \arg\min_{\mathbf{x}\in C} \|\mathbf{z}-\mathbf{x}\|^2, \quad \mathbf{z}\in\mathbb{R}^n. \tag{200}$$

this is called the proximal operator [Parikh and Boyd 2014]. This definition is illustrated in Figure 25. For friction modeling we usually only require C to be convex. Notice that C is strict convex then the solution is always unique. The weaker requirement of only being convex can imply multiple solutions. One advantage of this formulation is that we can work with a C that has a non-smooth boundary. That is the normal cone at a boundary point has a sub-space of normals. The requirement for C to be convex is one of the ingredients connected to giving strong guarantee that a fixed point scheme will converge. The proximal operator definition can be weaken to give meaning for non-convex sets too. We do not go into this discussion in this text.

Given the normal contact velocity $v_{\hat{n}} \in \mathbb{R}$ then the non-penetration constraints can be stated as

$$v_{\hat{n}} \geq 0, \quad \lambda_{\hat{n}} \geq 0, \quad \text{and} \quad v_{\hat{n}}\lambda_{\hat{n}} = 0, \tag{201}$$

or equivalently the fixed point relationship have the same solution as the non-penetration constraint [Jourdan et al. 1998]

$$\lambda_{\hat{n}} = \text{prox}_{\mathcal{N}}(\lambda_{\hat{n}} - r_{\hat{n}}v_{\hat{n}}) \quad \text{for} \quad r_{\hat{n}} > 0, \tag{202}$$

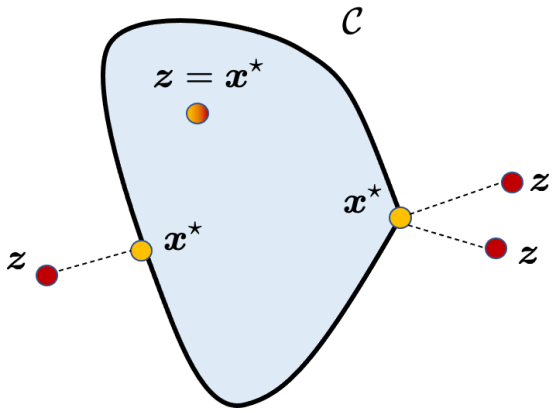


Fig. 25. The proximal operator for a set C applied to a point $\mathbf{z}$. Observe that computing a solution $\mathbf{x}^\star$ for the proximal operator is similar to point in set testing and projection to closest point on a boundary. The set C can have a non-smooth boundary.

where $\mathcal{N} \equiv \{\gamma \in \mathbb{R} \,|\, \gamma \geq 0\}$, $\lambda_{\hat{\mathbf{n}}} \in \mathbb{R}$ is the magnitude of the normal force, and $r_{\hat{n}}$ is a mathematical scalar variable named an r-factor, we use the subscript here to denote that the value is specific for the normal force as we will see later friction forces can have their own r-factor value. The equation holds for all $r_{\hat{n}} > 0$ and the exact r-factor value will have practical impact on convergence. We can graphically illustrate how the proximal operator identifies the same solutions as the complementary form of the non-penetration constraint by using the real number line. Here $\mathcal{N}$ is all numbers from origin to the right. The case-by-case analysis is shown in Figure 26.

It is not difficult to show that the proximal operator formulation includes all the LCP based formulations. For doing this it is convenient to use the slightly more general proximal operator definition

$$\mathbf{x}^\star \equiv \text{prox}_C^{\mathbf{K}} \left(\mathbf{x} - r\left(\mathbf{A}\mathbf{x} + \mathbf{b}\right)\right), \; \mathbf{K} = \mathbf{A}^{-1}, \; C = \{\mathbf{x} \,|\, \mathbf{x} \geq 0\} \; \forall r > 0\,. \tag{203}$$

Let $\mathbf{z} = \mathbf{x} - r\left(\mathbf{A}\mathbf{x} + \mathbf{b}\right)$ then by more general definition of the proximal operator we have

$$\mathbf{x}^\star \equiv \arg\min_{\mathbf{x} \in C} \frac{1}{2} \|\mathbf{x} - \mathbf{z}\|_{\mathbf{K}}^2, \tag{204}$$

which we can algebraic transform into

$$\mathbf{x}^\star \equiv \arg\min_{\mathbf{x} \in C} r^2 \left(\mathbf{A}\mathbf{x} + \mathbf{b}\right)^T \mathbf{K} \left(\mathbf{A}\mathbf{x} + \mathbf{b}\right)\,. \tag{205}$$

The positive term r^2 can be dropped without affecting the solution of the minimization problem and the quadratic term can be written out to yield the equivalent problem,

$$\mathbf{x}^\star \equiv \arg\min_{\mathbf{x} \in C} \mathbf{x}^T \mathbf{A}^T \mathbf{K} \mathbf{A} \mathbf{x} + \mathbf{b}^T \mathbf{K} \left(\mathbf{A}^T + \mathbf{A}\right) \mathbf{x} + \mathbf{b}^T \mathbf{K} \mathbf{b}\,. \tag{206}$$

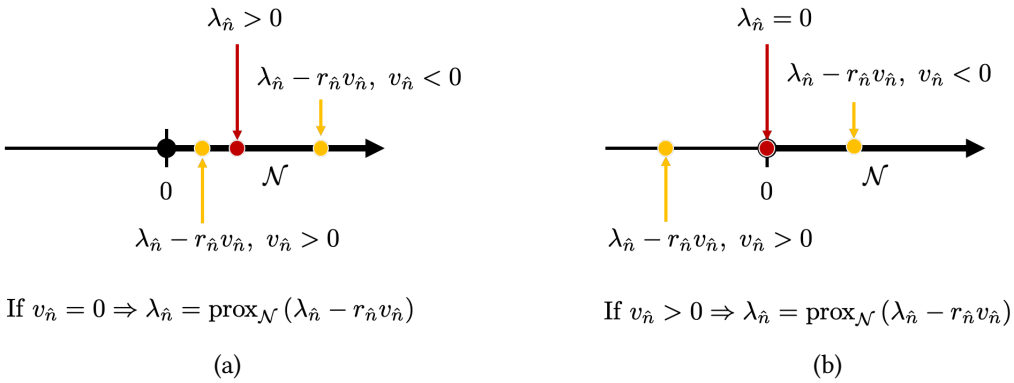


Fig. 26. In (a) we observe that if the normal force is positive then the proximal operator only has a fixed point if the normal velocity is zero. In (b) the case of zero normal shows that a fixed point only exist if the normal velocity is non-negative. This proves the fixed point formulation with the proximal operator has the same solutions as the non-penetration constraint.

Using the explicit choice of $\mathbf{K} = \mathbf{A}^{-1}$ and assuming $\mathbf{A}$ is symmetric this reduces to

$$\mathbf{x}^{\star} \equiv \arg\min_{\mathbf{x} \in C} \mathbf{x}^T \mathbf{A}\mathbf{x} + 2\mathbf{b}^T \mathbf{x} \tag{207}$$

or

$$\mathbf{x}^{\star} \equiv \arg\min_{\mathbf{x} \in C} \frac{1}{2}\mathbf{x}^T \mathbf{A}\mathbf{x} + \mathbf{b}^T \mathbf{x}\,. \tag{208}$$

This happens to be the equivalent quadratic problem (QP) reformulation of the linear complementarity problem. This can be verified by noticing that the solution to the first order optimality conditions for the QP formulation is the solution to the linear complementarity problem. This completes the proof that solutions of LCP formulations are also solutions for the proximal operator formulations.

Given the coefficient of friction $\mu > 0$ and the magnitude of normal force $\lambda_{\hat{n}} \geq 0$, then the planar friction force $\boldsymbol{\lambda}_{\hat{t}} \in \mathbb{R}^2$ is bounded by the friction cone,

$$\boldsymbol{\lambda}_{\hat{t}} \in \mathcal{F}(\mu\lambda_{\hat{n}}) \tag{209}$$

where the friction cone could be defined as we did in Section 1.5,

$$\mathcal{F}(\mu\lambda_{\hat{n}}) \equiv \left\{ \boldsymbol{\gamma}_{\hat{t}} \in \mathbb{R}^2 \;\middle|\; \left\| \boldsymbol{\gamma}_{\hat{t}} \right\| \leq \mu\lambda_{\hat{n}} \right\}\,. \tag{210}$$

This is the typical isotropic Coulomb friction law. We will later show another example. For the proximal operator model to work we only need $\mathcal{F}$ to be a convex set. We may visualize the friction cone in 3D where the x and y axis gives the components of the frictional force and the z-axis gives the normal force magnitude as shown in Figure 27. Another often used visualization is to make a corresponding 2D visualization of the intersection of the cone with the z-plane that corresponds to the current normal force as shown in middle of Figure 27.

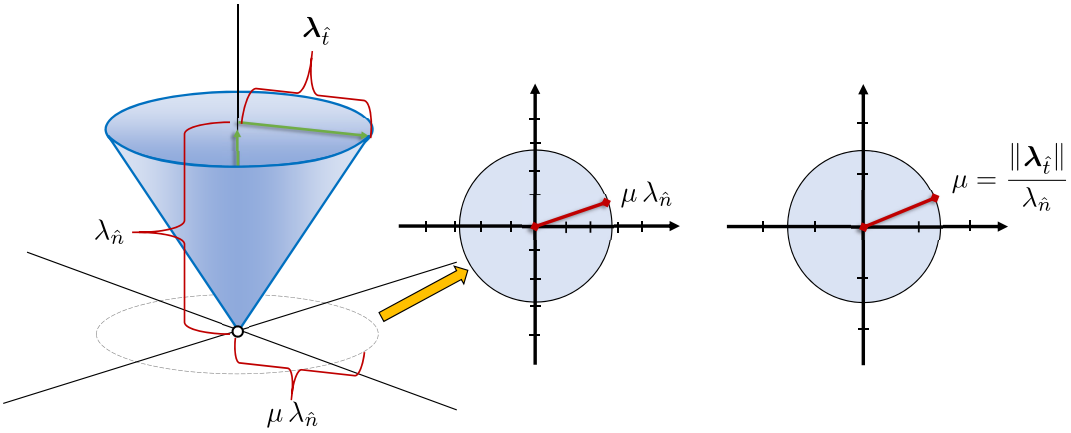


Fig. 27. On the left the isotropic Coulomb friction cone is shown as an ice-cream cone, in the middle the 2D plane intersection with the normal force plane is representing the shape of the friction set, and on the right we have a map of the set of coefficients of friction. Observe that the map of coefficients of frictions is a convenient representation of any friction cone that can be used to generate both the 2D friction set and the 3D friction cone if needed.

Because the friction force scales linear with normal force the shape of the 2D intersection will not change for any positive normal force and we may therefore just draw the shape of the cone for the normal force value $\lambda_{\hat{n}} = 1$. The cone shape then illustrates the magnitude of the coefficient of friction for different directions, as seen on right side of Figure 27.

According to principle of maximum dissipation the friction force should dissipate as much work as possible from the system. Given the friction force $\boldsymbol{\lambda}_{\hat{t}} \in \mathcal{F}(\mu\lambda_{\hat{n}})$ and the tangential contact velocity, $\mathbf{v}_{\hat{t}} \in \mathbb{R}^2$, then the dissipation power, $P_{\boldsymbol{\lambda}_{\hat{t}}}$, is,

$$P_{\boldsymbol{\lambda}_{\hat{t}}} = \boldsymbol{\lambda}_{\hat{t}}^T \mathbf{v}_{\hat{t}}\,. \tag{211}$$

Note that dissipation implies that $P_{\boldsymbol{\lambda}_{\hat{t}}} < 0$. According to the principle of maximum dissipation the power, $P_{\boldsymbol{\gamma}_{\hat{t}}}$, done by any other possible friction force, $\boldsymbol{\gamma}_{\hat{t}} \in \mathcal{F}(\mu\lambda_{\hat{n}})$, has to be larger than or equal to $P_{\boldsymbol{\lambda}_{\hat{t}}}$. From this we have

$$P_{\boldsymbol{\lambda}_{\hat{t}}} \leq P_{\boldsymbol{\gamma}_{\hat{t}}}\,, \tag{212a}$$

$$\boldsymbol{\lambda}_{\hat{t}}^T \mathbf{v}_{\hat{t}} \leq \boldsymbol{\gamma}_{\hat{t}}^T \mathbf{v}_{\hat{t}}\,, \tag{212b}$$

$$0 \leq \boldsymbol{\gamma}_{\hat{t}}^T \mathbf{v}_{\hat{t}} - \boldsymbol{\lambda}_{\hat{t}}^T \mathbf{v}_{\hat{t}} = \left(\boldsymbol{\gamma}_{\hat{t}} - \boldsymbol{\lambda}_{\hat{t}}\right)^T \mathbf{v}_{\hat{t}}\,. \tag{212c}$$

Resulting in the condition

$$\forall \boldsymbol{\gamma}_{\hat{t}} \in \mathcal{F}(\mu\lambda_{\hat{n}}) \quad \text{and} \quad \left(\boldsymbol{\gamma}_{\hat{t}} - \boldsymbol{\lambda}_{\hat{t}}\right)^T \mathbf{v}_{\hat{t}} \geq 0\,. \tag{213}$$

Recall, we derived this condition in Section 1.5 from first order optimality conditions, here we just took a more direct geometric approach for deriving this condition. Now a solution to this variational inequality is equivalent to the fixed point of the proximal operator,

$$\boldsymbol{\lambda}_{\hat{t}} = \text{prox}_{\mathcal{F}(\mu\lambda_{\hat{n}})}\left(\boldsymbol{\lambda}_{\hat{t}} - r_{\hat{t}}\mathbf{v}_{\hat{t}}\right) \quad \text{for} \quad r_{\hat{t}} > 0\,. \tag{214}$$

Figure 28 illustrates how the principle of maximum dissipation is connected to the proximal operator, by connecting the variational inequality to the definition of the tangent cone and finally testing if $-\mathbf{v}_{\hat{t}}$ is included in the normal cone at $\boldsymbol{\lambda}_{\hat{t}}$. The tangent and normal cone viewpoint is quite helpful in handling the case of zero sliding velocity. In this case we note that there are no dissipating forces. This means the tangent cone is all directions in the plane and the normal cone contains the zero-vector. This means that any force in the friction cone is a solution to principle of maximum dissipation. Notice that the proximal operator will have this solution too.

Since $\mathbf{v} = \begin{bmatrix} v_{\hat{n}}{}^T \mathbf{v}_{\hat{t}}{}^T \end{bmatrix}^T$ can be written as a linear combination of $\boldsymbol{\lambda} = \begin{bmatrix} \lambda_{\hat{n}}{}^T & \boldsymbol{\lambda}_{\hat{t}}{}^T \end{bmatrix}^T$ we have $\mathbf{v} = \mathbf{A}\boldsymbol{\lambda} + \mathbf{b}$. The equation holds for all values of the variable $r_{\hat{t}}$ but the actual value used will have impact on convergence. Substituting $\mathbf{v}$ into the proximal operator lead to a fixed point problem,

$$\underbrace{\begin{bmatrix} \lambda_{\hat{n}} \\ \boldsymbol{\lambda}_{\hat{t}} \end{bmatrix}}_{\boldsymbol{\lambda}} = \underbrace{\begin{bmatrix} \text{prox}_{\mathcal{N}}\left(\lambda_{\hat{n}} - r_{\hat{n}}\left(\mathbf{A}_{\mathsf{NN}}\lambda_{\hat{n}} + \mathbf{A}_{\mathsf{NF}}\boldsymbol{\lambda}_{\hat{t}} + \mathbf{b}_{\mathsf{N}}\right)\right) \\ \text{prox}_{\mathcal{F}(\mu\lambda_{\hat{n}})}\left(\boldsymbol{\lambda}_{\hat{t}} - r_{\hat{t}}\left(\mathbf{A}_{\mathsf{FN}}\lambda_{\hat{n}} + \mathbf{A}_{\mathsf{FF}}\boldsymbol{\lambda}_{\hat{t}} + \mathbf{b}_{\mathsf{F}}\right)\right) \end{bmatrix}}_{\mathbf{F}(\boldsymbol{\lambda})}\,. \tag{215}$$

Here N and F are index sets used to extract blocks from $\mathbf{A}$ corresponding to normal force variables or friction force variables. Now one could compute the iterates, $\boldsymbol{\lambda}^{k+1} = \mathbf{F}(\boldsymbol{\lambda}^k)$, more details are given in Section 3.4.2. The iteration sequence converges locally if the spectral radius of the Jacobian,

$$\rho\left(\frac{\partial \mathbf{F}(\boldsymbol{\lambda})}{\partial \boldsymbol{\lambda}}\right) < 1\,, \tag{216}$$

remains limited and smaller than one. This requirement can be realized by making a suitable choice of the parameters $r_{\hat{n}}$ and $r_{\hat{t}}$ [Foerg et al. 2006; Niebe 2014; Parikh and Boyd 2014; Studer 2008].

The Newton-Euler equations and kinematic maps are given by

$$\mathbf{M}\dot{\mathbf{u}} = \mathbf{f} + \mathbf{J}^T\boldsymbol{\lambda}\,, \tag{217a}$$

$$\dot{\mathbf{q}} = \mathbf{H}\,\mathbf{u}\,, \tag{217b}$$

where $\mathbf{u}$ is the generalized velocity vector, $\mathbf{q}$ is generalized position vector, $\mathbf{M}$ is the mass matrix, $\mathbf{J}$ is the contact Jacobian, and $\mathbf{f}$ holds external and gyroscopic force terms, see Section 1.10.1 for details. The relative contact velocity is computed from

$$\mathbf{v} = \mathbf{J}\mathbf{u}\,. \tag{218}$$

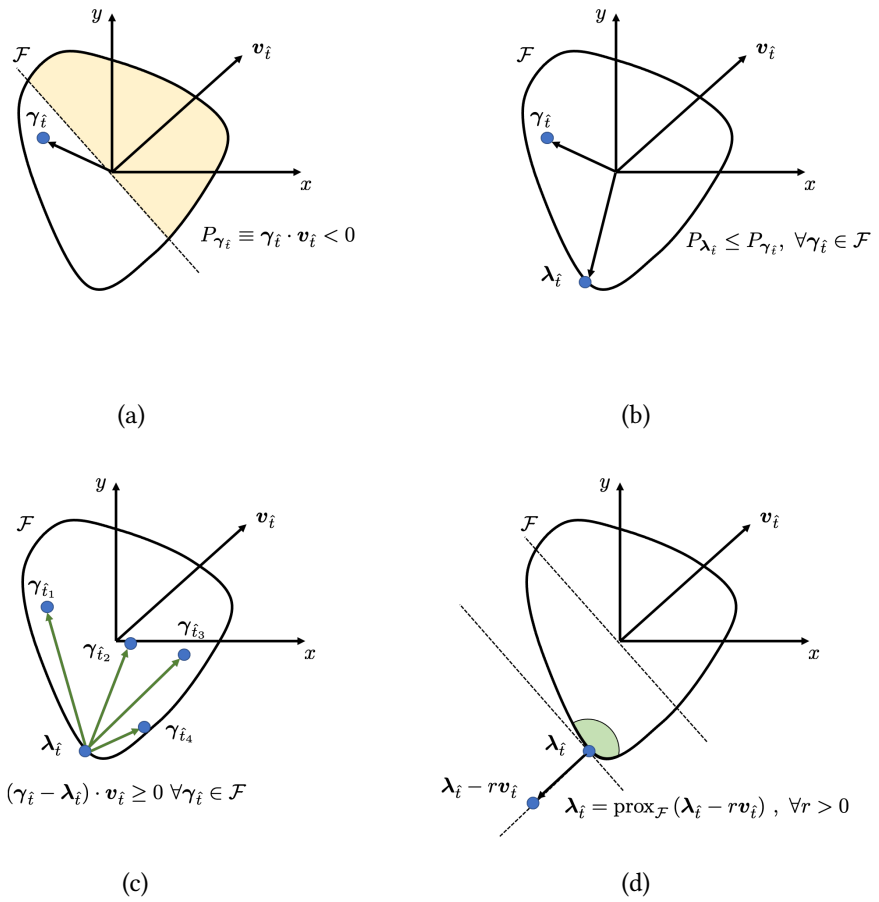


Fig. 28. The proximal operator can be derived from the principle of maximum dissipation. We start with some convex friction cone and assume a non-zero sliding velocity. Given the velocity vector we may look in the whole half-space of dissipating forces (a). In this space we locate the maximum dissipating force as done in (b). In (c) we slightly reformulate the dissipation test to look at the force difference vectors. Any such force difference vector must make a positive dot product with the velocity. That is equivalent to saying that any direction in the tangent cone at $\boldsymbol{\lambda}_{\hat{t}}$ must make a positive product with the velocity. Taking this one step further that means the negative velocity must be in the normal cone at $\boldsymbol{\lambda}_{\hat{t}}$. That is equivalent to say that any point on the half-line from $\boldsymbol{\lambda}_{\hat{t}}$ in the negative velocity direction must have $\boldsymbol{\lambda}_{\hat{t}}$ as the closest point on the surface as shown in (d).

All the contact constraints are now formulated using proximal operators,

$$\forall i \quad \lambda_{\hat{n}i} = \text{prox}_{\mathcal{N}_i}\left(\lambda_{\hat{n}i} - r_{\hat{n}i}\left(v_{\hat{n}_i}^{+} + \varepsilon_{\hat{n}i} v_{\hat{n}_i}^{-}\right)\right), \tag{219a}$$

$$\forall i \quad \boldsymbol{\lambda}_{\hat{t}i} = \text{prox}_{\mathcal{F}_i}\left(\boldsymbol{\lambda}_{\hat{t}i} - r_{\hat{t}i}\left(\mathbf{v}_{\hat{t}i}^{+} + \varepsilon_{\hat{t}i}\mathbf{v}_{\hat{t}i}^{-}\right)\right) \tag{219b}$$

where sub-index i refers the contact point index and

$$\mathcal{N}_i \equiv \{\gamma \in \mathbb{R} \,|\, \gamma \geq 0\} \,, \tag{220a}$$

$$\mathcal{F}_i \equiv \left\{ (\gamma_t, \gamma_b, \gamma_\tau) \in \mathbb{R}^3 \,\middle|\, \left(\frac{\gamma_t}{a}\right)^2 + \left(\frac{\gamma_b}{b}\right)^2 + \left(\frac{\gamma_\tau}{c}\right)^2 \leq 1 \right\} \tag{220b}$$

and

$$a = \mu_{t_i} \lambda_{\hat{n}i}, \quad b = \mu_{b_i} \lambda_{\hat{n}i}, \quad \text{and} \quad c = \mu_{\tau_i} \lambda_{\hat{n}i} \,. \tag{221}$$

Above we exploit the advantage of the proximal operator to work for any convex set and changed $\mathcal{F}$ to be the Coulomb-Contensou friction model, a generalization over the previous definition in Equation 210. We used $\mathbf{v}^-$ and $\mathbf{v}^+$ to denote pre- and post-impact contact velocities. We have also applied a Newton impact law. Usually one has $\varepsilon_{\hat{n}i} \in [0,1]$ and $\varepsilon_{\hat{t}i} = 0$. Further, one may choose $r = r_{\hat{n}i} = r_{\hat{t}i} > 0$. Although other r-factor strategies can be used. Using a time-discretization on the differential equations we have

$$\mathbf{u}^{t+h} = \mathbf{u}^t + h\mathbf{M}^{-1}\mathbf{f} + \mathbf{M}^{-1}\mathbf{J}^T\boldsymbol{\lambda} \,. \tag{222}$$

Both $\mathbf{M}$ and $\mathbf{J}$ depends on $\mathbf{q}$, $\mathbf{f}$ depends on both $\mathbf{q}$ and $\mathbf{u}$ due to gyroscopic force terms. For now we ignore these dependancies and will specify them when we introduce the time-stepping method in Section 3.4.5. The pre-impact contact velocities are given by $\mathbf{v}^- = \mathbf{J}\mathbf{u}^t$ and the post-impact velocities are given by $\mathbf{v}^+ = \mathbf{J}\mathbf{u}^{t+h}$. Multiplying Equation 222 by $\mathbf{J}$ from the left yields,

$$\mathbf{v}^+ = \mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T\boldsymbol{\lambda} + \mathbf{J}\mathbf{u}^t + h\mathbf{J}\mathbf{M}^{-1}\mathbf{f} \,. \tag{223}$$

Defining

$$\mathbf{z} \equiv \boldsymbol{\lambda} - \mathbf{R}\left(\mathbf{v}^+ + \mathbf{E}\mathbf{v}^-\right) \,, \tag{224a}$$

$$= \boldsymbol{\lambda} - \mathbf{R}\left(\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T\boldsymbol{\lambda} + \mathbf{J}\mathbf{u}^t + h\mathbf{J}\mathbf{M}^{-1}\mathbf{f} + \mathbf{E}\mathbf{J}\mathbf{u}^t\right) \tag{224b}$$

where $\mathbf{R}$ and $\mathbf{E}$ are diagonal matrices containing r-factors and ε-coefficients. Now the proximal operators read

$$\forall i \quad \lambda_{\hat{n}i} = \text{prox}_{\mathcal{N}_i}\left(\mathbf{z}_{\mathsf{N}_i}\right) \,, \tag{225a}$$

$$\forall i \quad \boldsymbol{\lambda}_{\hat{t}i} = \text{prox}_{\mathcal{F}_i}\left(\mathbf{z}_{\mathsf{F}_i}\right) \,. \tag{225b}$$

The solutions of these are given by

$$\forall i \quad \lambda_{\hat{n}i} = \max\left(0, \mathbf{z}_{\mathsf{N}_i}\right) \tag{226a}$$

$$\forall i \quad \boldsymbol{\lambda}_{\hat{t}i} = \begin{cases} \mathbf{z}_{\mathsf{F}_i} & ; \mathbf{z}_{\mathsf{F}_i} \in \mathcal{F}_i \\ \arg\min_{\boldsymbol{\gamma} \in \mathcal{F}_i} \left\| \mathbf{z}_{\mathsf{F}_i} - \boldsymbol{\gamma} \right\|^2 & ; \text{otherwise} \end{cases} \,. \tag{226b}$$

The last case computes the closest point in the $\mathcal{F}_i$ to the point $\mathbf{z}_{\mathsf{F}_i}$. This can be solved as described in Section 3.4.3.

3.4.2 Iterative Methods for the Fixed-Point Scheme. A Jacobi-scheme can be used to solve the fixed point problem in Equation 225. This consists of first computing the iterate,

$$\mathbf{z}^k = \boldsymbol{\lambda}^k - \mathbf{R}\left(\underbrace{\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T}_{\mathbf{A}}\boldsymbol{\lambda}^k + \underbrace{\mathbf{J}\mathbf{u}^t + h\mathbf{J}\mathbf{M}^{-1}\mathbf{f} + \mathbf{E}\mathbf{J}\mathbf{u}^t}_{\mathbf{b}}\right), \tag{227a}$$

$$= \boldsymbol{\lambda}^k - \mathbf{R}\left(\mathbf{A}\boldsymbol{\lambda}^k + \mathbf{b}\right). \tag{227b}$$

We solve for the next iterate λ^{k+1} by

$$\forall i \quad \lambda_{\hat{n}_i}^{k+1} = \text{prox}_{\mathcal{N}_i}\left(\mathbf{z}_{\mathsf{N}_i}^k\right), \tag{228a}$$

$$\forall i \quad \boldsymbol{\lambda}_{\hat{t}_i}^{k+1} = \text{prox}_{\mathcal{F}_i^k}\left(\mathbf{z}_{\mathsf{F}_i}^k\right) \tag{228b}$$

, where $\mathcal{F}_i^k$ is defined using the value of the k^{th} iterate, $a = \mu_{t_i}\lambda_{\hat{n}_i}^k$, $b = \mu_{b_i}\lambda_{\hat{n}_i}^k$ and $c = \mu_{\tau_i}\lambda_{\hat{n}_i}^k$. As it stands the scheme is inherently easily parallelized.

A Gauss–Seidel scheme can be created from the Jacobi scheme. The idea is to always use the most updated $\boldsymbol{\lambda}$-values in any computation. Let $\mathbf{z}$ denote the most updated value at all times. The normal and friction solves for the i^{th} contact is now computed using,

$$\lambda_{\hat{n}_i}^{k+1} = \text{prox}_{\mathcal{N}_i}\left(\mathbf{z}_{\mathsf{N}_i}\right), \tag{229a}$$

$$\boldsymbol{\lambda}_{\hat{t}_i}^{k+1} = \text{prox}_{\mathcal{F}_i^{k+1}}\left(\mathbf{z}_{\mathsf{F}_i}\right). \tag{229b}$$

Observe that $\mathcal{F}_i^{k+1}$ is used instead of $\mathcal{F}_i^k$. After having solved for $\lambda_{\hat{n}_i}^{k+1}$ and $\boldsymbol{\lambda}_{\hat{t}_i}^{k+1}$ then all $\mathbf{z}$ dependent entries must be updated before moving on to the next contact. For this update we present the factorization technique from [Erleben 2007]. The idea is to write $\mathbf{z}$ as $\mathbf{z} \equiv \boldsymbol{\lambda} + \mathbf{R}\left(\mathbf{J}\mathbf{w} + \mathbf{b}\right)$ and use $\mathbf{w} \equiv \mathbf{M}^{-1}\mathbf{J}^T\boldsymbol{\lambda}$. For convenience we introduce the index set of all the bodies, B, and the index set of the i^{th} contact point, $\mathsf{I} \equiv \{\mathsf{N}_i, \mathsf{F}_i\}$. Now we can find the most updated value of $\mathbf{z}_\mathsf{I}$ before computing the contact forces,

$$\mathbf{z}_\mathsf{I} = \boldsymbol{\lambda}_\mathsf{I}^k - \mathbf{R}_i\left(\mathbf{J}_{\mathsf{IB}}\,\mathbf{w} + \mathbf{b}_\mathsf{I}\right). \tag{230}$$

After having computed the contact forces we can update $\mathbf{w}$ so its value is ready for the next contact,

$$\mathbf{w} = \left(\mathbf{M}^{-1}\mathbf{J}^T\right)_{\mathsf{BI}}\left(\boldsymbol{\lambda}_\mathsf{I}^{k+1} - \boldsymbol{\lambda}_\mathsf{I}^k\right). \tag{231}$$

We have summarized the complete schemes in Algorithms 11 and 12. We will keep on iterating using the PROX scheme until the residual error has absolutely or relatively converged, or we exceed a maximum iteration count. A residual error can defined as

$$\mathbf{e}^{k+1} = \boldsymbol{\lambda}^{k+1} - \boldsymbol{\lambda}^k. \tag{232}$$

For making an implemetation we are missing two intrinsic parts of the PROX scheme, one is how the closest points on an ellipsoid is computed and the other is how to control the r-factors. The next two subsection will cover these aspects.

Data: K: indices of all contacts, B indices of all bodies, $\mathbf{J}, \mathbf{M}, \mathbf{b}, \mathbf{R}, \boldsymbol{\lambda}^0, \nu$
Result: $\boldsymbol{\lambda}^k$
$\left(k, \boldsymbol{\lambda}^k, \epsilon^k\right) \leftarrow \left(0, \boldsymbol{\lambda}^0, \infty\right)$;
while *not converged* **do**
  $\mathbf{w} \leftarrow \mathbf{M}^{-1}\mathbf{J}^T\boldsymbol{\lambda}^k$;
  $\mathbf{z} \leftarrow \boldsymbol{\lambda}^k - \mathbf{R}\left(\mathbf{J}\,\mathbf{w} + \mathbf{b}\right)$;
  foreach $i \in \mathsf{K}$ **do**
    $\lambda_{\hat{n}i}^{k+1} \leftarrow \text{prox}_{\mathcal{N}_i}\left(\mathbf{z}_{\mathsf{N}_i}\right)$;
    $\boldsymbol{\lambda}_{ti}^{k+1} \leftarrow \text{prox}_{\mathcal{F}_i(\lambda_{\hat{n}i}^k)}\left(\mathbf{z}_{\mathsf{F}_i}\right)$;
  end
  $\epsilon^{k+1} = \left\|\boldsymbol{\lambda}^{k+1} - \boldsymbol{\lambda}^k\right\|_\infty$;
  if $\epsilon^{k+1} > \epsilon^k$ **then**
    $\mathbf{R} \leftarrow \nu\mathbf{R}$;
  else
    $\left(\boldsymbol{\lambda}^k, \epsilon^k, k\right) \leftarrow \left(\boldsymbol{\lambda}^{k+1}, \epsilon^{k+1}, k+1\right)$;
  end
end

Algorithm 11: The PROX Jacobi variant with adaptive r-Factor strategy. For efficiency the matrix product $\mathbf{M}^{-1}\mathbf{J}^T$ may be precomputed and stored in transposed form.

An interesting question to take note of is how good accuracy one can expect from these iterative schemes. This is important when setting the tolerance for absolute convergence. If too aggressive one will never find a solution and if too loose one will never get close enough. Here we present a rough estimate of the upper bound of the expected residual error under the best possible assumptions. Assume the scheme is in the limit of convergence that means that the proximal operator fixed points has been reached. A fixed point means we have

$$\boldsymbol{\lambda}^\star = \boldsymbol{\lambda}^\star - r\,\left(\mathbf{A}\boldsymbol{\lambda}^\star + \mathbf{b}\right)\,.$$

The residual error is defined as the norm of the residual vector

$$\mathbf{e}\left(\boldsymbol{\lambda}^{k+1}\right) = \boldsymbol{\lambda}^{k+1} - \boldsymbol{\lambda}^k + r\,\left(\mathbf{A}\boldsymbol{\lambda}^k + \mathbf{b}\right)\,.$$

In the limit of $k \rightarrow \infty$ and assuming convergence to the closest possible numerical representation $\boldsymbol{\lambda}$ of the fixed point solution $\boldsymbol{\lambda}^\star$ we have $\boldsymbol{\lambda} = (1+\varepsilon)\boldsymbol{\lambda}^\star$ where ε is the machine epsilon

Data: K: indices of all contacts, B indices of all bodies, $\mathbf{J}, \mathbf{M}, \mathbf{b}, \mathbf{R}, \boldsymbol{\lambda}^0, \nu$
Result: $\boldsymbol{\lambda}^k$

$\left(k, \boldsymbol{\lambda}^k, \epsilon^k\right) \leftarrow \left(0, \boldsymbol{\lambda}^0, \infty\right)$;
while *not converged* **do**
  $\mathbf{w} \leftarrow \mathbf{M}^{-1}\mathbf{J}^T\boldsymbol{\lambda}^k$;
  foreach $i \in \mathsf{K}$ **do**
    $\mathsf{I} \equiv$ indices of block $\mathsf{N}_i, \mathsf{F}_i$;
    $\mathbf{z}_\mathsf{I} \leftarrow \boldsymbol{\lambda}^k_\mathsf{I} - \mathbf{R}_i\left(\mathbf{J}_{\mathsf{IB}}\,\mathbf{w} + \mathbf{b}_\mathsf{I}\right)$;
    $\lambda_{\hat{n}_i}^{k+1} \leftarrow \mathrm{prox}_{\mathcal{N}_i}\left(\mathbf{z}_{\mathsf{N}_i}\right)$;
    $\boldsymbol{\lambda}_{\hat{t}_i}^{k+1} \leftarrow \mathrm{prox}_{\mathcal{F}_i(\lambda_{\hat{n}_i}^{k+1})}\left(\mathbf{z}_{\mathsf{F}_i}\right)$;
    $\mathbf{w} \leftarrow \mathbf{w} + \left(\mathbf{M}^{-1}\mathbf{J}^T\right)_{\mathsf{B},\mathsf{I}}\left(\boldsymbol{\lambda}^{k+1}_\mathsf{I} - \boldsymbol{\lambda}^k_\mathsf{I}\right)$;
  end
  $\epsilon^{k+1} = \left\|\boldsymbol{\lambda}^{k+1} - \boldsymbol{\lambda}^k\right\|_\infty$;
  if $\epsilon^{k+1} > \epsilon^k$ **then**
    $\mathbf{R} \leftarrow \nu\mathbf{R}$;
  else
    $\left(\lambda^k, \epsilon^k, k\right) \leftarrow \left(\lambda^{k+1}, \epsilon^{k+1}, k+1\right)$;
  end
end

Algorithm 12: The PROX Gauss–Seidel variant with adaptive r-Factor strategy. The product $\mathbf{M}^{-1}\mathbf{J}^T$ may be precomputed as for the Jacobi variant.

and

$$\|\mathbf{e}\left(\boldsymbol{\lambda}\right)\| \leq r\,\varepsilon\,\|\mathbf{A}\|\left\|\boldsymbol{\lambda}^\star\right\|.$$

Of course $\boldsymbol{\lambda}^\star$ is computational unknown but a conservative upper bound can be computed by tracking the minimum norm of the iterates. The adaptive r-factor strategy guarantees a contraction mapping and hence the bound will continue to improve while iterating. The norm of $\mathbf{A}$ can be estimated too by exploiting the factorization of $\mathbf{A} \equiv \mathbf{J}\mathbf{M}^{-1}\mathbf{J}$ or from computing the spectral radius of the explicit assembly of $\mathbf{A}$ or by applying an iterative method such as Powers method that avoids the need for actual explicit assembly of $\mathbf{A}$.

3.4.3 Closest Point on Ellipsoid. For isotropic friction models or omission of torque effects the ellipsoid collapses to a 2D ellipse and finding the closest point can be done analytically by finding the largest positive real root of a 4^{th} order polynomial. However, we consider the Coulomb-Contensou law. That means our ellipsoid models anisotropic spatial friction with torque effects and corresponds to computing the roots of 6^{th} order polynomial. Hence, we seek

a robust and efficient numerical method. For general convex sets one may use the Minkowsky difference to transform the problem into that of finding the closest point between the origin and a convex set. The GJK algorithm is an example of such an algorithm exploiting these ideas. The advantage of a GJK approach is that it is generally applicable to any friction model as long as the limit surfaces defines a convex set. However, in our case the friction model is an ellipsoid and the generality of GJK is uncalled for. Here we present an approach that exploits the corresponding 6^{th} order polynomial and apply a root finding method to search for the root.

Let $a, b, c > 0$ be given. These parameters defines an ellipsoid surface. The ellipsoid surface consists of all points $\mathbf{x} \in \mathbb{R}^3$ where

$$f(\mathbf{x}) = \mathbf{x}^T \mathbf{K} \mathbf{x} - 1 = 0 \tag{233}$$

and $\mathbf{K}$ is a positive diagonal matrix given by

$$\mathbf{K} = \begin{bmatrix} \frac{1}{a^2} & 0 & 0 \\ 0 & \frac{1}{b^2} & 0 \\ 0 & 0 & \frac{1}{c^2} \end{bmatrix} . \tag{234}$$

Given a point $\mathbf{z} \in \mathbb{R}^3$ outside the ellipsoid, $f(\mathbf{z}) > 0$, then we wish to find the closest point, $\mathbf{x}^\star$ on the ellipsoid surface to $\mathbf{z}$,

$$\mathbf{x}^\star = \arg\min_x \frac{1}{2} \|\mathbf{x} - \mathbf{z}\|^2 \quad \text{subject to} \quad f(\mathbf{x}) = 0 \,. \tag{235}$$

The first order optimality conditions yields

$$\mathbf{x}^\star - \mathbf{z} - \lambda^\star \nabla f(\mathbf{x}^\star) = 0 \,. \tag{236}$$

Using $\nabla f(\mathbf{x}^\star) = 2\mathbf{K}\mathbf{x}^\star$ and $t = -2\lambda^\star$ we have,

$$\mathbf{x}^\star = (t\mathbf{K} + \mathbf{I})^{-1} \mathbf{z} \,. \tag{237}$$

Further, we must have that $\mathbf{x}^\star$ lies on the surface of the ellipsoid,

$$f(\mathbf{x}^\star) = f((t\mathbf{K} + \mathbf{I})^{-1} \mathbf{z}) = 0 \,. \tag{238}$$

We now define this as a scalar function like this,

$$g(t) \equiv f((t\mathbf{K} + \mathbf{I})^{-1} \mathbf{z}) \,, \tag{239}$$

$$= \frac{a^2 z_1^2}{(a^2 + t)^2} + \frac{b^2 z_2^2}{(b^2 + t)^2} + \frac{c^2 z_3^2}{(c^2 + t)^2} - 1 \,. \tag{240}$$

Observe that the problem of finding the closest point has been reformulated into the problem of finding a root of $g(t) = 0$. From geometry, we know that the curve given by $g(t)$ can pierce a non-degenerate ellipsoid in at most two points. If $f(\mathbf{z}) > 0$ then we are seeking the intersection point with $t > 0$. If $f(\mathbf{z}) \leq 0$, then $\mathbf{z}$ is already inside the ellipsoid and we simply return $\mathbf{z}$ as the solution.

For $t > 0$ we have $\frac{d}{dt}g(t) < 0$. However, $g(t)$ is very steep for small t-values and very flat for larger t-values. Thus, a binary search numerical method is a good choice for finding the positive root of $g(t)$. Alternatively a Newton-Raphson method can be applied, although Erleben [2017] found a Newton-Raphson method to diverge or find negative roots when the root approaches the flat parts of the g-function.

An initial bracketing technique is needed for the binary search method. The minimum t-value for the search interval is given by $t_{\text{min}} = 0$ the maximum t-value for the search interval can be estimated as,

$$t_{\text{guess}} = \max\{a, b, c\}\|\mathbf{z}\| \,. \tag{241}$$

The maximum t-value is then given by the minimum non-negative integer, k where $g(t_{\text{max}}) < 0$ and $t_{\text{max}} \equiv \alpha^k t_{\text{guess}}$, where $\alpha > 1$ is the scalar expansion coefficient of the interval. Erleben [2017] uses $\alpha = 1.5$. In order to make the binary search method more robust in the sense of having better precision the problem can be scaled to be within the unit-cube. That is a scaling factor is computed as

$$s = \frac{1}{\max\{1, a, b, c, z_t, z_b, z_\tau\}} \tag{242}$$

and then re-define the problem as follows.

$$\mathbf{z} \leftarrow s\,\mathbf{z}, \quad \text{and} \quad \{a, b, c\} \leftarrow \{s\,a, s\,b, s\,c\} \,. \tag{243}$$

However, when the scheme has converged one must remember to convert the solution back to the unscaled problem. The bisection method is summarized in Algorithm 13.

3.4.4 The r-Factor Strategies. If the r-value is sufficiently low then the fixed point schemes will converge as proven by Foerg et al. [2006]; Studer [2008]. Convergence speed is expected to be worse for small r-values than larger r-values. Let us study a simple example to build intuition about how different r values affect the fixed point search. For simplicity we will study a 2D disk $\mathcal{S}$ defined by the radius $\Delta \in \mathbb{R}_+$. That is

$$\mathcal{S} \equiv \{\boldsymbol{\lambda} \in \mathbb{R}^2 \mid \|\boldsymbol{\lambda}\| \leq \Delta\} \,.$$

Then we iteratively compute

$$\boldsymbol{\lambda}^{k+1} \leftarrow \text{prox}_{\mathcal{S}}\left(\boldsymbol{\lambda}^k - r\mathbf{v}\right) \,.$$

For a fixed given value of r and $\mathbf{v} \equiv \begin{bmatrix}0 & 1\end{bmatrix}^T$ and $\boldsymbol{\lambda}^0 \equiv \begin{bmatrix}\Delta & 0\end{bmatrix}^T$. The results of solving this simple 2D example are shown in Figure 29. One can easily observe the benefit of larger r-values in this simple 2D example. A simple divergence example can be created as well by choosing $\mathcal{S}$ to be the unit disk and solving

$$\boldsymbol{\lambda}^{k+1} \leftarrow \text{prox}_{\mathcal{S}}\left(\boldsymbol{\lambda}^k - r\left(\begin{bmatrix}2 & 1\\ 1 & 2\end{bmatrix}\boldsymbol{\lambda}^k + \begin{bmatrix}-1\\ 1\end{bmatrix}\right)\right) \,.$$

Data: $\{a, b, c\}$: Ellipsoid shape parameters, $\mathbf{z} = \{z_t, z_b, z_\tau\}$: Query point, $\epsilon > 0$: User specified accuracy
Result: $\mathbf{x}$ The closest point on the ellipsoid

// Re-scale problem for better numerical precision
$s \leftarrow \frac{1}{\max(1,a,b,c,|z_t|,|z_b|,|z_\tau|)}$;
$\mathbf{z} \leftarrow s\,\mathbf{z}$;
$\{a, b, c\} \leftarrow \{s\,a, s\,b, s\,c\}$;
// Perform inside ellipsoid test
if $f(\mathbf{z}) < \epsilon$ **then**
  return $\mathbf{z}$;
end
// Perform bracketing for root finding
$t_0 \leftarrow 0$;
$t_1 \leftarrow \max(a, b, c)\ \|\mathbf{c}\|$;
$\alpha \leftarrow 1.5$ // Expansion coefficient, can be tuned
while $g(t_1) > 0$ **do**
  $t_1 \leftarrow \alpha\, t_1$;
end
// Perform root finding and compute closest point
$t^* \leftarrow \mathit{BisectionRootSearch}(g, t_0, t_1, \epsilon)$;
$t^* \leftarrow \frac{t^*}{s^2}$;
$\mathbf{x} \leftarrow t^* \left(\mathbf{K} + \mathbf{I}\right)^{-1} \mathbf{z}$;
return $\mathbf{x}$;

Algorithm 13: Numerical-Ellipsoid-Solver: Iterative numerical root finding method for computing the closest point on an ellipsoid.

Using $\boldsymbol{\lambda}^0 \equiv \begin{bmatrix} 0 & 1 \end{bmatrix}^T$ results in the behaviors seen in Figure 30. Here we see the benefit of increasing the r-value. The iterations in this example are illustrated in Figure 31.

We may apply an adaptive back-tracking approach for adjusting the r-values to auto-tune the r-value for better convergence behavior while solving for the fixed point of the proximal operator. Regardless of chosen strategy, if we detect divergence while iterating, ie. the norm of the residual $\left\|\mathbf{e}^{k+1}\right\| > \left\|\mathbf{e}^{k}\right\|$ then we drop updating $\boldsymbol{\lambda}^{k+1}$. Instead we roll-back to $\boldsymbol{\lambda}^k$ and reduce all r-values by a user specified fraction ν. We will next present three different strategies for adjusting the r-factors: global, local and blocked strategies.

For the global r-factor strategy we simply initialize all r-values of all constraints to a single global constant value. Ideally based on theory by Foerg et al. [2006] one should initialize this

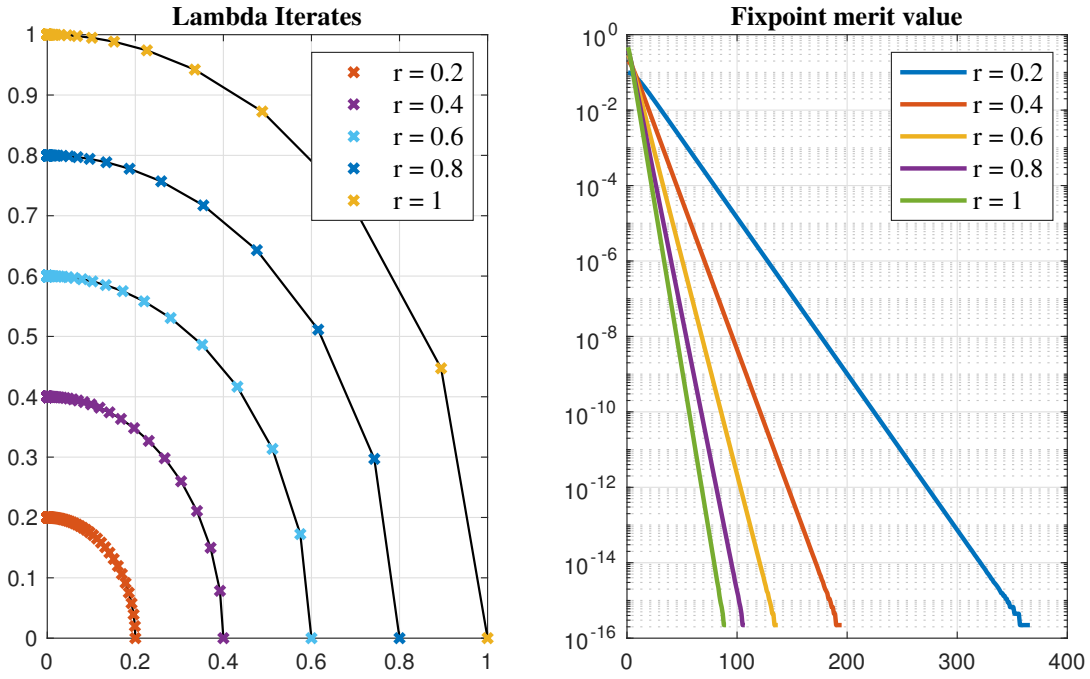


Fig. 29. Left figure illustrates the iterates of solving the 2D problem $\lambda = \text{prox}_{\mathcal{S}}(\lambda - r\mathbf{v})$ where $\mathcal{S}$ is a 2D disk for different r-values. The iterates have been re-scaled by r/Δ to separate them into distinct curves for better comparison. On the right the corresponding convergence is displayed. Observe that the linear convergence rate constant depends on the r-value. Notice that larger r-values results in a smaller constant.

to the reciprocal of the sum of the absolute values of the minimum and maximum eigenvalues of $\mathbf{A}$. However, these are not easily obtained.

For the local r-factor strategy one would initially choose the r-value of the i^{th} variable to be

$$r_i = \frac{1}{\mathbf{A}_{ii}}. \tag{244}$$

Comparing this choice to the algebraic form of the variable updates used in PGS and PSOR we immediately recognize this as using a PGS-strategy for setting the initial value. If one used $r_i = \frac{\omega}{\mathbf{A}_{ii}}$ where ω is the successive over-relaxation coefficient then the PSOR variant is achieved, see Section 3.2 for derivation of PGS and PSOR. Hence, one may replace $\mathcal{F}_i$ with a box-model and keeping local r-values as fixed constants and the PROX scheme will deteriorate into the well known PGS/PSOR variants used in many physics engines.

For the blocked strategy the idea is to replace the scalar r-values with small blocked matrices build from the diagonal blocks of the $\mathbf{A}$-matrix. Let us consider the k^{th} block corresponding to

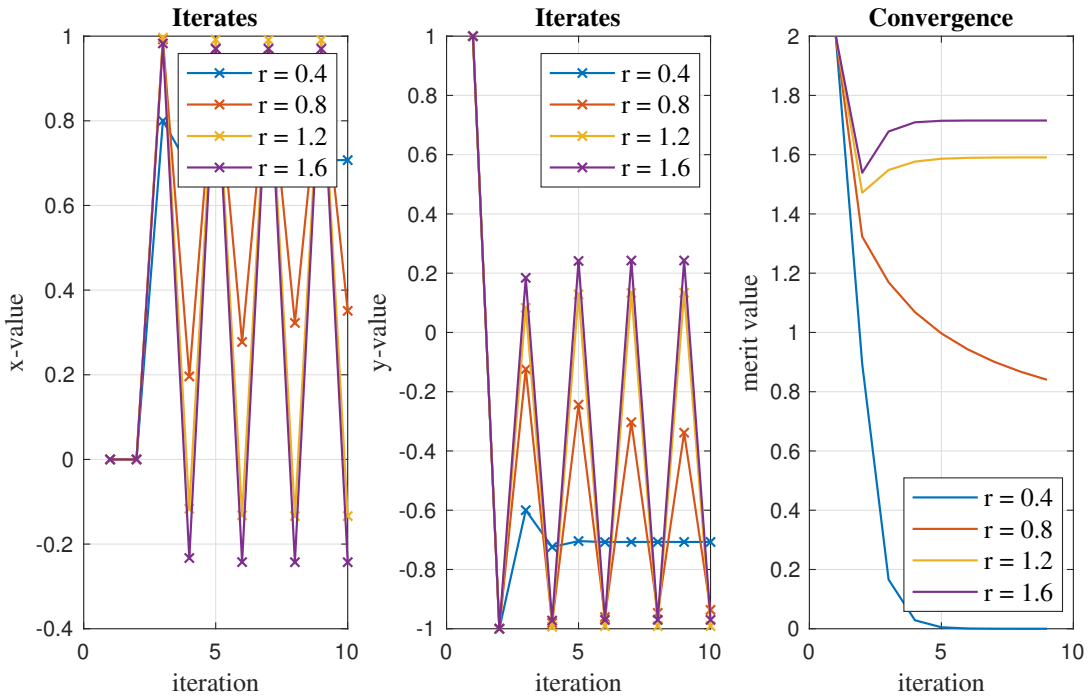


Fig. 30. Left figure illustrates the iterates of solving the 2D problem $\lambda = \text{prox}_{\mathcal{S}}\left(\lambda - r\left(\mathbf{A}\lambda + \mathbf{b}\right)\right)$ where $\mathcal{S}$ is a 2D unit disk for different r-values. As one may observe the convergence behavior gets worse for larger r-values in that the iterates starts to oscillate wildly.

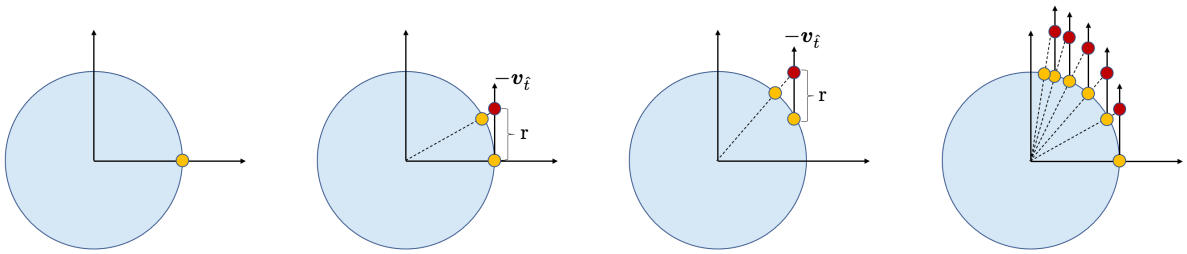


Fig. 31. From left to right: the starting iterate, the effect of the first iteration, the second iteration, and finally a sequence of operations. Observe how the r value always "push" the iterate the same distance in the $-\mathbf{v}$ direction. Increase r-value makes the iterations go faster to the north pole of the disk.

the k^{th} contact, then the r-factor strategy uses the block $\mathbf{R}_k \in \mathbb{R}^{4\times 4}$,

$$\mathbf{R}_k = \begin{bmatrix} \frac{1}{\mathrm{A}_{ii}} & 0 \\ 0 & \mathbf{A}^{-1}_{t:\tau,t:\tau} \end{bmatrix}, \tag{245}$$

where $n = 4k$, $t = n+1$, $b = n+2$, $\tau = n+3$. The blocked strategy has been reported by Merlhiot [2007] to work quite well.

For the global and blocked strategies one may use $\nu = 0.5$ and for the local strategy $\nu = 0.9$. These values are not critical but was tuned experimentally and reported by Erleben [2017]. Note r-factors are not the same as numerical damping or regularization as known from LCP methods shown in Cottle et al. [1992]. For those the coefficient matrix is changed like $\mathbf{A} \leftarrow \mathbf{I}\rho + \mathbf{A}$ for some damping parameter $\rho > 0$. This changes the model and adds compliance to the contact. The r-factors does not change the model, the solutions hold for all positive r-values. The specific r-value only affect the convergence constants. Erleben [2017] reports that local r-value strategy combined with Gauss-Seidel type variant of PROX gives more predictable performance in most cases, and that the global strategy for Jacobi scheme has interesting capabilities for dealing with large structured stacks like masonry structures.

3.4.5 Time-Stepping Methods. The PROX schemes can be used with different time-stepping methods. In this presentation we favor the mid-point scheme of Moreau [1999]. It consists of using an explicit half-step position update

$$\mathbf{q}^{t+\frac{1}{2}} = \mathbf{q}^{t} + \frac{h}{2}\mathbf{H}\,\mathbf{u}^{t}\,. \tag{246}$$

Then one computes $\mathbf{M} = \mathbf{M}(\mathbf{q}^{t+\frac{1}{2}})$, $\mathbf{J} = \mathbf{J}(\mathbf{q}^{t+\frac{1}{2}})$ and $\mathbf{f} = \mathbf{f}(\mathbf{q}^{t+\frac{1}{2}}, \mathbf{u}^{t})$ and solves

$$\lambda = \text{PROX-SOLVER}\left(\mathbf{q}^{t+\frac{1}{2}}, \mathbf{u}^{t}, h, \ldots\right), \tag{247a}$$

$$\mathbf{u}^{t+1} = u^{t} + h\mathbf{M}^{-1}\mathbf{f} + M^{-1}\mathbf{J}^{T}\lambda, \tag{247b}$$

$$\mathbf{q}^{t+1} = \mathbf{q}^{t+\frac{1}{2}} + \frac{h}{2}\mathbf{u}^{t+1}, \tag{247c}$$

where PROX-SOLVER $(\ldots)$ denotes invocation of the PROX scheme developed above, see Algorithm 11 and 12. Previous work on interactive simulation by Erleben [2007] have used a simple semi-implicit scheme . Here we favor the Moreau-variant due to the mid-point evaluation adds some "softness" into how contact is detected compared to the simpler semi-implicit scheme.

3.4.6 Constraint Stabilization by Post-step Projection. The velocity based formulation solves constraints on a velocity-level, and this means that numerical drift will occur for position-level constraints. Regardless of how many iterations are used, there will also be some inaccuracy in the constraint impulses.

Stabilization can be used to reduce constraint errors. For instance, Baumgarte stabilization adds a penalty term to the kinematic constraints, and a spring-based version of this stabilization technique was presented in Section 1.9. Alternatively, the approach we derive here is similar in spirit to the one proposed by Baraff [1993] and Cline and Pai [2003]. It uses a position-level

update as a post-step process to resolve the constraint errors, and we further demonstrate how this can be formulated using the prox operator.

Consider a gap vector $\boldsymbol{\phi}$ containing penetration errors for all contacts at the current time step. We can compute a gap-displacement that corrects constraint errors as

$$\Delta\boldsymbol{\phi} = \boldsymbol{\phi}^* - \boldsymbol{\phi}\,,$$

where $\boldsymbol{\phi}^*$ are the corrected gap values without constraint violation. Given a finite displacement of the bodies, $\Delta\mathbf{q} = \mathbf{q}^* - \mathbf{q}$, the first order relationship mapping body displacements to changes in the gap values can be written as

$$\Delta\mathbf{g} = \mathbf{J}\Delta\mathbf{q}\,. \tag{248}$$

Observe that here we write the gap displacements using a vector notation. However, if we only want to correct the penetration error for one contact, then the gap displacement vector would reduce to the scalar ϕ.

From first-order physics, we have the equation of motion

$$\mathbf{M}\Delta\mathbf{q} = \mathbf{J}^T\mathbf{d}\,. \tag{249}$$

In this formulation, $\mathbf{d}$ are the contact displacements and $\mathbf{J}^T$ transforms the contact displacements into body-displacements. The body-displacements are also weighted by the mass matrix $\mathbf{M}$, and this has the effect that linear contact displacements are distributed in a physical plausible way as linear and rotational displacements of the bodies. From the above equation of motion, we can write the body-displacements as

$$\Delta\mathbf{q} = \mathbf{M}^{-1}\mathbf{J}^T\mathbf{d}\,. \tag{250}$$

Next, we substitute the expression into our kinematic gap-equation:

$$\Delta\boldsymbol{\phi} = \mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T\mathbf{d} \tag{251}$$

We seek a body displacement such that the gap-function becomes non-negative, such that

$$\boldsymbol{\phi}^* = (\Delta\boldsymbol{\phi} + \boldsymbol{\phi}) \geq 0\,. \tag{252}$$

Recall that contact displacements can only "push" bodies apart, which means that $\mathbf{d} \geq 0$, and furthermore we cannot have a contact displacement at a separated contact. Thus, we wish to find a body displacement such that

$$(\Delta\boldsymbol{\phi} + \boldsymbol{\phi}) \geq 0,\ \mathbf{d} \geq 0 \quad \text{and} \quad \mathbf{d}^T\,(\Delta\boldsymbol{\phi} + \boldsymbol{\phi}) = 0\,. \tag{253}$$

Combining these yields

$$\left(\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T\mathbf{d} + \boldsymbol{\phi}\right) \geq 0,\ \mathbf{d} \geq 0 \quad \text{and} \quad \mathbf{d}^T\left(\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T\mathbf{d} + \boldsymbol{\phi}\right) = 0\,, \tag{254}$$

which is a linear complementarity problem. This can be re-cast as a proximal operator model

$$\mathbf{d} = \text{prox}_{\mathcal{N}}\left(\mathbf{d} - r\left(\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T\mathbf{d} + \boldsymbol{\phi}\right)\right) \tag{255}$$

for all $r > 0$ where $\mathcal{N} = \mathcal{N}_1 \times \cdots \times \mathcal{N}_n$. Having solved for $\mathbf{d}$ a position update can be performed.

$$\mathbf{q}^* = \mathbf{q} + \mathbf{H}(\mathbf{q})\mathbf{M}^{-1}\mathbf{J}^T\mathbf{d}\,. \tag{256}$$

Recall that $\mathbf{H}(\mathbf{q})$ is the kinematic mapping between velocities and positions. The above equation can be seen as computing an instantaneous change of the body positions disregarding velocities, external forces or velocity dependent forces. The scheme is of course based on a linearization of the true gap-function. Therefore, it may not be able to resolve the errors with a single step and quite often several steps must be taken.

4 SOFT BODY CONTACT APPROACHES

In Section 1.10, we introduced the first concepts and notation needed to understand how to deal with solving contact for soft bodies compared to rigid bodies. The focus was on how to assemble the system of ordinary differential equations that explain the dynamics of soft bodies due to elastic deformations, material damping and contact forces. This resulted in Equation 76, which we then combined with an implicit time-discretization approach to derive a complementarity model for soft body contact.

In this chapter, we will rewind and elaborate further into several of the typical techniques used for dealing with time-discretizations of soft body contact, as well as the numerical methods that have dominated soft body dynamics solvers. Hence, our goal is to provide a more nuanced overview of simulating soft body contact, as well to dive deeper into some of the limitations that pop-up when shifting from rigid to soft bodies.

4.1 Equations of Motion for a Collection of Soft Bodies

We begin by restating the differential equations that govern soft body motion:

$$\mathbf{M}\ddot{\mathbf{q}} = \mathbf{f}_{\text{ext}} + \mathbf{f}_{\text{d}}(\mathbf{u}) + \mathbf{f}_{\text{e}}(\mathbf{q}) + \mathbf{f}_{\text{c}}(\mathbf{q}, \mathbf{u}) \tag{257}$$

Here, $\mathbf{q}$ is the concatenation of the positions of all mesh nodes, $\mathbf{u}$ are the velocities of those nodes, and $\mathbf{f}_{\text{ext}}$ are the external forces acting on the nodes. The matrix $\mathbf{M}$ is the mass matrix of the soft bodies, $\mathbf{f}_{\text{d}}(\mathbf{u})$ are damping forces, $\mathbf{f}_{\text{e}}(\mathbf{q})$ are the internal elastic forces, and $\mathbf{f}_{\text{c}}(\mathbf{q}, \mathbf{u})$ gives the contact forces between soft bodies (or the same body in the case of self-collision).

We next make an important conceptual change about how the simulation views multiple soft bodies. Consider that there are N nodes in a multi-body system, and assume that each body is modeled using a tetrahedral mesh, which is a common choice for computer graphics. We can then assemble all tetrahedral elements of all the soft bodies into one single global tetrahedral mesh, and thus the dimensions of the introduced quantities are $\mathbf{q}, \mathbf{u}, \mathbf{f}_{\text{ext}}, \mathbf{f}_{\text{e}}, \mathbf{f}_{\text{d}}, \mathbf{f}_{\text{c}} \in \mathbb{R}^{3N}$ and $\mathbf{M} \in \mathbb{R}^{3N \times 3N}$. This is convenient as there is really no need to think of a collection of individual local volume meshes (i.e., one for each soft body). From a numerical viewpoint, the simulation is now just a big soup of volume elements, with some elements being connected and others are not. From an implementation viewpoint, there are also benefits since soft-bodies are simply subsets of the global state. Therefore, there are memory footprint and computational savings since information does not have to be copied back-and-forth between "global" and "local" data structures before and after each time-step.

Having merged all bodies into one global volume mesh, let us revisit how to assemble the contact Jacobian when using a constraint-based approach. That is we have $\mathbf{f}_{\text{c}}(\mathbf{q}, \mathbf{u}) \equiv \mathbf{J}(\mathbf{q})^T \boldsymbol{\lambda}$ where $\boldsymbol{\lambda}$ contains the Lagrange multipliers for the contact forces. The Lagrange multiplier could depend on $\mathbf{u}$ if friction forces are included. Here we keep notation simplified to not over-clutter the presentation. Much of what we will present in this chapter holds for both

non-penetration and friction forces, and any of the models and constraint formulations we have presented elsewhere can in principle be used for setting up a model to solve for $\boldsymbol{\lambda}$.

In Section 1.4.2, we derived the contact Jacobian for the case of non-interpenetration at a single contact point defined by the node of one tetrahedral element colliding with a second element. The barycentric coordinates of the tetrahedral element that embed the contact point are used to setup the Jacobian. The same recipe can now be used in a more general setting.

Let $\mathbf{p} \in \mathbb{R}^3$ be the position of a contact point occurring between two tetrahedral elements. Let one element be denoted by A with nodal indices i, j, k and m, and the other element be denoted by B with nodal indices n, o, p, and r. Using barycentric coordinates, we can write the position of a contact point as a weighted sum of the nodal positions of element A:

$$\mathbf{p}_A = w_i\,\mathbf{x}_i + w_j\,\mathbf{x}_j + w_k\,\mathbf{x}_k + w_m\,\mathbf{x}_m$$

Here, $\mathbf{x}_{\{i,j,k,m\}}$ are the nodal positions, and $w_{\{i,j,k,m\}}$ are the barycentric coordinates, with subscripts indicating the corresponding node index. A similar expression can be written for the contact point using element B's nodal positions as follows:

$$\mathbf{p}_B = w_n\,\mathbf{x}_n + w_o\,\mathbf{x}_o + w_p\,\mathbf{x}_p + w_r\,\mathbf{x}_r$$

Obviously, at the time of contact, we have $\mathbf{p} = \mathbf{p}_A = \mathbf{p}_B$. The relative velocity at the contact point is then given by

$$\Delta\mathbf{v} = \left(\frac{\partial \mathbf{p}^B}{\partial t} - \frac{\partial \mathbf{p}^A}{\partial t}\right).$$

However, it is more convenient to express the relative velocity in the contact space, which is achieved by projecting $\Delta\mathbf{v}$ onto the vectors spanning the local coordinate frame, such that

$$\mathbf{v} = \underbrace{\begin{bmatrix}\hat{n} & \hat{t} & \hat{b}\end{bmatrix}^T}_{\mathbf{C}^T} \Delta\mathbf{v}\,.$$

Since the barycentric coordinates do not depend on time, we rewrite the above equation as:

$$\mathbf{v} = \underbrace{\mathbf{C}^T \begin{bmatrix} -w_i\mathbf{I}_{3\times3} & -w_j\mathbf{I}_{3\times3} & -w_k\mathbf{I}_{3\times3} & -w_m\mathbf{I}_{3\times3} & w_o\mathbf{I}_{3\times3} & w_p\mathbf{I}_{3\times3} & w_q\mathbf{I}_{3\times3} & w_r\mathbf{I}_{3\times3} \end{bmatrix}}_{\mathbf{J}} \underbrace{\begin{bmatrix} \mathbf{v}_i \\ \mathbf{v}_j \\ \mathbf{v}_k \\ \mathbf{v}_m \\ \mathbf{v}_o \\ \mathbf{v}_p \\ \mathbf{v}_q \\ \mathbf{v}_r \end{bmatrix}}_{\mathbf{u}}.$$

Here we have the nodal velocities given by sub-scripted $\mathbf{v}$'s. Equivalently we can write $\mathbf{J}$ for a single contact between colliding elements as:

$$\mathbf{J} = \begin{bmatrix} -w_i\mathbf{C}^T & -w_j\mathbf{C}^T & -w_k\mathbf{C}^T & -w_m\mathbf{C}^T & w_o\mathbf{C}^T & w_p\mathbf{C}^T & w_q\mathbf{C}^T & w_r\mathbf{C}^T \end{bmatrix} \tag{258}$$

This form is closer to how one would implement the assembly of the $\mathbf{J}$ matrix. If there are multiple contact points and multiple tetrahedral elements, then Equation 258 is repeated for each contact. The global matrix $\mathbf{J}$ for all contacts and all elements can then be assembled in a sparse block-wise fashion, where for each contact a 3-by-3 block is added to the global matrix corresponding to each of the vertices of elements involved in the contact. In fact, there could be multiple contacts between two given tetrahedral elements, in which case the barycentric coordinates would be different in the corresponding block-rows of $\mathbf{J}$, but the fill pattern would be the same. Hence, if there are K contacts, then the global Jacobian matrix has dimensions $\mathbf{J} \in \mathbb{R}^{3K\times3N}$ and the vector of constraint impulses is $\boldsymbol{\lambda} \in \mathbb{R}^{3K}$. Algorithm 14 shows how to perform the assembly given a collection of K contact points.

4.2 Approaches to Time Discretization

The semi-implicit first-order time discretization from Section 1.2 is a popular scheme for fast and interactive rigid body simulators. However, for soft bodies with stiff elasticity, such schemes are often avoided as they limit the time-step sizes. Rather, implicit schemes are sought, and in this section we embark on explaining these variations that are common to soft body contact simulations [Baraff and Witkin 1998].

There are two general approaches for time stepping soft body simulations. One approach is based on deriving a non-smooth root finding problem, where solving for the roots yields the time-updated state vectors. The second approach is based on formulating an optimization problem for minimizing an energy potential, and a first-order optimizer is used to produce a solution of the time-updated state vectors. The minimization approach can be further broken

Data: Collection of all contacts $\mathcal{K}$, numbers of contacts K, and number of nodes N
Result: Contact Jacobian matrix $\mathbf{J}$.

$\mathbf{J} \leftarrow$ create a $K \times N$ blocked sparse matrix 3×3 blocks;
foreach $c \in \mathcal{K}$ **do**
  $\hat{n}, \hat{t}, \hat{b} \equiv$ contact normal, tangent and binormal contact c;
  C $\equiv$ block row index for contact c;
  I, J, K, M, O, P, Q, R $\equiv$ block column indices for vertices of contact c;
  $w_i, w_j, w_k, w_m, w_o, w_p, w_q, w_r \equiv$ barycentric coordinates of contact c;
  $\mathbf{C} \leftarrow \begin{bmatrix} \hat{n} & \hat{t} & \hat{b} \end{bmatrix}$;
  $\mathbf{J}_{\mathrm{C,I}} \leftarrow -w_i \mathbf{C}^T$;
  $\mathbf{J}_{\mathrm{C,J}} \leftarrow -w_j \mathbf{C}^T$;
  $\mathbf{J}_{\mathrm{C,K}} \leftarrow -w_k \mathbf{C}^T$;
  $\mathbf{J}_{\mathrm{C,M}} \leftarrow -w_m \mathbf{C}^T$;
  $\mathbf{J}_{\mathrm{C,O}} \leftarrow \ \ w_o \mathbf{C}^T$;
  $\mathbf{J}_{\mathrm{C,P}} \leftarrow \ \ w_p \mathbf{C}^T$;
  $\mathbf{J}_{\mathrm{C,Q}} \leftarrow \ \ w_q \mathbf{C}^T$;
  $\mathbf{J}_{\mathrm{C,R}} \leftarrow \ \ w_r \mathbf{C}^T$;
end
return $\mathbf{J}$

Algorithm 14: AssembleSoftBodyJacobian Assembly of the contact Jacobian for soft bodies is accomplished by iterating over contacts and filling in $\mathbf{J}$ row-by-row. Each row block has exactly 8 non-zero column blocks, for a total of 24 non-zero values for each row. Hence, $\mathbf{J}$ is often quite sparse for soft bodies since the degrees of freedom $3N$ is typically larger than $3K$.

down into (i) position-level and (ii) velocity-level approaches. We briefly derive both variants below to illustrate their differences.

4.2.1 Root-finding Approach. Starting from the equations of motion, we rewrite the second-order ODE in Equation 257 into a system of coupled first-order ODEs:

$$\mathbf{M}\dot{\mathbf{u}} = \mathbf{f}_{\text{ext}} + \mathbf{f}_{\text{d}}(\mathbf{u}) + \mathbf{f}_{\text{e}}(\mathbf{q}) + \mathbf{f}_{\text{c}}(\mathbf{q}, \mathbf{u}), \tag{259a}$$

$$\dot{\mathbf{q}} = \mathbf{u} \tag{259b}$$

This avoids difficulties related to the treatment of higher order time derivatives. There are many different ways of deriving the specific root-finding formulation that will give us a time-update. One way to present the generic recipe for the derivation is as follows:

(1) Write the full system of equations. There are two main forms: the primary form and the dual form.
(2) Apply finite difference approximations to all time-derivatives to derive the root-finding function. The finite difference approximation originate from the chosen time-integration method. However, below we show only the derivation for implicit Euler integration, although this recipe will work for other integration schemes.
(3) Derive the equation for Newton's method by substituting first-order Taylor approximations and then reformulate using matrix notation. This results in a linear system. In practice one may solve this linear system in many different ways (e.g., applying Schur complement reduction techniques, preconditioners, or iterative solvers). We refer you to other chapters for more details about numerical solver methods.

The above steps give the recipe for assembling the Newton equations, and iteratively assembling and solving this equation yields a final approximate solution for time stepping the simulation.

Primary Form. Equation 259 is not necessarily complete in its current form, since it also depends on the contact model. However, for the moment, we consider that the contact forces are the negative gradient of some potential function, $\mathcal{E}_c$, whose specific algebraic form depends on the chosen contact model. We may now write $\mathbf{f}_c(\mathbf{q},\mathbf{u}) \equiv -\nabla_{\mathbf{q}}\mathcal{E}_c(\mathbf{q},\mathbf{u})$.

Next, the equations of motion are discretized and rewritten using the implicit Euler method, giving

$$\mathbf{M}\frac{\mathbf{u}^{t+h}-\mathbf{u}^{t}}{h} = \mathbf{f}_{\text{ext}} + \mathbf{f}_{\text{d}}(\mathbf{u}^{t+h}) + \mathbf{f}_{\text{e}}(\mathbf{q}^{t+h}) + \mathbf{f}_{\text{c}}(\mathbf{q}^{t+h},\mathbf{u}^{t+h}) \tag{260a}$$

$$\frac{\mathbf{q}^{t+h}-\mathbf{q}^{t}}{h} = \mathbf{u}^{t+h}, \tag{260b}$$

where superscripts denote the time that a physical quantity is evaluated, and quantities without a superscript are considered time independent. A slight reorganization of Equation 260 gives:

$$\underbrace{\begin{bmatrix} \mathbf{M}\,\mathbf{u}^{t+h} - h\,\mathbf{f}_{\text{d}}(\mathbf{u}^{t+h}) - h\,\mathbf{f}_{\text{e}}(\mathbf{q}^{t+h}) - h\,\mathbf{f}_{\text{c}}(\mathbf{q}^{t+h},\mathbf{u}^{t+h}) - \mathbf{M}\,\mathbf{u}^{t} - h\,\mathbf{f}_{\text{ext}} \\ \mathbf{q}^{t+h} - h\,\mathbf{u}^{t+h} - \mathbf{q}^{t} \end{bmatrix}}_{\equiv \mathbf{F}_{\text{primary}}} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}. \tag{261}$$

Having defined this vector function $\mathbf{F}_{\text{primary}}$, we derive the problem of computing the time-updated values $\mathbf{q}^{t+h}$ and $\mathbf{u}^{t+h}$ as a root-finding problem:

$$\text{compute } \mathbf{q}^{t+h} \text{ and } \mathbf{u}^{t+h}$$
$$\text{s.t. } \mathbf{F}_{\text{primary}}(\mathbf{q}^{t+h},\mathbf{u}^{t+h}) = \mathbf{0}$$

Moving forward, we remove the superscripts and simply say that we are searching for $\mathbf{q}$ and $\mathbf{u}$ where $\mathbf{F}_{\text{primary}}(\mathbf{q},\mathbf{u}) = \mathbf{0}$. Such a root-finding problem can be solved by applying Newton's

method. Assuming values $\mathbf{q}^k$ and $\mathbf{u}^k$ at the k^{th}-iterate are known, we seek updates $\Delta\mathbf{q}$ and $\Delta\mathbf{u}$:

$$\mathbf{q}^{k+1} \leftarrow \mathbf{q}^k + \Delta\mathbf{q} \tag{262a}$$

$$\mathbf{u}^{k+1} \leftarrow \mathbf{u}^k + \Delta\mathbf{u} \tag{262b}$$

We omit details of a line-search, but this can be incorporated by adding a step-length parameter to above equations. The first-order Taylor approximations of the terms in Equation 261 can thus be written as:

$$\begin{aligned}
\mathbf{M}\,\mathbf{u}^{k+1} &\approx \mathbf{M}\,\mathbf{u}^k + \mathbf{M}\,\Delta\mathbf{u} \\
h\,\mathbf{f}_\mathrm{d}(\mathbf{u}^{k+1}) &\approx h\,\mathbf{f}_\mathrm{d}(\mathbf{u}^k) + h\,\mathbf{B}^k\,\Delta\mathbf{u} \\
h\,\mathbf{f}_\mathrm{e}(\mathbf{q}^{k+1}) &\approx h\,\mathbf{f}_\mathrm{d}(\mathbf{q}^k) + h\,\mathbf{K}^k\,\Delta\mathbf{q} \\
h\,\mathbf{f}_\mathrm{c}(\mathbf{q}^{k+1}, \mathbf{u}^{k+1}) &\approx h\,\mathbf{f}_\mathrm{c}(\mathbf{q}^k, \mathbf{u}^k) - h\,\mathbf{H}_\mathbf{q}^k\,\Delta\mathbf{q} - h\,\mathbf{H}_\mathbf{u}^k\,\Delta\mathbf{u}
\end{aligned}$$

Note that $\mathbf{B}^k$ and $\mathbf{K}^k$ are the well-known damping matrix and the stiffness matrix [Kim and Eberle 2020; Sifakis and Barbic 2012]:

$$\mathbf{B}^k \equiv \frac{\partial \mathbf{f}_\mathrm{d}(\mathbf{u}^k)}{\partial \mathbf{u}}, \mathbf{K}^k \equiv \frac{\partial \mathbf{f}_\mathrm{e}(\mathbf{q}^k)}{\partial \mathbf{q}}, \mathbf{H}_\mathbf{q}^k \equiv \frac{\partial^2 \mathcal{E}_\mathrm{c}(\mathbf{q}^k, \mathbf{u}^k)}{\partial \mathbf{q}^2}, \text{ and } \quad \mathbf{H}_\mathbf{u}^k \equiv \frac{\partial^2 \mathcal{E}_\mathrm{c}(\mathbf{q}^k, \mathbf{u}^k)}{\partial \mathbf{q}\partial \mathbf{u}}.$$

Combining the approximations above with $\mathbf{F}_\text{primary}$ and adopting a short-hand notation of the super-script denoting time or iterate where a term is evaluated, this gives the set of equations:

$$\begin{bmatrix} \left(\mathbf{M} - h\,\mathbf{B}^k + h\,\mathbf{H}_\mathbf{u}^k\right)\,\Delta\mathbf{u} - h\,\left(\mathbf{H}_\mathbf{q}^k - \mathbf{K}^k\right)\Delta\mathbf{q} \\ -h\,\Delta\mathbf{u} + \Delta\mathbf{q} \end{bmatrix} = \begin{bmatrix} \mathbf{M}\,\left(\mathbf{u}^t - \mathbf{u}^k\right) + h\,\left(\mathbf{f}_\mathrm{d}^k + \mathbf{f}_\mathrm{e}^k + \mathbf{f}_\mathrm{c}^k + \mathbf{f}_\text{ext}\right) \\ \mathbf{q}^t - \mathbf{q}^k + h\mathbf{u}^k \end{bmatrix}$$

Furthermore, it can be shown that $\Delta\mathbf{q} = h\,\Delta\mathbf{u}$. We use this relation to further simplify the system:

$$\begin{bmatrix} \left(\mathbf{M} - h\,\mathbf{B}^k - h^2\,K^k + h\,\mathbf{H}_\mathbf{u}^k + h^2\,\mathbf{H}_\mathbf{q}^k\right)\,\Delta\mathbf{u} \\ -h\,\Delta\mathbf{u} + \Delta\mathbf{q} \end{bmatrix} = \begin{bmatrix} \mathbf{M}\,\left(\mathbf{u}^t - \mathbf{u}^k\right) + h\,\left(\mathbf{f}_\mathrm{d}^k + \mathbf{f}_\mathrm{e}^k + \mathbf{f}_\mathrm{c}^k + \mathbf{f}_\text{ext}\right) \\ \mathbf{q}^t - \mathbf{q}^k + h\mathbf{u}^k \end{bmatrix}$$

A solution can thus be obtained by solving the first row for $\Delta\mathbf{u}$, then substituting this solution into the second row and solving for $\Delta\mathbf{q}$. Finally, the solution of the current iterate is updated using Equation 262.

There are cases where the terms $\mathbf{H}_\mathbf{q}$ and $\mathbf{H}_\mathbf{u}$ may be simplified. For instance, if no damping is present in the normal forces and a "lagging" model is used to approximate the friction directions, then the velocity dependent term can be set to zero. Also, using a diagonal approximation of $\mathbf{H}_\mathbf{q}$ gives rise to another type of approach that is similar to adding geometric stiffness of the contacts.

Dual Form. For constraint-based approaches, then the contact forces can be computed as Langrange multipliers, $\mathbf{f}_c \equiv \mathbf{J}^T\boldsymbol{\lambda}$, as we have already seen in earlier chapters. Recall that the Jacobian matrix depends on $\mathbf{q}$, and the Lagrange multipliers $\boldsymbol{\lambda}$ represent both non-interpenetration and friction forces. The $\boldsymbol{\lambda}$ originate from a set of constraints that describes the feasible contact forces, and thus the motion ODEs need to be augmented with these constraint equations. This results in a set of differential algebraic equations (DAEs).

However, we use a more generic form of the constraints to describe the high-level ideas of the theory. The equations of motion are augmented with the constraint function, ψ, that models feasible normal and friction forces, such that $\psi\left(\phi(\mathbf{q}), \boldsymbol{\lambda}\right) = 0$. Combining it with the set of coupled first-order differential equations from before, the system of equations for the constraint-based formulation can be written as:

$$\begin{aligned} \mathbf{M}\dot{\mathbf{u}} &= \mathbf{f}_{\text{ext}} + \mathbf{f}_{\text{d}}(\mathbf{u}) + \mathbf{f}_{\text{e}}(\mathbf{q}) + \mathbf{J}^T\boldsymbol{\lambda} \\ \dot{\mathbf{q}} &= \mathbf{u} \\ \psi\left(\phi(\mathbf{q}), \boldsymbol{\lambda}\right) &= 0 \end{aligned}$$

Once again, applying the implicit Euler integration method gives the discretized form:

$$\mathbf{M}\,\mathbf{u}^{t+h} - \mathbf{M}\,\mathbf{u}^{t} = h\,\mathbf{f}_{\text{ext}} + h\,\mathbf{f}_{\text{d}}^{t+h} + h\,\mathbf{f}_{\text{e}}^{t+h} + h\,\left(\mathbf{J}^{t+h}\right)^T\boldsymbol{\lambda}^{t+h} \tag{263a}$$

$$\mathbf{q}^{t+h} = \mathbf{q}^{t} + h\,\mathbf{u}^{t+h} \tag{263b}$$

$$\psi\left(\phi(\mathbf{q}^{t+h}), \boldsymbol{\lambda}^{t+h}\right) = 0 \tag{263c}$$

Some further reorganization of terms gives the root-finding function:

$$\underbrace{\begin{bmatrix} \mathbf{M}\,\left(\mathbf{u}^{t+h} - \mathbf{u}^{t}\right) - h\,\mathbf{f}_{\text{ext}} - h\,\mathbf{f}_{\text{d}}^{t+h} - h\,\mathbf{f}_{\text{e}}^{t+h} - h\,\left(\mathbf{J}^{t+h}\right)^T\boldsymbol{\lambda}^{t+h} \\ \mathbf{q}^{t+h} - \mathbf{q}^{t} - h\,\mathbf{u}^{t+h} \\ \psi\left(\phi(\mathbf{q}^{t+h}), \boldsymbol{\lambda}^{t+h}\right) \end{bmatrix}}_{\equiv \mathbf{F}_{\text{dual}}} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ 0 \end{bmatrix}.$$

We are now ready to perform substitutions using first-order Taylor approximations. It is often assumed that the contact Jacobian does not change significantly over a time-step, hence it can be seen as time independent. Sticking with this assumption, and re-scaling the Lagrange multiplier by the time-step h, gives:

$$\begin{bmatrix} \mathbf{A}^k & \mathbf{0} & -\mathbf{J}^T \\ -h\mathbf{I} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \nabla_{\mathbf{q}}\psi^k & \nabla_{\boldsymbol{\lambda}}\psi^k \end{bmatrix} \begin{bmatrix} \Delta\mathbf{u} \\ \Delta\mathbf{q} \\ \Delta\boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{b}^k \\ \mathbf{q}^t - \mathbf{q}^k + h\mathbf{u}^k \\ -\psi\left(\phi(\mathbf{q}^k), \boldsymbol{\lambda}^k\right) \end{bmatrix} \tag{264}$$

where

$$\mathbf{A}^k \equiv \left(\mathbf{M} - h\,\mathbf{B}^k - h^2\,\mathbf{K}^k\right),$$
$$\mathbf{b}^k \equiv \mathbf{M}\,\left(\mathbf{u}^t - \mathbf{u}^k\right) + h\,\left(\mathbf{f}_{\text{ext}} + \mathbf{f}_{\text{d}}^k + \mathbf{f}_{\text{e}}^k\right) - \mathbf{J}^T\,\boldsymbol{\lambda}^k\,.$$

Again, the superscript indicates the time step or iterate where a term is evaluated, and additionally we adopt the following short-hand notation:

$$\nabla_{\mathbf{q}}\psi^k \equiv \frac{\partial\psi\left(\phi(\mathbf{q}^k), \boldsymbol{\lambda}^k\right)}{\partial\mathbf{q}} \quad \text{and} \quad \nabla_{\boldsymbol{\lambda}}\psi^k \equiv \frac{\partial\psi\left(\phi(\mathbf{q}^k), \boldsymbol{\lambda}^k\right)}{\partial\boldsymbol{\lambda}}\,.$$

We have now derived the Newton equation in its most generic form. Section 3.3 later provides a more in-depth presentation about how to solve this type of system for the specific case of isotropic Coulomb friction models.

4.2.2 Position-level Minimization Approach. We next consider another class of methods for time-stepping soft body systems. Optimization-based approaches rewrite the time-integration problem as a minimization problem where a first-order optimization technique can be applied to compute kinematic quantities at the end of each time-step. Upon convergence, the optimization-based approaches should give the same solutions as the root-finding approaches, even though intermediate solutions may be different. Hence, we postulate that the solution of the constrained minimization problem

$$\mathbf{q}^{t+h} \equiv \underset{\mathbf{q},\phi(\mathbf{q})\geq 0}{\arg\min}\ \underbrace{\frac{1}{2}\,(\mathbf{q}-\tilde{\mathbf{q}})^T\,\mathbf{M}\,(\mathbf{q}-\tilde{\mathbf{q}}) + h^2\mathcal{E}_{\text{e}}(\mathbf{q}) - \mathbf{q}^T h^2\mathbf{f}_{\text{d}}(\mathbf{u})}_{\equiv f(\mathbf{q},\mathbf{u})} \tag{265}$$

is a solution to the fully implicit Euler time integration. The first term of Equation 265 minimizes the momentum potential, where $\tilde{\mathbf{q}} = \mathbf{q}^t + h\,\mathbf{u}^t + h^2\mathbf{M}^{-1}\mathbf{f}_{\text{ext}}$. The second term minimizes the energy potential of elastic strain, $\mathcal{E}_{\text{e}}$, and the third term maximizes work done by damping forces, $\mathbf{f}_{\text{d}}$.

Stating the optimization problem is only the first step. We must next derive the optimality conditions that a solution of the minimization problem must satisfy. To do so, we write the Lagrangian function

$$\mathcal{L}(\mathbf{q},\mathbf{u}) \equiv f(\mathbf{q},\mathbf{u}) - \boldsymbol{\gamma}^T\phi(\mathbf{q})\,,$$

where $\boldsymbol{\gamma} \geq 0$ are the Lagrange multipliers for unilateral constraints $\phi(\mathbf{q}) \geq 0$. We thus define the first-order optimality conditions by the gradient of the Lagrangian:

$$\begin{aligned}\nabla_{\mathbf{q}}\mathcal{L}(\mathbf{q}^{t+h},\mathbf{u}^{t+h}) = \nabla_{\mathbf{q}} f(\mathbf{q}^{t+h},\mathbf{u}^{t+h}) - \frac{\partial \phi(\mathbf{q}^{t+h})}{\partial \mathbf{q}}^{T} \boldsymbol{\gamma} = 0\,,\\ \boldsymbol{\gamma} \geq 0\,,\\ \phi(\mathbf{q}^{t+h}) \geq 0\,,\\ \boldsymbol{\gamma}^{T}\phi(\mathbf{q}^{t+h}) = 0\,.\end{aligned}$$

The last three equations can be replaced by the NCP function that was previously introduced. That is, $\psi\left(\phi(\mathbf{q}^{t+h}), \boldsymbol{\gamma}\right) = 0$. Expanding $f(\mathbf{q}^{t+h},\mathbf{u}^{t+h})$ from Equation 265, the derivative is rewritten

$$\nabla_{\mathbf{q}}\mathcal{L}(\mathbf{q}^{t+h},\mathbf{u}^{t+h}) = \mathbf{M}\left(\mathbf{q}-\tilde{\mathbf{q}}\right) + h^2\nabla_{\mathbf{q}}\mathcal{E}_{\mathrm{e}}(\mathbf{q}) - h^2\mathbf{f}_{\mathrm{d}}(\mathbf{u}) - \left(\mathbf{J}^{t+h}\right)^{T}\boldsymbol{\gamma}\,,$$

where $\mathbf{J}^{t+h} \equiv \frac{\partial \phi(\mathbf{q}^{t+h})}{\partial \mathbf{q}}$ is the Jacobian of the contact constraints. The elastic forces are the negative gradient of the strain energy potential, $\mathbf{f}_{\mathrm{e}}(\mathbf{q}) \equiv -\nabla_{\mathbf{q}}\mathcal{E}_{\mathrm{e}}(\mathbf{q})$, and so the above expression can be simplified even further:

$$\nabla_{\mathbf{q}}\mathcal{L}(\mathbf{q}^{t+h},\mathbf{u}^{t+h}) = \mathbf{M}\left(\mathbf{q}^{t+h}-\tilde{\mathbf{q}}\right) - h^2\mathbf{f}_{\mathrm{e}}(\mathbf{q}^{t+h}) - h^2\mathbf{f}_{\mathrm{d}}(\mathbf{u}^{t+h}) - \left(\mathbf{J}^{t+h}\right)^{T}\boldsymbol{\gamma}\,.$$

Finally, substituting for $\tilde{\mathbf{q}}$ gives:

$$\begin{aligned}&\mathbf{M}\mathbf{q}^{t+h} - \mathbf{M}\mathbf{q}^{t} - h\mathbf{M}\mathbf{u}^{t} - h^2\mathbf{f}_{\mathrm{ext}} - h^2\mathbf{f}_{\mathrm{e}}(\mathbf{q}^{t+h}) - h^2\mathbf{f}_{\mathrm{d}}(\mathbf{u}^{t+h}) - \left(\mathbf{J}^{t+h}\right)^{T}\boldsymbol{\gamma} = 0\\ \Rightarrow\ &h\mathbf{M}\mathbf{u}^{t+h} - h\mathbf{M}\mathbf{u}^{t} - h^2\mathbf{f}_{\mathrm{ext}} - h^2\mathbf{f}_{\mathrm{e}}(\mathbf{q}^{t+h}) - h^2\mathbf{f}_{\mathrm{d}}(\mathbf{u}^{t+h}) - \left(\mathbf{J}^{t+h}\right)^{T}\boldsymbol{\gamma} = 0\\ \Rightarrow\ &\mathbf{M}\left(\mathbf{u}^{t+h}-\mathbf{u}^{t}\right) - h\mathbf{f}_{\mathrm{ext}} - h\mathbf{f}_{\mathrm{e}}(\mathbf{q}^{t+h}) - h\mathbf{f}_{\mathrm{d}}(\mathbf{u}^{t+h}) - \frac{1}{h}\left(\mathbf{J}^{t+h}\right)^{T}\boldsymbol{\gamma} = 0\end{aligned}$$

Compared with Equation 263a, the above expression is almost identical. The last piece of the puzzle comes with the realization that substituting $\phi \leftarrow h^2\phi$ and $\gamma \leftarrow \frac{1}{h}\boldsymbol{\lambda}^{t+h}$ do not change the underlining problem, but makes the equations match exactly.

The minimization approach for the primal form comes easy by recalling that $\mathbf{f}_{\mathrm{c}}(\mathbf{q},\mathbf{u}) \equiv -\nabla_{\mathbf{q}}\mathcal{E}_{\mathrm{c}}(\mathbf{q},\mathbf{u})$. Hence, the equivalent minimization problem of the primal form is given by:

$$\mathbf{q}^{t+h} \equiv \arg\min_{\mathbf{q}}\ \frac{1}{2}\left(\mathbf{q}-\tilde{\mathbf{q}}\right)^{T}\mathbf{M}\left(\mathbf{q}-\tilde{\mathbf{q}}\right) + h^2\mathcal{E}_{\mathrm{e}}(\mathbf{q}) + h^2\mathcal{E}_{\mathrm{c}}(\mathbf{q},\mathbf{u}) - \mathbf{q}^{T}h^2\mathbf{f}_{\mathrm{d}}(\mathbf{u})$$

Observe that the unilateral constraint is gone, and instead we have an unconstrained minimization problem to solve.

Position-level minimization approaches have become fundamental for implicit integration of soft body simulations. In Section 5.2, one such approach is presenting that combines a Newton method with a line-search technique based on continuous collision detection.

4.2.3 Velocity-level Minimization Approach. A velocity form of the minimization approach presented in the previous section may also be derived, such that

$$\mathbf{u}^{t+h} \equiv \underset{\mathbf{u},\, \mathbf{J}\mathbf{u}\geq 0}{\arg\min}\ \frac{1}{2}\left(\mathbf{u}-\tilde{\mathbf{u}}\right)^T \mathbf{M}\left(\mathbf{u}-\tilde{\mathbf{u}}\right) + h\,\mathcal{E}_{\mathrm{e}}(\mathbf{q})\,,$$

where $\tilde{\mathbf{u}} \equiv \mathbf{u}^t + h\mathbf{M}^{-1}\mathbf{f}_{\mathrm{ext}}$. Here, damping forces are ignored. In this approach, the constraint function $\phi(\mathbf{q})$ is approximated by its first-order Taylor expansion $\phi(\mathbf{q}) \approx \phi(\mathbf{q}^t) + h\mathbf{J}\mathbf{u} \implies \mathbf{J}\,\mathbf{u} \geq 0$. This assumes that initially $\phi(\mathbf{q}^t) = 0$. Further, the elastic force term is handled by considering $\mathbf{q}$ as a function of $\mathbf{u}$. Applying the chain rule gives:

$$\begin{aligned}\nabla_{\mathbf{u}}\mathcal{E}_{\mathrm{e}}(\mathbf{q}) &= \frac{\partial \mathbf{q}}{\partial \mathbf{u}}^T \frac{\partial \mathcal{E}_{\mathrm{e}}(\mathbf{q})}{\partial \mathbf{q}}\,,\\ &= \frac{1}{h}\frac{\partial \mathcal{E}_{\mathrm{e}}(\mathbf{q})}{\partial \mathbf{q}}\,.\end{aligned}$$

Many of the ideas from solving the position-level minimization problem can be transferred to the velocity-level formulation, and so we do not expand further on velocity level approaches in these notes. The velocity-level formulation as stated here is often not used for soft body contact problems. Instead, the position-based formulation is preferred as the non-linear form of constraints is maintained. The velocity level formulation has been used for fluid simulations [Batty et al. 2007].

4.3 A Taxonomy of Preconditioners for Dual Form Approaches

So far, we have presented several fundamental approaches for how to formulate the time integration problem as either a root-finding problem or a minimization problem. We focused completely on the full implicit first order time integration. These approaches are quite powerful and represent the state-of-the-art for soft body dynamics with contact. In contrast, we will now cover a range of approaches and techniques that have been used to create fast and efficient simulators. Before diving into the details, we briefly outline the general idea behind many of these methods we will cover. We will write up a typical form of a first-order Euler

time-discretized system as:

$$\begin{aligned}
\mathbf{M}\mathbf{u}^{t+h} - h\,\mathbf{f}_{\mathrm{d}}(\mathbf{u}^{\tau}) - h\,\mathbf{f}_{\mathrm{e}}(\mathbf{q}^{\tau}) - \mathbf{J}^{T}\,\boldsymbol{\lambda}^{t+h} &= h\,\mathbf{f}_{\mathrm{ext}} + \mathbf{M}\mathbf{u}^{t} \\
\mathbf{q}^{t+h} - h\,\mathbf{u}^{t+h} &= \mathbf{q}^{t} \\
\phi(\mathbf{q}^{t+h}) &\geq 0 \\
\boldsymbol{\lambda}^{t+h} &\geq 0 \\
\phi(\mathbf{q}^{t+h})^{T}\boldsymbol{\lambda}^{t+h} &= 0
\end{aligned}$$

This is the dual form, but with the NCP written using three equations rather than a single equation. Furthermore, τ is used to abstract over implicit versus explicit choices for evaluating elastic and damping forces. Recall that $\mathbf{J}$ is dependent on $\mathbf{q}$. Hence, strictly speaking a full implicit time-stepping would use $\mathbf{J}^{t+h}$ in above equations. Here we assume contact information stays unchanged over the time-step. This way notation is less verbose when we present our ideas. All concepts do extend to the full implicit notation as well.

When working with constraint-based approaches, such as the one above, typically a linearization of the constraint is used:

$$\phi(\mathbf{q}^{t+h}) \approx \phi(\mathbf{q}^{t}) + h\frac{\partial\phi(\mathbf{q}^{t+h})}{\partial\mathbf{q}}\mathbf{u}^{t+h} \geq 0 \tag{266a}$$

$$\Rightarrow \phi(\mathbf{q}^{t}) + h\mathbf{J}\,\mathbf{u}^{t+h} \geq 0 \tag{266b}$$

$$\Rightarrow \mathbf{J}\,\mathbf{u}^{t+h} \geq -\frac{1}{h}\phi(\mathbf{q}^{t}) \tag{266c}$$

This form reveals that constraint errors can be corrected, or future contacts anticipated, by keeping the term $-\frac{1}{h}\phi(\mathbf{q}^{t})$ and results in a Baumgarte-type of constraint stabilization, as we saw in Section 1.9.

Other approaches assume that constraints are only formed when touching contact exists between soft bodies. This implies $\phi(\mathbf{q}^{t}) = 0$, and the right-hand side of Equation 266c can be set to zero. Defining $\mathbf{v}^{t+h} \equiv \mathbf{J}\,\mathbf{u}^{t+h}$ and $\epsilon^{t} \equiv -\frac{1}{h}\phi(\mathbf{q}^{t})$, we can then rewrite the system of equations as:

$$\begin{aligned}
\mathbf{M}\mathbf{u}^{t+h} - h\,\mathbf{f}_{\mathrm{d}}(\mathbf{u}^{\tau}) - h\,\mathbf{f}_{\mathrm{e}}(\mathbf{q}^{\tau}) - \mathbf{J}^{T}\,\boldsymbol{\lambda}^{t+h} &= h\,\mathbf{f}_{\mathrm{ext}} + \mathbf{M}\mathbf{u}^{t} \\
\mathbf{q}^{t+h} - h\,\mathbf{u}^{t+h} &= \mathbf{q}^{t} \\
\mathbf{v}^{t+h} &= \mathbf{J}\,\mathbf{u}^{t+h} \\
0 \leq \mathbf{v}^{t+h} - \epsilon^{t} \;\perp\; \boldsymbol{\lambda}^{t+h} &\geq 0
\end{aligned}$$

This is the form of the soft body system with contact we will work with going forward. The approach for solving this problem now depends on the specific choice of using $\tau = t$ or $\tau = t+h$ for each of the terms in the first equation of our system. These choices combined with solving

for $\mathbf{u}^{t+h}$ from the first equation and substituting into the third equation results in a reduced complementarity problem of the form:

$$0 \leq \mathbf{W}_{\mathrm{P}}\, \boldsymbol{\lambda}^{t+h} + \mathbf{w}_{\mathrm{P}} \;\perp\; \boldsymbol{\lambda}^{t+h} \geq 0$$

The subscript on the Delassus operator $\mathbf{W}_{\mathrm{P}}$ and right-hand side vector $\mathbf{w}_{\mathrm{P}}$ is used here to denote that different decisions in the different methods lead to different matrices and vectors. We will show how these derivations are done a bit later.

Once the complementarity problem is solved, $\boldsymbol{\lambda}^{t+h}$ is known and $\mathbf{u}^{t+h}$ can be computed from the first equation. Finally, from the second equation we obtain $\mathbf{q}^{t+h}$. Hence, $\mathbf{u}^{t+h}$ may be eliminated from the first and third row of our system to give the reduced system:

$$\begin{aligned} \mathbf{M}\mathbf{u}^{t+h} - h\,\mathbf{f}_{\mathrm{d}}(\mathbf{u}^{\tau}) - h\,\mathbf{f}_{\mathrm{e}}(\mathbf{q}^{\tau}) - \mathbf{J}^{T}\,\boldsymbol{\lambda}^{t+h} &= h\,\mathbf{f}_{\mathrm{ext}} + \mathbf{M}\mathbf{u}^{t}\,, \\ \mathbf{v}^{t+h} &= \mathbf{J}\,\mathbf{u}^{t+h} \end{aligned}$$

Now, our attention turns toward finding methods of efficiently computing $\mathbf{W}_{\mathrm{P}}$, or methods that give $\mathbf{W}_{\mathrm{P}}$ a simple form that allows us to solve the complementarity problem easily. A key insight from this mathematical exercise is that we can frame the methods as different choices for computing the preconditioner $\mathbf{P} \approx \left(\mathbf{M} + h\mathbf{B} + h^2\mathbf{K}\right)^{-1}$, and then preconditioned form of the Delassus operator may be used, such that

$$\mathbf{W}_{\mathrm{P}} \equiv \mathbf{J}\,\mathbf{P}\,\mathbf{J}^{T}\,.$$

Hence, the subscript now simply denotes a specific choice of preconditioner. We are now ready to outline the many different choices of preconditioners that can be applied.

4.3.1 Quality of a Preconditioners. By a "good" preconditioner, we mean $\mathbf{P}\,\mathbf{A} \approx \mathbf{I}$. Expanding the system matrix as $\mathbf{A} \equiv \mathbf{M} + h\mathbf{B} + h^2\mathbf{K}$ allows us to write a more exact definition:

$$\mathbf{P}\,\left(\mathbf{M} - h\,\mathbf{B} - h^2\,\mathbf{K}\right) \approx \mathbf{I}$$

Clearly, this is only a good approximation for $h \to 0$. Observe the h^2 term on the stiffness matrix. This too reveal that when lowering the value of h which is necessary to have a good preconditioner then we get a more "weak" coupling of anticipation of how elastic forces work between mesh nodes.

The view that different methods are simply different preconditioner choices allow us to interpret and rank the methods in terms of how close their preconditioner is to the real system. Table 4 presents an overview of the various preconditioners that we cover in this section, as well as some of the highlights and differences of these approaches.

4.3.2 Explicit Time-Discretization and Mass Lumping. Let us derive a very simple preconditioner. The idea is to evaluate elastic and damping forces in a fully explicit way and assume a

Table 4. A coarse comparison of the preconditioner cases covered in this chapter. The (‡) case considers building $\mathbf{W}_\mathrm{A}$ from a Cholesky factorization whereas (†) avoids building the Delassus operator.

P-case	Advantages	Disadvantages	See Section
$\mathbf{W}_\mathrm{M}$	Very fast	Small time-steps, sensitive to instabilities, weak nodal couplings	4.3.2
$\mathbf{W}_\mathrm{A}$	Large time-steps, strong nodal couplings	Matrix inversion is expensive.	4.3.3
$\mathbf{W}_\mathrm{LDLT}$ (‡)	Parallel building	Computing Cholesky factorization	4.3.4
$\mathbf{W}_\mathrm{LDLT}$ (†)	No need for $\mathbf{W}_\mathrm{A}$	Computing Cholesky factorization, Partial evaluation is too expensive	4.3.5
$\mathbf{W}_\mathrm{D}$	Easy to implement, Large time-steps	Convergence rate is linear, no proof of convergence, solves a sequence of complementarity problems	4.3.6

lumped mass matrix. Then our reduced system becomes:

$$\mathbf{M}\mathbf{u}^{t+h} - h\,\mathbf{f}_\mathrm{d}(\mathbf{u}^t) - h\,\mathbf{f}_\mathrm{e}(\mathbf{q}^t) - \mathbf{J}^T\boldsymbol{\lambda}^{t+h} = h\,\mathbf{f}_\mathrm{ext} + \mathbf{M}\mathbf{u}^t$$
$$\mathbf{v}^{t+h} = \mathbf{J}\,\mathbf{u}^{t+h}$$

Solving for $\mathbf{u}^{t+h}$ is now efficient since $\mathbf{M}$ is assumed to be a diagonal matrix:

$$\mathbf{u}^{t+h} = \mathbf{M}^{-1}\mathbf{J}^T\boldsymbol{\lambda}^{t+h} + \mathbf{u}^t + h\,\mathbf{M}^{-1}\left(\mathbf{f}_\mathrm{d}(\mathbf{u}^t) + \mathbf{f}_\mathrm{e}(\mathbf{q}^t) + \mathbf{f}_\mathrm{ext}\right)$$

Multiplying by $\mathbf{J}$ on both sides gives:

$$\mathbf{v}^{t+h} = \underbrace{\mathbf{J}\,\mathbf{M}^{-1}\mathbf{J}^T}_{\equiv\mathbf{W}_\mathrm{M}}\boldsymbol{\lambda}^{t+h} + \mathbf{J}\,\mathbf{u}^t + h\,\mathbf{J}\,\mathbf{M}^{-1}\left(\mathbf{f}_\mathrm{d}(\mathbf{u}^t) + \mathbf{f}_\mathrm{e}(\mathbf{q}^t) + \mathbf{f}_\mathrm{ext}\right)$$

Putting things back into the complimentarity formulation, we arrive at the lumped mass preconditioned system with

$$\mathbf{W}_\mathrm{M} = \mathbf{J}\,\mathbf{M}^{-1}\mathbf{J}^T,$$
$$\mathbf{w}_\mathrm{M} = \mathbf{J}\,\mathbf{u}^t + h\,\mathbf{J}\,\mathbf{M}^{-1}\left(\mathbf{f}_\mathrm{d}(\mathbf{u}^t) + \mathbf{f}_\mathrm{e}(\mathbf{q}^t) + \mathbf{f}_\mathrm{ext}\right) - \epsilon^t.$$

The time integration now boils down to solving the complementarity problem:

$$0 \leq \mathbf{W}_\mathrm{M}\boldsymbol{\lambda}^{t+h} + \mathbf{w}_\mathrm{M} \;\perp\; \boldsymbol{\lambda}^{t+h} \geq 0$$

This can be done very efficiently with a projected Gauss-Seidel type of method due to the diagonal lumped mass matrix, which results in a fast per-iteration update and straightforward implementation. It avoids the matrix inversion of $\left(\mathbf{M} + h\mathbf{B} + h^2\mathbf{K}\right)$ that other implicit methods face when building the Delassus operator. Conversely, the explicit nature of elastic forces can severely limit the time-step size due to instabilities. Propagation and accumulation of round-off errors can also cause instabilities for longer running simulations. Lumping the mass matrix may not always be desired either. The mass lumping will create a "weaker" coupling of force interactions between mesh nodes, and contact forces will inherit this weaker coupling. Numerically, it will take more time-steps to have the effect of contact forces from one local area propagate to other areas of the soft body volume mesh. Hence, the mass lumping preconditioner is limited in this way too. Finally, we note that this form is almost identical to its rigid body counterpart (see Equation 9).

4.3.3 Implicit Euler. Let us investigate another preconditioner case. Let us rewind back to the core system and this time we do not assume mass lumping and further we wish to evaluate the damping and elastic forces implicitly. That is we have,

$$\mathbf{M}\mathbf{u}^{t+h} - h\,\mathbf{f}_\mathrm{d}(\mathbf{u}^{t+h}) - h\,\mathbf{f}_\mathrm{e}(\mathbf{q}^{t+h}) - \mathbf{J}^T\boldsymbol{\lambda}^{t+h} = h\,\mathbf{f}_\mathrm{ext} + \mathbf{M}\mathbf{u}^t\,, \tag{267a}$$

$$\mathbf{v}^{t+h} = \mathbf{J}\,\mathbf{u}^{t+h}\,. \tag{267b}$$

Let us with out loss of generality assume a simple damping model $\mathbf{f}_\mathrm{d} = \mathbf{B}\,\mathbf{u}$ and make a linearization of the elastic forces,

$$\mathbf{f}_\mathrm{e}(\mathbf{q}^{t+1}) \approx \mathbf{f}_\mathrm{e}(\mathbf{q}^t) + h\mathbf{K}\,\mathbf{u}^{t+1}\,.$$

Substitution into our first equation and cleaning up gives us,

$$\underbrace{\left(\mathbf{M} - h\,\mathbf{B} - h^2\,\mathbf{K}\right)}_{\equiv\mathbf{A}}\,\mathbf{u}^{t+h} - \mathbf{J}^T\,\boldsymbol{\lambda}^{t+h} = h\,\left(\mathbf{f}_\mathrm{ext} + \mathbf{f}_\mathrm{e}(\mathbf{q}^t)\right) + \mathbf{M}\mathbf{u}^t\,,$$

One may observe that $\mathbf{A}$ is a symmetric positive definite matrix. Hence, it is invertible and we can solve for $\mathbf{u}^{t+h}$,

$$\mathbf{u}^{t+h} = h\mathbf{A}^{-1}\,\left(\left(\mathbf{f}_\mathrm{ext} + \mathbf{f}_\mathrm{e}(\mathbf{q}^t)\right) + \mathbf{M}\mathbf{u}^t\right) + \mathbf{A}^{-1}\,\mathbf{J}^T\,\boldsymbol{\lambda}^{t+h}\,.$$

Substituting $\mathbf{v}^{t+h} = \mathbf{J}\,\mathbf{u}^{t+h}$ yields:

$$\mathbf{v}^{t+h} = \underbrace{\mathbf{J}\,\mathbf{A}^{-1}\,\mathbf{J}^T}_{\equiv\mathbf{W}_\mathrm{A}}\,\boldsymbol{\lambda}^{t+h} + h\mathbf{J}\,\mathbf{A}^{-1}\,\left(\left(\mathbf{f}_\mathrm{ext} + \mathbf{f}_\mathrm{e}(\mathbf{q}^t)\right) + \mathbf{M}\mathbf{u}^t\right)$$

Cleaning up and putting results into the complementarity formulation gives:

$$\mathbf{W}_\mathrm{A} = \mathbf{J}\,\mathbf{A}^{-1}\,\mathbf{J}^T$$

$$\mathbf{w}_\mathrm{A} = h\mathbf{J}\,\mathbf{A}^{-1}\,\left(\left(\mathbf{f}_\mathrm{ext} + \mathbf{f}_\mathrm{e}(\mathbf{q}^t)\right) + \mathbf{M}\mathbf{u}^t\right) - \epsilon^t$$

The time integration now boils down to solving the complementarity problem,

$$0 \leq \mathbf{W}_{\mathrm{A}} \boldsymbol{\lambda}^{t+h} + \mathbf{w}_{\mathrm{A}} \perp \boldsymbol{\lambda}^{t+h} \geq 0 .$$

We notice that $\mathbf{W}_{\mathrm{A}}$ is a perfect preconditioner. Hence, it will be able to handle larger time-steps and is not limited to the lumped mass matrix choice that we used for $\mathbf{W}_{\mathrm{M}}$. On the other hand, computing $\mathbf{P}$ requires the inversion of the $\mathbf{A}$-matrix and naive approaches may require a complexity of $O(N^3)$ in the worst case. Because the number of degrees-of-freedom in the mesh, N is a number that for most solids will be much larger than the number of contacts K, a naive inversion approach will dominate any solver we use for computing the contact forces. For instance, consider a cube with M nodes along each axis, or $N = M^3$ nodes in total. If the cube is resting on the ground on one of its faces, then we will have $K = N^2$ nodes in contact. Obviously, there will be an order of magnitude more nodal degrees-of-freedom than contact constraints.

The approach we have outlined here is often termed the backward Euler method [Baraff and Witkin 1998]. It is related to the dual form root-finding approach we presented earlier in that it corresponds to setting up and solving the Newton equation from Equation 264 only once. In the following sections, we will present different approaches to deal with the computational bottleneck associated with computing the $\mathbf{W}_{\mathrm{A}}$-matrix. We will explore a parallel algorithm for building $\mathbf{W}_{\mathrm{A}}$ and a partial evaluation approach that can be applied in iterative methods, such as projected Gauss-Seidel. Both of these methods use a Cholesky factorization as a building block. Finally, we present a splitting approach known as the Iterative Constraint Anticipation (ICA) method [Otaduy et al. 2009b]. ICA has been the dominating state-of-the-art method for quite many years in the field, but recent work on Newton-type methods and barrier functions has surpassed it. Building $\mathbf{W}_{\mathrm{A}}$ is often the preferred choice over partial evaluation, since one partial evaluation is almost as expensive as building $\mathbf{W}_{\mathrm{A}}$.

4.3.4 Parallel Assembly using the Cholesky Factorization. The $\mathbf{W}_{\mathrm{A}}$ version of the Delassus operator requires the inversion of the matrix $\mathbf{A} \equiv \mathbf{M} + h\mathbf{B} + h^2\mathbf{K}$. We know that $\mathbf{A}$ is a symmetric positive definite matrix, and this characteristic can be exploited to create an efficient parallel method for computing $\mathbf{W}_{\mathrm{A}}$. For instance, this approach has been applied in GPU simulators for surgical training [Courtecuisse et al. 2011]. The idea is centered around first computing an incomplete Cholesky factorization of $\mathbf{A}$,

$$\mathbf{A} = \mathbf{L}\,\mathbf{D}\,\mathbf{L}^T .$$

Here, $\mathbf{L}$ is a lower triangular matrix and $\mathbf{D}$ is a positive diagonal matrix. This we can consider as a special preconditioner choice of $\mathbf{P} = (\mathbf{L}\,\mathbf{D}\,\mathbf{L}^T)^{-1}$. Hence, we may write $\mathbf{W}_{\mathrm{LDLT}}$ when the Cholesky factorization is used to make this explicit, although we obviously have $\mathbf{W}_{\mathrm{LDLT}} = \mathbf{W}_{\mathrm{A}}$ and $\mathbf{w}_{\mathrm{LDLT}} = \mathbf{w}_{\mathrm{A}}$.

A key aspect of this approach is that all the columns of $\mathbf{W}_{\mathrm{A}}$ are computed in parallel. To mask out columns of $\mathbf{W}_{\mathrm{A}}$, we introduce the notation of the k^{th} unit-axis vector, $\mathbf{e}_k$. The i^{th}

component of $\mathbf{e}_k$ is defined as follows:

$$\mathbf{e}_{k,i} = \begin{cases} 0 & \text{if } i \neq k \\ 1 & \text{if } i = k \end{cases}$$

Using the notation of $\mathbf{e}_k$, we mask out the k^{th} column of $\mathbf{W}_{\text{A}}$ as:

$$\mathbf{w}_k = \mathbf{W}_{\text{A}}\, \mathbf{e}_k$$

Substituting $\mathbf{W}_{\text{A}} = \mathbf{J}(\mathbf{L}\,\mathbf{D}\,\mathbf{L}^T)^{-1}\mathbf{J}^T$ gives:

$$\mathbf{w}_k = \mathbf{J}\, \left(\mathbf{L}\,\mathbf{D}\,\mathbf{L}^T\right)^{-1}\, \mathbf{J}^T\, \mathbf{e}_k$$

Observe that $\mathbf{J}^T\, \mathbf{e}_k$ is simply the transpose of the k^{th} row of the Jacobian matrix. Let us define $\mathbf{j}_k \equiv \mathbf{J}^T\, \mathbf{e}_k = (\mathbf{J}_{k\star})^T$. We can now write:

$$\mathbf{w}_k = \mathbf{J}\, \underbrace{\left(\mathbf{L}\,\mathbf{D}\,\mathbf{L}^T\right)^{-1}\, \mathbf{j}_k}_{\equiv \mathbf{h}_k} = \mathbf{J}\,\mathbf{h}_k$$

Hence, we just need to compute $\mathbf{h}_k$ and $\mathbf{w}_k$ is easy to obtain. However, to compute $\mathbf{h}_k$ we must solve the system:

$$\left(\mathbf{L}\,\mathbf{D}\,\mathbf{L}^T\right)\, \mathbf{h}_k = \mathbf{j}_k$$

This can be done efficiently given that the Cholesky factorization is available. First, perform a forward substitution to solve for $\mathbf{g}_k$:

$$\mathbf{L}\,\mathbf{g}_k = \mathbf{j}_k$$

Then, solve for $\mathbf{s}_k$ using the diagonal matrix $\mathbf{D}$:

$$\mathbf{D}\,\mathbf{s}_k = \mathbf{g}_k$$

Finally, a backward-substitution solves for $\mathbf{h}_k$:

$$\mathbf{L}^T\,\mathbf{h}_k = \mathbf{s}_k$$

The insight from this is we can compute all columns of $\mathbf{W}_{\text{A}}$ in parallel. For each column computation we can exploit the same Cholesky factorization. A pseudo-code version of the full method is shown in Algorithm 15.

The benefit of this approach is that existing matrix libraries, both CPU or GPU version, may be used to build the full Delassus matrix, $\mathbf{W}_{\text{A}}$, in parallel. This does not change the overall time-complexity, but it does dramatically reduce the performance. This is particularly relevant when extremely fast and high-fidelity simulations are needed. GPU acceleration is quite easy to exploit too, but the model size of the soft bodies will be limited by the total amount of GPU memory available [Courtecuisse et al. 2011]. If the stiffness matrix, $\mathbf{K}$, does not depend on the

Data: **A**: The coefficient matrix, **J**: The Jacobian matrix.
Result: $\mathbf{W}_{\mathrm{A}}$: The assembled Delassus operator.

$\mathbf{W}_{\mathrm{A}} \leftarrow$ Allocate empty matrix;
$\mathbf{D}, \mathbf{L} \leftarrow$ Cholesky-Factorization($\mathbf{A}$);
foreach *k in parallel* **do**
  $\mathbf{j} \leftarrow$ transposed of k^{th} row of $\mathbf{J}$;
  Solve $\mathbf{L}\,\mathbf{g} = \mathbf{j}$ for $\mathbf{g}$;
  Solve $\mathbf{D}\,\mathbf{s} = \mathbf{g}$ for $\mathbf{s}$;
  Solve $\mathbf{L}^T\,\mathbf{h} = \mathbf{s}$ for $\mathbf{h}$;
  $\mathbf{w} \leftarrow \mathbf{J}\,\mathbf{h}$;
  Insert $\mathbf{w}$ into k^{th} column of $\mathbf{W}_{\mathrm{A}}$;
end

Algorithm 15: Parallel-Build-Delassus Building Delassus operator in parallel. The Cholesky factorization is computed only once, and thereafter it is reused in parallel to evaluate the columns of $\mathbf{W}_{\mathrm{A}}$ matrix. The three solves step are a "textbook" Cholesky solver. Factorization and solver sub-routines are standard features in most matrix librarie, and thus this algorithm is straightforward to implement.

current value of $\mathbf{q}$, such as with linear elastic material models, then the Choleksy factorization can be re-used throughout the whole simulation. It will then be quite similar in spirit to the projective dynamics method [Bouaziz et al. 2014; Narain et al. 2016; Wang 2015].

4.3.5 Partial Evaluation using Choleskey Factorization. In past section we presented a method that allows one to build the complete $\mathbf{W}_{\mathrm{A}}$ matrix. In doing this we exploited a Choleksy factorization to make a very efficient parallel method. In this section we explore a different idea. The idea originates from intrinsic knowledge about how factorization of the Delassus operator can be exploited inside a projected Gauss-Seidel type of method. We provide a very detailed treatment of these factorization ideas in Section 3.4.2. Here we will quickly summarize a simplistic Gauss-Seidel type of solver method to introduce the computational building blocks of such a type of method. Here is the rough outline of a bare-bone Gauss-Seidel method,

while *not converged* **do**
  foreach k **do**
    $\mathbf{v}_k \leftarrow (\mathbf{W}_{\mathrm{A}}\boldsymbol{\lambda} + \mathbf{w}_{\mathrm{A}})_k$;
    $\boldsymbol{\lambda}_k \leftarrow \mathrm{proj}\,(\mathbf{v}_k, \boldsymbol{\lambda}_k)$;
  end
end

The main components in this type of method are: An outer loop monitoring convergence and an inner loop that sweeps over all variables, aka contacts. For each variable a type of projection, proj $(\cdot)$, is performed. The projection updates the Lagrange multiplier values for this individual variable. For now we do not have to understand why such a method converges or why it computes a solution. The reader may find this type of information in Section 3.4.1 where we present an example of such a method where the projection operation is done using a proximal operator on a smooth elliptical cone, or in Section 3.2 where we use a simpler box projection, or later in Section 6 where we generalize the concept of projections onto cones that change shape depending on system state. For now we will go on a leap of faith that such a method will compute a solution and instead try and analyze the computations going into the method.

The nasty part of the algorithm outline is found in Line 3. Here the Delassus operator is used to compute the relative sliding velocity vector $\mathbf{v}_k$ for the k^{th} variable (aka contact). This is nasty because it requires us to actual build the $\mathbf{W}_{\text{A}}$-matrix. Once we have $\mathbf{W}_{\text{A}}$ it is of cause quite simple to efficiently compute Line 3. Now the idea we want to explore is whether we can circumvent to build $\mathbf{W}_{\text{A}}$ at all. As seen from the code we actual do not need to know $\mathbf{W}_{\text{A}}$ we only need to know its effect when applied to a vector. This is our ticket to a different way of computing the same thing. To do this we will need to introduce an auxiliary variable, $\mathbf{u}$ and we will factorize the Delassus operator into two terms, $\mathbf{W}_{\text{A}} = (\mathbf{J})\ \left(\mathbf{A}^{-1}\mathbf{J}^T\right)$. This way the algorithm outline can now be re-written as follows.

```
1 u ← w_A;
2 while not converged do
3 |   foreach k do
4 |   |   v_k ← (J u)_k;
5 |   |   λ_k ← proj (v_k, λ_k);
6 |   |   Δλ ← the change in λ from the
  |   |     projection;
7 |   |   u ← u + A^{-1} J^T Δλ;
8 |   end
9 end
```

The computation of the values of $\mathbf{v}_k$ and $\boldsymbol{\lambda}_k$ are exactly the same as before. Our little rewrite only changed that we never need to build $\mathbf{W}_{\text{A}}$. Instead we have to be very efficient at computing $\mathbf{v}_k \leftarrow (\mathbf{J}\,\mathbf{u})_k$ and $\mathbf{u} \leftarrow \mathbf{u} + \mathbf{A}^{-1}\mathbf{J}^T\,\Delta\boldsymbol{\lambda}$ as these are done for each time we sweep over a single contact point.

- The term $\mathbf{v}_k \leftarrow (\mathbf{J}\,\mathbf{u})_k$ is quite easy, it is simply the dot product of the k^{th} row of the Jacobian matrix with our auxiliary vector $\mathbf{u}$. If a sparse matrix library is used and we

assume a linear tetrahedral mesh then this operation can be done in $O(8)$ time as the k^{th} row of $\mathbf{J}$ will have at most 8 non-zero blocks.

- The other term involves the computation of $\mathbf{A}^{-1}\mathbf{J}^T\,\Delta\boldsymbol{\lambda}$, let us denote the value of this computation by $\mathbf{x}$, so we have $\mathbf{x} = \mathbf{A}^{-1}\mathbf{J}^T\,\Delta\boldsymbol{\lambda}$. This term is slightly more complicated to compute. First we observe that vector $\Delta\boldsymbol{\lambda}$ only will have non-zero values for the variables corresponding to the k^{th} variable. All other values in this vector will be zero. Hence, the multiplication of $\mathbf{J}^T$ and $\Delta\boldsymbol{\lambda}$ then will mask-out the k^{th} row of the Jacobian, and scale this by the change in the Lagrange multiplier. Let us call this resulting vector $\mathbf{y}$, then we have,

$$\begin{aligned}\mathbf{x} &= \mathbf{A}^{-1}\mathbf{J}^T\,\Delta\boldsymbol{\lambda}\,,\\ &= \mathbf{A}^{-1}\underbrace{\left(\mathbf{J}^T\,\Delta\boldsymbol{\lambda}\right)}_{=\mathbf{y}}\,.\end{aligned}$$

  This means we should solve for $\mathbf{x}$ such that

$$\mathbf{A}\,\mathbf{x} = \mathbf{y}\,.$$

  Once we know $\mathbf{x}$ we can compute the update of the auxiliary variable as $\mathbf{u} \leftarrow \mathbf{u}+\mathbf{x}$. When solving for $\mathbf{x}$ we can exploit a Cholesky factorization of $\mathbf{A}$.

It is worthwhile to consider the case of using $\mathbf{W}_{\text{M}}$ instead of $\mathbf{W}_{\text{A}}$. In this case solving for $\mathbf{x}$ becomes simply $\mathbf{x} = \mathbf{M}^{-1}\,\mathbf{y}$. Given that $\mathbf{M}$ is a lumped matrix it means the whole update of $\mathbf{u}$ would only take $O(8)$ time. In the more general case of $\mathbf{W}_{\text{A}}$ we have to pay the price of a Cholesky solver for each variable update. That is $O(N^2)$ time-complexity for the update. Hence, the per-iteration cost of one sweep will be $O(K\,N^2)$. This is quite expensive as this is roughly the same price it will cost to build $\mathbf{W}_{\text{A}}$. Hence, partial-updates are often not done for $\mathbf{W}_{\text{A}}$ but is very attractive for $\mathbf{W}_{\text{M}}$.

4.3.6 Splitting Method. As we have seen in previous sections, much of the challenge when dealing with soft body contact comes from the computation of $\mathbf{A}^{-1}$. This is a well-known headache when solving linear systems too. One remedy is to apply iterative methods instead. We will now outline one such idea, which introduces the idea of applying a matrix splitting technique. The end result will be a sequence of complementarity problems. Each problem will be simple to solve, and the converging solution of the simpler problems will be a solution to the original hard complimentary problem which involves the inverted $\mathbf{A}$-matrix. Hence, the origin for this method is the model given by $\mathbf{W}_{\text{A}}$ and $\mathbf{w}_{\text{A}}$. These quantities were derived from Equation 267 which we show here again for convenience:

$$\begin{aligned}\mathbf{A}\mathbf{u}^{t+h} &= h\,\left(\mathbf{f}_{\text{e}}(\mathbf{q}^t) + \mathbf{f}_{\text{ext}}\right) + \mathbf{M}\mathbf{u}^t + \mathbf{J}^T\,\boldsymbol{\lambda}^{t+h}\\ \mathbf{v}^{t+h} &= \mathbf{J}\,\mathbf{u}^{t+h}\end{aligned}$$

Defining $\mathbf{b} = h\ \left(\mathbf{f}_{\mathrm{e}}(\mathbf{q}^t) + \mathbf{f}_{\mathrm{ext}}\right) + \mathbf{M}\mathbf{u}^t$, then the first equation can be written compactly as

$$\mathbf{A}\,\mathbf{u}^{t+h} = \mathbf{b} + \mathbf{J}^T\,\boldsymbol{\lambda}^{t+h}\,.$$

Let us now introduce a splitting of the $\mathbf{A}$-matrix, such that $\mathbf{A} = \mathbf{L} + \mathbf{D} + \mathbf{U}$ where $\mathbf{L}$ is the strict lower triangular part, $\mathbf{D}$ the diagonal and $\mathbf{U}$ the strict triangular upper part of $\mathbf{A}$. This allow us to rewrite our equation:

$$\mathbf{D}\,\mathbf{u}^{t+h} = \mathbf{b} - (\mathbf{L} + \mathbf{U})\ \mathbf{u}^{t+h} + \mathbf{J}^T\,\boldsymbol{\lambda}^{t+h}$$

This provides us with a Jacobi method that iteratively computes an approximation for $\mathbf{u}^{t+h}$. Let us denote the solution from the k^{th} iteration as $\mathbf{u}^k$, which gives a Jacobi update for the new iterate:

$$\mathbf{u}^{k+1} = \mathbf{D}^{-1}\left(\mathbf{b} - (\mathbf{L} + \mathbf{U})\ \mathbf{u}^k\right) + \mathbf{D}^{-1}\,\mathbf{J}^T\,\boldsymbol{\lambda}^{k+1}$$

Substitution into $\mathbf{v}^{k+1} = \mathbf{J}\,\mathbf{u}^{k+1}$ gives us the k^{th} iteration approximation for the relative sliding contact velocity,

$$\mathbf{v}^{k+1} = \underbrace{\mathbf{J}\,\mathbf{D}^{-1}\,\mathbf{J}^T}_{\equiv\mathbf{W}_{\mathrm{D}}}\ \boldsymbol{\lambda}^{k+1} + \mathbf{J}\,\mathbf{D}^{-1}\left(\mathbf{b} - (\mathbf{L} + \mathbf{U})\ \mathbf{u}^k\right)\,.$$

Now we have all the pieces for solving for the $\boldsymbol{\lambda}^{k+1}$ iterate. That is in the k^{th} iteration we solve the complementarity problem,

$$0 \le \mathbf{W}_{\mathrm{D}}^k\,\boldsymbol{\lambda}^{k+1} + \mathbf{w}_{\mathrm{D}}^k\ \perp\ \boldsymbol{\lambda}^{k+1} \ge 0\,,$$

where

$$\mathbf{W}_{\mathrm{D}} = \mathbf{J}\,\mathbf{D}^{-1}\,\mathbf{J}^T\,,$$
$$\mathbf{w}_{\mathrm{D}}^k = \mathbf{J}\,\mathbf{D}^{-1}\left(\mathbf{b} - (\mathbf{L} + \mathbf{U})\ \mathbf{u}^k\right) - \epsilon^t\,.$$

Once $\boldsymbol{\lambda}^{k+1}$ is computed then $\mathbf{u}^{k+1}$ can be computed before one is ready to setup $\mathbf{w}_{\mathrm{D}}^{k+1}$ which will be needed for the next iteration. The overall pseudo-code for this splitting method can be seen in Algorithm 16.

The method we have outlined is quite similar to the Iterative Constraint Anticipation (ICA) method by Otaduy et al. [2009b]. In their work they used a projected Gauss-Seidel solver for the complementarity problem in Line 8 of Algorithm 16. Only an approximate solution is needed. Hence, a few Gauss-Seidel iterations per Jacobi iteration is reported to work well. In context of the partial evaluation analysis we presented in Section 4.3.5 it is instructive to note that $\mathbf{W}_{\mathrm{D}}$ has the exact same form as $\mathbf{W}_{\mathrm{M}}$ and can hence exploit the same computational tricks. Comparing to the $\mathbf{W}_{\mathrm{M}}$-approach we observe that ICA essentially in each outer Jacobi iteration add some of the "coupling" expressed by the $\mathbf{L}$ and $\mathbf{U}$ parts of $\mathbf{A}$ back into the solution by this lagged iterative approach. This was one of the drawbacks of the $\mathbf{W}_{\mathrm{M}}$-approach we treated in

Data: $\mathbf{A}$: The system coefficient matrix, $\mathbf{J}$: The Jacobian matrix, $\mathbf{b} = h\left(\mathbf{f}_e(\mathbf{q}^t) + \mathbf{f}_{ext}\right) + \mathbf{M}\mathbf{u}^t$: The "right-hand-side vector", $\mathbf{u}^t$: The initial body-space velocities.

Result: $\mathbf{u}^{k+1}$: The approximation to the body-space velocities, $\boldsymbol{\lambda}^{k+1}$: the approximation to the Lagrange multiplier solution.

```
Something ;
W_D ← J D^{-1} J^T;
k ← 0;
u^k ← u^t;
while not converged do
    ũ ← D^{-1} (b − (L + U) u^k);
    w_D^k ← J ũ − ε^t;
    Solve: 0 ≤ W_D λ^{k+1} + w_D^k ⊥ λ^{k+1} ≥ 0;
    u^{k+1} ← ũ + D^{-1} J^T λ^{k+1};
    k ← k + 1;
end
```

Algorithm 16: JACOBI-SPLITTING A Jacobi splitting method is used to iteratively update the body-space velocity vector $\mathbf{u}^{k+1}$ until it converges to $\mathbf{u}^{t+h}$. In each iteration a simple constraint problem must be solved. The benefit of the splitting is that the $\mathbf{W}_D$-matrix has a particular nice form similar to $\mathbf{W}_M$. Hence, a projected Gauss-Seidel solver using partial evaluation will be quite efficient from quickly finding an approximation for $\boldsymbol{\lambda}^{k+1}$.

Section 4.3.2. Hence, the ICA method is expected to handle larger time-steps better than the mass lumping approach, and it is not confined to lumping.

5 PENALTY AND BARRIER METHODS

In this chapter, we consider a fundamentally different approach to the constraint-based methods that are covered extensively in earlier chapters. The topic is penalty-based and barrier methods for contact modeling, and the goal here is to provide a broader and more complete coverage of contact and friction simulation. Besides, penalty-based and constraint-based approaches are frequently combined for many implementations, giving a sort of "hybrid" simulator. This is evident from the strong connection between constraint stabilization and penalty-based methods.

5.1 Overview of Classical Penalty Methods

Solid objects in the real world do not penetrate each other. The constraint-based formulations of contact that we have considered so far try to eliminate any penetration between bodies by applying an impulse at the point of contact. In penalty-based methods, the goal is similar. However, rather than computing a constraint impulse, a spring-damper system is used for penalizing penetrations. A spring-damper system behaves like a harmonic oscillator in classical mechanics, and thus the trick is often to tune stiffness and damping parameters to get a critically damped system. Penalty methods have a lot in common with mass-spring systems because all effects are modeled by spring and damping forces. Therefore, these approaches are generally applicable to both deformable and rigid objects. Spring-damper models are even found in biomechanical muscle models, such as Hill's muscle model and Zajac's force model used by Chen and Zeltzer [1992]. Baumgarte stabilization may also be interpreted as a spring-damper system, as we showed in Section 1.9. Hence, it is quite a general purpose modeling tool.

The central idea behind penalty methods is to find a force $\mathbf{f}$ and torque τ at a given instant that can be included in the dynamical equations of motion in order to perform the numerical integration. These forces and torques include contributions from external forces, such as gravity, as well as contact forces stemming from interaction with other rigid bodies. Specifically, with regards to contact, forces are generated by springs with zero rest-length that are inserted at all penetrations. Larger penetrations produced larger spring forces, and thus springs penalize penetrations. Hence the name penalty method. The contact force from each point of contact is thus computed as a spring force.

5.1.1 Rigid Bodies. Consider a rigid body A that penetrates another rigid body B. The spring force $\mathbf{f}_k^{\text{spring}}$ acting at the k^{th} contact point gives rise to the corresponding torque term, τ_k^{spring}, acting on the rigid body. Let the vector arm from center of mass of A to the contact point be given by $\mathbf{r}_{k,A}$, and the resulting torque is computed as

$$\tau_k^{\text{spring}} \equiv \mathbf{r}_{k,A} \times \mathbf{f}_k^{\text{spring}} . \tag{268}$$

To use the penalty method, one would simply sum all the penalty forces and add them to the generalized force vector in Equation 2:

$$\mathbf{f} = \begin{bmatrix} \mathbf{f}^{\text{ext}} + \sum_k \mathbf{f}_k^{\text{spring}} \\ \tau^{\text{ext}} + \sum_k \tau_k^{\text{spring}} \end{bmatrix} \tag{269}$$

Here, the external forces and torques are given by $\mathbf{f}^{\text{ext}}$ and τ^{ext}, and without loss of generalization we consider the effect of the gyroscopic forces to be included in the external torque component.

Recalling there are two bodies in contact, there is a similar force term acting on the opposing body B at the k^{th} contact. Due to Newton's second law, the spring force on that body is equal in magnitude, but opposite in direction. In a practical implementation, one would iterate over contact points and compute the penalty force only once per contact and then distribute the spring force to the two bodies that are in contact. In its most pure form the simulation loop of the penalty method can now be summarized as

- Detect contact points (run collision detection).
- Compute and accumulate spring forces.
- Integrate equations of motion forward in time.

This is the pure form of the penalty method. In practice, it is combined with several other techniques, some of which we will elaborate on below.

First we will turn our attention toward the actual computation of the penalty forces in a general simulator. Let the k^{th} contact point between the two bodies A and B with center of mass at $\mathbf{x}_A$ and $\mathbf{x}_B$ respectively be at position $\mathbf{p}_k$ given in 3D world coordinates. The k^{th} contact point has the contact normal, $\mathbf{n}_k$, also in 3D world space, pointing from body A toward body B, and a measure of penetration depth, ϕ_k. Then the penalty force acting on body B is given by

$$\mathbf{f}_B^{\text{spring}} = (-k\phi_k - b\mathbf{v}_k \cdot \hat{n}_k)\, \hat{n}_k \,, \tag{270}$$

where $\mathbf{v}_k$ denotes the relative contact velocity at the k^{th} contact point. Observe, that the penalty force here only works in the normal direction of the contact. Hence, we are currently ignoring friction. The coefficients k and b are known as the stiffness and damping coefficients, sometimes the stiffness coefficient is referred to as the spring constant as well. The term $\mathbf{v}_k \cdot \hat{n}_k$ is essentially only measuring the relative velocity in the normal direction, let us define this as $v_{\hat{n},k} = \mathbf{v}_k \cdot \hat{n}_k$. For $v_{\hat{n},k} < 0$ objects are approaching each other, $v_{\hat{n},k} = 0$ there is no relative movement, and if $v_{\hat{n},k} > 0$ objects are moving away from each other. Notice how carefully the viscosity is modeled only to work in the direction of the contact normal to ensure that

damping should only work in the constraint direction. We may compute $v_{\hat{n},k}$ like this

$$v_{\hat{n},k} = \hat{n}_k \cdot \left(\left(\mathbf{v}_A + \omega_A \times \mathbf{r}_{k,A} \right) - \left(\mathbf{v}_B + \omega_j \times \mathbf{r}_{k,B} \right) \right) , \tag{271}$$

$$= \underbrace{\hat{n}_k^T \begin{bmatrix} \mathbf{I}_{3\times3} & -\mathbf{r}_{k,A}^{\times} & -\mathbf{I}_{3\times3} & \mathbf{r}_{k,B}^{\times} \end{bmatrix}}_{\mathbf{J}_k} \underbrace{\begin{bmatrix} \mathbf{v}_A \\ \omega_A \\ \mathbf{v}_B \\ \omega_B \end{bmatrix}}_{\mathbf{u}} = \mathbf{J}_k \mathbf{u} , \tag{272}$$

where $\mathbf{r}_{k,A} = \mathbf{p}_k - \mathbf{x}_A$ and $\mathbf{r}_{k,B} = \mathbf{p}_k - \mathbf{x}_B$. We have seen this before in Equation 17, and if we had a soft body instead of a rigid body then the normal relative velocity can be computed as we did in Equation 20. If we write up the magnitude of the spring force as

$$\lambda_{\hat{n},k} \equiv -k\phi_k - b\dot{\phi}_k . \tag{273}$$

Here, we use the substitution that $\dot{\phi}_k = v_{\hat{n},k}$ and with this notation in place we can write the spring forces and torques on body A and B as follows:

$$\begin{bmatrix} \mathbf{f}_A^{\text{spring}} \\ \tau_A^{\text{spring}} \\ \mathbf{f}_B^{\text{spring}} \\ \tau_B^{\text{spring}} \end{bmatrix} = \begin{bmatrix} -\hat{n}_k \lambda_{\hat{n},k} \\ -\mathbf{r}_{k,A} \times \hat{n}_k \lambda_{\hat{n},k} \\ \hat{n}_k \lambda_{\hat{n},k} \\ \mathbf{r}_{k,B} \times \hat{n}_k \lambda_{\hat{n},k} \end{bmatrix} = -\mathbf{J}_k^T \lambda_{\hat{n},k} . \tag{274}$$

It is interesting to generalize Equation 271 and Equation 274 to a system with multiple bodies and multiple contacts. Hence, let us do this. We obtain,

$$\dot{\boldsymbol{\phi}} = \mathbf{J}\mathbf{u} , \tag{275a}$$

$$\boldsymbol{\lambda} = -\mathbf{K}\boldsymbol{\phi} - \mathbf{B}\dot{\boldsymbol{\phi}} , \tag{275b}$$

$$\mathbf{M}\dot{\mathbf{u}} = \mathbf{f} + \mathbf{J}^T \boldsymbol{\lambda} , \tag{275c}$$

here $\mathbf{K}$ is a diagonal matrix with stiffness coefficients, $\mathbf{B}$ a diagonal matrix with damping coefficients, and $\boldsymbol{\lambda}$ is a vector of spring force magnitudes. The vector $\mathbf{u}$ is the generalized velocities and $\mathbf{M}$ is the mass matrix, and $\mathbf{J}$ is the system Jacobian matrix. If we have K contacts and N rigid bodies then $\boldsymbol{\phi}, \dot{\boldsymbol{\phi}}, \boldsymbol{\lambda} \in \mathbb{R}^K$, and $\mathbf{f}, \mathbf{u} \in \mathbb{R}^{6N}$, $\mathbf{J} \in \mathbb{R}^{K\times 6N}$, $\mathbf{M} \in \mathbb{R}^{6N\times 6N}$, and $\mathbf{K}, \mathbf{B} \in \mathbb{R}^{K\times K}$.

5.1.2 Soft Bodies. In case of soft bodies, dimension $6N$ is replaced with $3V$, where V is the total number of vertices in the system. However, note that for soft bodies, there is no torque component generated by the spring forces (meaning the last row of Equation 269 is dropped). Also note that for soft bodies, the contact point may not coincide with a node of the computational mesh. In this case, the penalty force is often distributed to the enclosing nodes

of the contact point using some weighting scheme. The most common choice for computer graphics is to use linear weights based on the barycentric coordinates of the nodes. With these considerations, it is trivial to perform a time-discretization to Equation 275 in order to solve this system of ordinary differential equations. Xu et al. [2014] showed how to take the ideas we presented here into an implicit time-stepping scheme.

5.1.3 Implementation Notes. It is quite interesting to compare the penalty method algebraic form given in Equation 275 with that of a constraint based method. We observe that the $\mathbf{J}$-matrices appear in similar places, but $\boldsymbol{\lambda}$ is now given by a closed-form solution. The spring force, or rather impulses, are simply computed by evaluating Equation 275 at the instant corresponding to the chosen time-stepping scheme. Hence, the per-time-step complexity of the penalty method is very fast compared to solving an LCP problem per time-step. Furthermore, in most implementations of the penalty method, the above matrices are not assembled, which further reduces the computational and storage complexity. Yet the algebraic form here provides a straightforward connection to the constraint based approaches that we have already covered in full detail.

There are, however, several difficulties associated with the classical penalty method. These are listed below without going into details about possible solutions:

- It is not trivial to compute satisfactory contact normals or penetration depths as they lack global knowledge about the entire state. The problem of determining meaningful contact normals and penetration depths is mostly caused by the use of local computations. If a global computation [Kim et al. 2002] is used instead, these problems can be resolved.
- Some contact points like the face-face case appearing in a box stack is difficult to handle by applying a penalty force at the contact point of deepest penetration [Hasegawa et al. 2003]. To alleviate this problem, researchers have tried to sample the entire contact region with contact points. There also exist methods, that integrate over the intersection area and/or volume.
- During motion, there can be a discontinuous changes of the contact normals.
- It is not trivial to pick the stiffness and damping parameters. Stiffness parameter need to be large enough to remove any penetration. Unfortunately, this value limits the time-step size severely for explicit time-stepping methods.
- The springy nature of the contact forces can create secondary oscillation effects that leads to a more "spongy" contact behavior than the hard contact response that is expected from rigid bodies.

5.1.4 Penalty Based Friction. Until now, we have only been concerned with generating non-interpenetration forces using penalty methods. How should one go about modeling friction in a penalty-based simulator? It seems reasonable to use a spring-damper system for the friction force as well. This is often done by tracking the contact points. At first point of contact, an anchor point, $\mathbf{a}$, is saved. This is an unmovable point in the world coordinate system. The

friction force is determined by the spring force stemming from the zero-length spring with stiffness coefficient k between the anchor point and the current position $\mathbf{p}$ of the contact point that caused the creation of the anchor point:

$$\mathbf{f}_{\text{friction}}^{\text{spring}} = k\,(\mathbf{a} - \mathbf{p})\ . \tag{276}$$

Typically, the value of the friction stiffness coefficient is different from the value of the non-interpenetration stiffness coefficient. During subsequent time steps, the simulation is monitored to ensure the following condition holds:

$$\left\|\mathbf{f}_{\text{friction}}^{\text{spring}}\right\| \leq \mu\left\|\mathbf{f}^{\text{spring}}\right\|, \tag{277}$$

here, μ is the Coulomb coefficient of friction and $\mathbf{f}^{\text{spring}}$ is the normal penalty force. If the condition is fulfilled, we have the case of static friction and nothing needs to be done. If, on the other hand, the condition is broken, then we have the case of dynamic friction (sliding) and the anchor point is moved towards the current contact position $\mathbf{p}$ such that the condition in Equation 277 is true.

The frictional spring force model can be physically justified as a crude model of the external shear stresses acting between two bodies. As such, the friction model adds an element of compliance to the rigid bodies. Another approach to model friction is as a damping force [McKenna and Zeltzer 1990], like this,

$$\mathbf{f}_{\text{friction}}^{\text{spring}} = -\mu\left\|\mathbf{f}^{\text{spring}}\right\|\frac{\mathbf{v}_{\hat{t}}}{\|\mathbf{v}_{\hat{t}}\|}, \tag{278}$$

where $\mathbf{v}_{\hat{t}}$ is the tangential relative sliding velocity. This approach only models dynamic friction, thus static friction is ignored.

5.1.5 Other Penalty Methods. We will now briefly survey a few of the usual tricks and ideas that have been used in the context of penalty methods over the years in the field of computer graphics. Our coverage is not complete but most central ideas are represented. Moore and Wilhelms [1988] were among the first to model contact forces by springs in computer animation. They used a simple linear spring, but added a twist of letting the spring constant depend on whether the motion is receding or approaching. The relationship is as follows

$$k_{\text{recede}} = \varepsilon k_{\text{approach}}\,, \tag{279}$$

where ε describes the elasticity of the collision, $\varepsilon = 0$ corresponds to totally inelastic collisions, and $\varepsilon = 1$ to perfectly elastic collisions. Furthermore, Moore and Wilhelms extended the penalty method with an algebraic collision resolving method, similar to Newton's collision law. The main idea is to handle colliding contacts before applying springs. This is because colliding contacts requires very stiff springs, which are numerically intractable.

Terzopoulos et al. [1987] used another kind of penalty force. The main idea behind their derivation comes from conservative forces, which are known to be the negative gradient of an

energy potential, that is the negative gradient of a scalar function. A scalar energy function is then designed that penalizes penetration, such that

$$c \exp\left(\frac{-\phi}{\varepsilon}\right), \tag{280}$$

where c and ε are constants used to determine the specific shape of the energy potential. The penalty force is then computed as

$$\mathbf{f}^{\text{spring}} = -\left(\frac{\nabla\phi}{\varepsilon} \exp\left(\frac{-\phi}{\varepsilon}\right) \cdot \hat{n}\right) \hat{n}\,. \tag{281}$$

Such penalty forces are called exponential springs. Exponential springs are stiffer for large displacement than linear springs, which can quickly lead to stability problems for explicit integration schemes. However, for small displacements exponential springs are less stiff than linear springs [McKenna and Zeltzer 1990].

McKenna and Zeltzer [1990] introduce a twist on penalty forces for modeling collisions that is inspired by Newton's impact law, relating pre- and post velocities through a coefficient of restitution, ε. A value of zero for the coefficient corresponds to completely inelastic collisions, and a value of 1 corresponds to fully elastic collisions. For contacts where the bodies are moving away from each other, the penalty force is multiplied by ε, such that

$$\mathbf{f}^{\text{spring}} \leftarrow \begin{cases} \varepsilon\, \mathbf{f}^{\text{spring}} & \text{if } \phi < 0 \text{ and } v_{\hat{n}} > 0 \\ \mathbf{f}^{\text{spring}} & \text{otherwise} \end{cases}. \tag{282}$$

In Jansson and Vergeest [2003], a mass-spring model is used for modeling both rigid and deformable objects, here simple springs are created when particles move within a nominal distance, and springs are deleted again when their incident particles move further away than a given fracture distance. The spring force is simply modeled as

$$\mathbf{F} = -k\left(\left\|\mathbf{x}_i - \mathbf{x}_j\right\| - l\right) \frac{\mathbf{x}_i - \mathbf{x}_j}{\left\|\mathbf{x}_i - \mathbf{x}_j\right\|}, \tag{283}$$

where k is the spring constant, $\mathbf{x}_i$ and $\mathbf{x}_j$ are the particle positions, and l is the nominal distance. Details for determining nominal and fracture distances can be found in Jansson and Vergeest [2002]. Hence, there is no need for contact points in the traditional sense, and instead contact points are simply particle pairs.

Penalty methods are popular for differentiable simulators too as they provide differentiable forces. Here we briefly describe the approach in Macklin et al. [2020b]. One view of penalty forms of contact is as a regularization of the complementarity form of the non-penetration constraint. Penalty methods of contact associate a stiff potential with the non-penetration constraint. In the simplest case this is a quadratic function of the clamped constraint error,

$$U(\mathbf{q}) \equiv \frac{1}{2} k_n \min(0, \phi)^2\,. \tag{284}$$

The associated (non-smooth) force due to this potential:

$$\mathbf{f}^{\text{spring}} \equiv -k_n \mathbf{J}^T \min(0, \phi) \tag{285}$$

where k_n controls the stiffness of the contact. We observe that as $k_n \to \infty$ it approaches a hard constraint limit. One advantage of penalty based approaches is that they can easily support nonlinear contact models. For example, applying a simple exponentiation with $p \geq 1$ gives the exponential spring equation

$$\mathbf{f}^{\text{spring}} \equiv -k_n \mathbf{J}^T \min(0, \phi)^p \,, \tag{286}$$

which produces smooth contact forces with continuous derivatives. Similar modeling was done by Geilinger et al. [2020] and found to perform better than other regularized complementarity models.

A regularized form of Coulomb friction may be expressed as the derivative of the following dissipative potential in terms of the slip velocity,

$$U_f(\mathbf{q}) \equiv \begin{cases} \frac{1}{2} k_f \|\mathbf{v}_{\hat{t}}\|^2 & k_f \|\mathbf{v}_{\hat{t}}\| < \mu \left\|\mathbf{f}^{\text{spring}}\right\| \\ \mu \left\|\mathbf{f}^{\text{spring}}\right\| \|\mathbf{v}_{\hat{t}}\| - \gamma & \text{otherwise} \end{cases} , \tag{287}$$

where the parameter k_f controls stiffness in the "stick" regime, and γ is a constant to make the potential have C0 continuity. Since we only ever evaluate the potential's gradient, γ may generally be ignored. This potential is quadratic around the origin and is linear past a certain point (in the slip regime). The force for this potential is

$$\mathbf{f}^{\text{spring}}_{\text{friction}} \equiv -\min\left(k_f, \mu \frac{\left\|\mathbf{f}^{\text{spring}}\right\|}{\|\mathbf{v}_{\hat{t}}\|}\right) \mathbf{v}_{\hat{t}} \,, \tag{288}$$

which in 1D looks like the relaxed step function.

5.2 Barrier Methods

Penalty-based methods are limited by the choice of the penalty stiffness k [Baraff 1989]. A small k allow for a easier problem to solve but lacks the stiffness needed to prevent tunneling. On the other hand, a large k will repel contacting bodies more effectively, but results in a stiffer system and is therefore harder to solve (can be unstable when using explicit methods and require more iterations when using a fully implicit approach).

Beyond this, no matter the choice of k there always exists a large enough velocity between contacting bodies that will result in tunneling. To address this problem we can turn to *barrier methods*. Barrier methods are similar to penalty-based methods but apply penalties that diverge as objects get closer. This is done through the use of so called barrier functions [Nocedal and Wright 2006] which grows to infinity as our gap shrinks to zero. Immediately, we can see one problem with such methods which is that, similar to solving with large k, special care is needed to solve such a stiff problem. Here we will briefly discuss a powerful explicit method

for handling contact through barriers. Additionally, in Section 5.3 we discuss in greater detail the recent Incremental Potential Contact [Li et al. 2020].

5.2.1 Asynchronous Contact Mechanics. We start from a simple quadratic penalty

$$g(\mathbf{q}, \eta) = \|\mathbf{x}_b - \mathbf{x}_a\| - \eta, \tag{289}$$

where η encodes the surface thickness and x_a and x_b are the contacting point. We can then use this penalty to define a contact potential

$$V(\mathbf{q}, \eta, k) = \begin{cases} \frac{1}{2} k g^2(\mathbf{q}, \eta) & \text{if } g \leq 0 \\ 0 & \text{if } g > 0 \end{cases} \tag{290}$$

where again k is the penalty stiffness.

Harmon et al. [2009] propose to layer these quadratic penalties such that their sum grows to infinite at the boundary $g(\mathbf{q}) = 0$. That is they apply a sequence of $V(\mathbf{q}, \eta(i), k(i))$ for $i = 1, 2, \ldots$, where $\eta(i)$ is monotonically decreasing and $k(i)$ is monotonically increasing with respect to i. Harmon et al. [2009] choose $k(l) = k(1)i^3$ and $\eta(l) = \eta(1)i^{-1/4}$, for a given initial $k(1)$ and $\eta(1)$. The sum of these potentials then form a barrier because $\sum_i k(i)\eta(i)^2 \to \infty$.

For each penalty layer, we can apply friction as in Section 5.1.4. This applies an increasing friction force to effectively handle high-speed tangential motion.

To handle the increasingly stiff problem (i.e., increasingly large forces/impulses), we can use an adaptive time-stepping method or as Harmon et al. [2009] propose step each contacting point asynchronously. While this method is more effective than a simple global adaptive time-stepping, it still leads to exceedingly high running times. Subsequent works [Ainsley et al. 2012; Ni et al. 2015] have improved upon the original method, but despite the guarantees provide by this method, the high running-time has limited its applications.

5.3 Incremental Potential Contact

Recently there have been several advancements in robust contact handling by applying barrier potentials within an optimization-based implicit time integration [Chen et al. 2022; Ferguson et al. 2021; Lan et al. 2022, 2021; Li et al. 2020, 2021, 2022a,b; Zhao et al. 2022]. As we saw in Section 4.2.2, the implicit time integration problem can be written as a optimization over nodal positions $\mathbf{q}$ with velocities $\mathbf{u}$. For ease of reading let us restate Equation 265 here:

$$\mathbf{q}^{t+h} = \arg\min_{\mathbf{q}} f(\mathbf{q}, \mathbf{q}^t, \mathbf{u}^t) \quad \text{s.t.} \quad \phi_k(\mathbf{q}) \geq 0 \quad \forall k \in C,$$

where C is the set of all "active" contacts and ϕ_k is a gap function encoding our contacts (see Figure 33 for example gap functions). Here contacts are treated as inequality constraints (friction is omitted for now). IPC instead transforms these constraints into barrier potentials

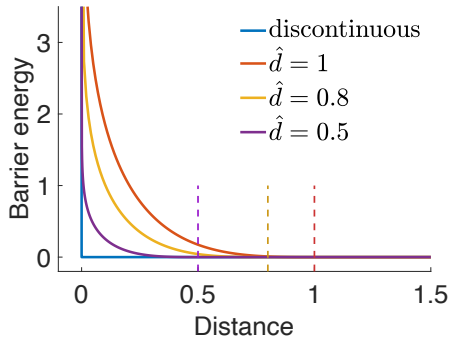


Fig. 32. IPC's clamped barrier approximation to the discontinuous function (blue) improves as we make our geometric accuracy threshold, $\hat{d}$, shrinks to zero.

added to the objective function (similar to interior-point methods [Nocedal and Wright 2006])

$$\mathbf{q}^{t+h} = \arg\min_{\mathbf{q}} f(\mathbf{q}, \mathbf{q}^t, \mathbf{u}^t) + \kappa \sum_{k \in C} b(d_k(\mathbf{q}), \hat{d}), \tag{291}$$

where d_k is the unsigned distance between the contacting geometry (e.g., point and triangle or two edges) and

$$b(d, \hat{d}) = \begin{cases} -(d - \hat{d})^2 \ln\left(\frac{d}{\hat{d}}\right), & 0 < d < \hat{d} \\ 0 & d \geq \hat{d} \end{cases}, \tag{292}$$

is a smoothly clamped barrier which vanishes as the distance d approaches a value of $\hat{d}$ (see Figure 32). This $\hat{d}$ acts as an intuitive parameter to control the performance versus accuracy of the simulation. By decreasing $\hat{d}$, accuracy increases because forces at unphysical distances vanish, but the sharper barriers requires more optimization iterations. On the other hand, increasing $\hat{d}$ smooths the problem leading to and easier optimization, but the contact forces activate at a possibly non-realistic distance.

For simplicity active contacts can be all possible contacts (quadratic number in terms of the surface triangles). However, thanks to the local support provide by the clamped barrier function in Equation 292, we can instead use a fast spatial data structure (e.g., spatial hash) to cull out contacts that will result in zero force (i.e., ignore contacts with distance greater than $\hat{d}$).

The benefit of this approach is it is a unconstrained minimization. This can be robustly solved using Newton's method (Section 5.3.2) with line search (Section 5.3.3). Importantly, by using CCD within the line-search we can provide guarantees that every step will be intersection-free independent of the parameter choices.

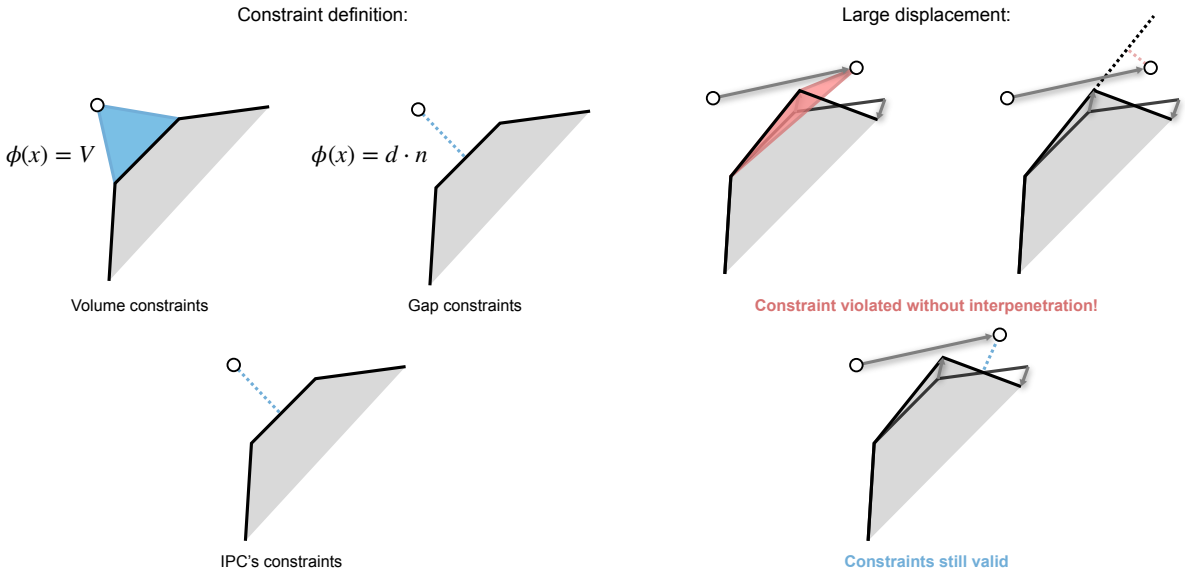


Fig. 33. Traditional contact constraints come in many flavors. For example, volume constraints [Kane et al. 1999; Müller et al. 2015] compute a signed volume of the tetrahedron formed by a point-triangle or edge-edge pair (illustrated in 2D). Also popular, gap constraints [Harmon et al. 2008; Otaduy et al. 2009a; Verschoor and Jalba 2019] define constraints as the signed distance between point-triangle or edge-edge pairs by projecting to the plane spanned by the triangle or three points of the edge-edge pair. However, when undergoing large displacements these constraints can be violated without interpenetrations. This leads to artificially smaller steps due to being over-constrained. Instead IPC uses an unsigned distance as constraints. This is globally consistent as the closest point is determined on each evaluation.

A limitation of this model is that we have to be careful with the input geometry and selection of $\hat{d}$. Not only does the initial configuration need to be intersection-free but there needs to be an initial nonzero gap. Furthermore, using an tiny initial gap, relative to $\hat{d}$, will cause the gaps to expand as the (possibly large) contact forces push them apart.

5.3.1 Consistent Distance Functions. Traditionally distance-based constraints $d(\mathbf{q}) \geq 0$ are evaluated using a signed distance function. However, IPC utilizes unsigned (e.g., standard Euclidean) distances. This is important as it ensures a consistent constraint definition as contacts shift through time (e.g., Figure 33).

In 3D, we need to compute the minimum distance between triangles. As long as the triangles are not intersecting, this can be equivalently broken down into computing the minimum of the distance between vertices and triangles and between a pairs of edges. The distance between a point and triangle can be expressed as a small constrained minimization over Barycentric

coordinates u and v

$$\begin{aligned} d_{\mathrm{PT}}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4) = &\min_{\alpha,\beta} \|((1-u-v)\mathbf{x}_1 + u\mathbf{x}_2 + v\mathbf{x}_3) - \mathbf{x}_4\| \\ &\text{s.t.} \quad u \geq 0, \quad v \geq 0, \quad u+v \leq 1, \end{aligned} \tag{293}$$

where again $\mathbf{x}_1$, $\mathbf{x}_2$, and $\mathbf{x}_3$ are the vertices of the triangle and $\mathbf{x}_4$ is the point. Similarly for a pair of edges parameterized by a and b the distance can be computed as

$$\begin{aligned} d_{\mathrm{EE}}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4) = &\min_{a,b} \|(\mathbf{x}_1 + a(\mathbf{x}_2 - \mathbf{x}_1)) - (\mathbf{x}_3 + b(\mathbf{x}_4 - \mathbf{x}_3))\| \\ &\text{s.t.} \quad 0 \leq a, b \leq 1, \end{aligned} \tag{294}$$

where, in this context, $\mathbf{x}_1$ and $\mathbf{x}_2$ are the end points of the first edge and $\mathbf{x}_3$ and $\mathbf{x}_4$ are the end points of the second edge.

Conveniently, both of these constrained minimizations have a closed form solution and the solution depend on which sub-element are closest. For example if the point is closest to an edge of the triangle, then the distance can be computed as the distance between the point and the analytical line collinear with the edge. For a full breakdown of each possible case we refer to Li et al. [2020] and the implementation provided in the IPC Toolkit [Ferguson et al. 2020].

5.3.2 Optimization-based Time Integration. Let us start by defining

$$f(\mathbf{q}) \equiv E(\mathbf{q}, \mathbf{q}^t, \mathbf{u}^t) + \kappa \sum_{k \in C} b(d_k(\mathbf{q}), \hat{d})$$

as our objective function to the minimization in Equation 291. Because this is an unconstrained optimization we can simply use a gradient descent or Newton's method to find the minimum.

Data: $\mathbf{q}_0, f(\mathbf{q})$
Result: $\mathbf{q}^\star$
$i \leftarrow 0$;
$C \leftarrow \text{ActiveContactSet}(\mathbf{q}_i)$;
while $\|\nabla f(\mathbf{q}_i)\| > \epsilon$ **do**
  $\Delta\mathbf{q}_i \leftarrow -\nabla f(\mathbf{q}_i)$;
  $\alpha \leftarrow \text{LineSearch}(\mathbf{q}_i, \Delta\mathbf{q}_i)$;
  $\mathbf{q}_{i+1} \leftarrow \mathbf{q}_i + \alpha\Delta\mathbf{q}_i$;
  $C \leftarrow \text{ActiveContactSet}(\mathbf{q}_i)$;
  $i \leftarrow i + 1$;
end
$\mathbf{q}^\star \leftarrow \mathbf{q}_i$;

Algorithm 17: Gradient descent with line-search

Data: $\mathbf{q}_0, f(\mathbf{q})$
Result: $\mathbf{q}^\star$
$i \leftarrow 0$;
$C \leftarrow \text{ActiveContactSet}(\mathbf{q}_i)$;
while $\|\nabla f(\mathbf{q}_i)\| > \epsilon$ **do**
  $\mathbf{H} \leftarrow \text{PDProject}(\nabla^2 f(\mathbf{q}_i))$;
  $\Delta\mathbf{q}_i \leftarrow -\mathbf{H}^{-1}\nabla f(\mathbf{q}_i)$;
  $\alpha \leftarrow \text{LineSearch}(\mathbf{q}_i, \Delta\mathbf{q}_i)$;
  $\mathbf{q}_{i+1} \leftarrow \mathbf{q}_i + \alpha\Delta\mathbf{q}_i$;
  $C \leftarrow \text{ActiveContactSet}(\mathbf{q}_i)$;
  $i \leftarrow i + 1$;
end
$\mathbf{q}^\star \leftarrow \mathbf{q}_i$;

Algorithm 18: Projected Newton with line-search

A simple gradient descent method. To start off simple we can minimize $f(\mathbf{q})$ by using a gradient descent method (Algorithm 17). This requires computing the gradient $\nabla f(\mathbf{q}_i)$ of our objective, and then use this as an update direction $\Delta\mathbf{q}_i$ that decreases the energy. We use line-search to find a intersection-free descent step length α (more on this in the next section).

Our function $f(\mathbf{q})$ (and namely our distances $d(\mathbf{q})$) need to be at least C^1 (continuous and differentiable).

A better method: Projected Newton. In general we want to use Newton's method because it provides an improved rate of convergence (close to a minimum) and therefore takes fewer iterations. While Newton does not promise global convergence, the benefit of time dependent simulation is the previous time step usually provides a good warm start. Here we can also see a trade off of the time step size: as the time step size increase the further the initial positions $\mathbf{q}_0$ will be from $\mathbf{q}^\star$ (i.e., larger timesteps require more optimization iterations).

Algorithm 18 provides a basic outline of the solve. We need to compute the gradient $\nabla f(\mathbf{q}_i)$ *and Hessian* $\nabla^2 f(\mathbf{q}_i)$ of our objective. These are then used to find an update direction $\Delta\mathbf{q}_i$ that decreases the energy.

Projecting to Positive Definite. Notice the Hessian is projected to positive definite (PD). This is to ensure $\Delta\mathbf{q}_i$ is a descent direction ($\Delta\mathbf{q}_i \cdot \nabla f(\mathbf{q}_i) < 0$). Formally this is, if $\mathbf{H} \in \mathbb{R}^{n\times n}$ is PD (i.e., $\mathbf{x}^T\mathbf{H}\mathbf{x} > 0 \quad \forall x \neq \mathbf{0}$), then $\Delta\mathbf{q} \cdot \nabla f = -\mathbf{H}^{-1}\nabla f \cdot \nabla f < 0$.

Proof. $\mathbf{x}^T\mathbf{H}\mathbf{x} > 0 \quad \forall \mathbf{x} \neq \mathbf{0} \implies \mathbf{x}^T\mathbf{H}^{-1}\mathbf{x} > 0 \quad \forall \mathbf{x} \neq \mathbf{0}$
$\mathbf{x}^T\mathbf{H}^{-1}\mathbf{x} = \mathbf{H}^{-1}\mathbf{x} \cdot \mathbf{x} > 0 \implies \mathbf{H}^{-1}\nabla f \cdot \nabla f > 0 \implies -\mathbf{H}^{-1}\nabla f \cdot \nabla f = \Delta\mathbf{q} \cdot \nabla f < 0$ □

The projection to PD is done by first projecting the elasticity, barrier, and friction Hessians (soon to come in Section 5.3.4) to positive semi-definite (PSD). Then when we add the lumped mass matrix (Hessian of the inertia term in f) which is a PD diagonal matrix, we get a resulting matrix $\mathbf{H}$ that is PD. Alternatively, in the static case we can add a small scaled identity to the projected PSD Hessian. You can think of this as blending between the full Newton direction and the gradient descent direction to ensure progress is made toward a minimum.

Projecting a matrix to PSD can be done by computing its Eigendecomposition and then setting all negative Eigenvalues to zero. However, computing a full Eigendecomposition can be prohibitively expensive. To avoid this large cost, we can instead project the local Hessian blocks rather than the entire matrix. This is, when we compute the Hessian of the k^{th} contact term (which is a small 12×12 matrix at most) we project it to PSD and then assemble the full matrix as usual. The same can be done for the elasticity Hessian on a per element bases [Teran et al. 2005].

A limitation of this local projection strategy is that it might be overkill as the full matrix might be PSD on its own, and further while projecting to PD guarantees a descent direction it can worsen the rate of convergence of the method (less than the expected quadratic rate of convergence from a standard Newton's method).

Because there are no slack variables, the linear systems is only the size of the dimension times the number of nodes (when computing nodal displacements). Assembling this system requires adding an entry per contacting pair (e.g., a 12×12 matrix for a single point-triangle collision) in addition to the standard FEM matrix assembly.

This system can be large and the sparsity pattern changes dynamically (preventing pre-factorization), so solving the linear system becomes the dominating component as the scene grows in number of elements.

5.3.3 CCD-Aware Filtered Line-Search. A critical part of the IPC algorithm is ensuring the optimization does not take a step that jumps over the barrier. To accomplish this CCD is added to the line-search within the optimization.

Data: $\mathbf{q}, \Delta\mathbf{q}, f(\mathbf{q})$
Result: α
$\alpha \leftarrow 1$;
while $f(\mathbf{q}) \leq f(\mathbf{q} + \alpha\Delta\mathbf{q})$ **do**
  $\alpha \leftarrow \alpha/2$;
  $C \leftarrow \text{ActiveContactSet}(\mathbf{q} + \alpha\Delta\mathbf{q})$;
end

Algorithm 19: Backtracking line-search

Data: $\mathbf{q}, \Delta\mathbf{q}, f(\mathbf{q})$
Result: α
$\alpha \leftarrow \text{CCD}(\mathbf{q}, \mathbf{q} + \Delta\mathbf{q})$;
while $f(\mathbf{q}) \leq f(\mathbf{q} + \alpha\Delta\mathbf{q})$ **do**
  $\alpha \leftarrow \alpha/2$;
  $C \leftarrow \text{ActiveContactSet}(\mathbf{q} + \alpha\Delta\mathbf{q})$;
end

Algorithm 20: CCD-aware backtracking line-search

A *line-search* is a search for a minimum (or acceptable decrease in energy) along a one-dimensional space. In our case, this means finding a step length $\alpha \in [0, 1]$ that decreases the energy between $\mathbf{q}_i$ and $\mathbf{q}_i + \Delta\mathbf{q}_i$. One efficient form of line-search is backtracking (Algorithm 19). In backtracking line-search, α is initialized to one. Then α is repeatedly halved until a descent step ($f(\mathbf{q}_i + \alpha\Delta\mathbf{q}_i) < f(\mathbf{q}_i)$) is found.

For those knowledgeable about numerical methods, you will see that this is a very relaxed form of the the Armijo rule [Armijo 1966]: $f(x_i + \alpha\Delta x_i) \leq f(x_i) + c_1\alpha\Delta x_i^{\mathrm{T}}\nabla f(x_i)$ where $\Delta x_i^{\mathrm{T}}\nabla f(x_i) < 0$. The method described here is the most relaxed form of this condition ($c_1 = 0$, so any decrease in the objective is sufficient).

To incorporate CCD into a backtracking line-search (Algorithm 20), the initial value of α is set to the minimum time of impact. By doing this it is guaranteed that all $\alpha \leq \text{CCD}(\mathbf{q}, \mathbf{q} + \Delta x)$ are intersection free.

5.3.4 Variational Friction Model. Up to this point we have only discussed friction-less contact. In order to add friction into the IPC model, we need to derive a potential such that its gradient generates the frictional forces. Several others have looked at the problem of defining a smooth friction model [Geilinger et al. 2020; Macklin et al. 2020b] in particular for its applications in differentiable simulation. Here we will focus solely on the model proposed by Li et al. [2020].

Starting from the force perspective, we can defines friction forces variationally by maximizing the dissipation rate subject to the Coulomb constraint. That is for the k^{th} contact, compute the

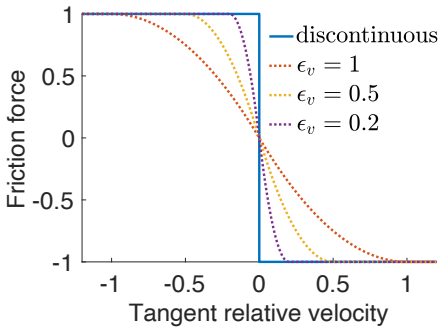


Fig. 34. The smooth transition from static to dynamic friction converges to the exact relation as the frictional accuracy threshold, ϵ_v gets smaller.

friction force F_k as

$$F_k(\mathbf{q}, \mathbf{u}) = T_k(\mathbf{q}) \underset{\beta \in \mathbb{R}^2}{\operatorname{argmin}} \beta^T \mathbf{u}_k \quad \text{s.t.} \quad \|\beta\| \leq \mu N_k(\mathbf{q}), \tag{295}$$

where $T_k(\mathbf{q}) \in \mathbb{R}^{3n \times 2}$ is the sliding basis at the point of contact, $N_k(\mathbf{q})$ is the magnitude of the normal force, and μ is the local friction coefficient. To remove the dependency on $\mathbf{u}$ we can use our time integration method to compute the velocity from change in position (e.g., $\mathbf{u} = \frac{1}{h}(\mathbf{q} - \mathbf{q}^t)$ for implicit Euler).

Notice that both T_k and N_k depend on $\mathbf{q}$. This is because as the position of the vertices move the tangential basis of the contact change. For example, in a point-triangle contact the tangential basis lies co-planar with the triangle, so the basis moves as the the triangle deforms. Similarly the normal force grows as the distance between elements shrinks and vice versa.

Let $\Delta\tilde{\mathbf{q}}_k = T_k(\mathbf{q})(\mathbf{q} - \mathbf{q}^t) \in \mathbb{R}^2$ be the relative sliding displacement of the k^{th} contacting point over the timestep. These friction forces have the form $F_k = -\mu N_k(\mathbf{q}) T_k(\mathbf{q}) f$, where in the dynamic case ($\|\mathbf{u}\| > 0$), $f = \frac{\Delta\tilde{\mathbf{q}}_k}{\|\Delta\tilde{\mathbf{q}}_k\|}$, but in the static case ($\|\mathbf{u}\| = 0$), f can be any vector in the 2D unit disk. This demonstrates the first challenge when integrating such forces: the transition between sticking and sliding comes with a discontinuous jump in both the force magnitude and direction.

To address this, the transition between static and dynamic friction is mollified using a function

$$m_{\epsilon_v}(s) = \begin{cases} -\frac{s^2}{\epsilon_v^2 h^2} + \frac{2s}{\epsilon_v h}, & |s| < h\epsilon_v \\ 1, & \text{otherwise} \end{cases} \tag{296}$$

where s is the input (speed) and, similar to $\hat{d}$ in the contact formulation, ϵ_v controls the accuracy of the model with smaller values approaching the full discontinuous transition (see Figure 34).

This lead us to a smooth friction force defined as

$$F_k(\mathbf{q}) = -\mu N_k(\mathbf{q}) T_k(\mathbf{q}) \frac{m_{\epsilon_v}(\|\Delta\tilde{\mathbf{q}}_k\|)}{\|\Delta\tilde{\mathbf{q}}_k\|} \Delta\tilde{\mathbf{q}}_k. \tag{297}$$

Notice that for the static case ($\|\Delta\tilde{\mathbf{q}}_k\| = 0$) it may seem that we have a division by zero, but in fact by dividing the first case in m_{ϵ_v} by s:

$$\frac{1}{s}\left(\frac{-s^2}{\epsilon_v^2 h^2} + \frac{2s}{\epsilon_v h}\right) = \frac{-s}{\epsilon_v^2 h^2} + \frac{2}{\epsilon_v h},$$

we see that for $s = 0$ this function is well defined.

There still remains one challenge with this formulation however. Because N_k and T_k depend on $\mathbf{q}$, there does not exist a well defined potential whose derivative produces the friction force. To resolve this, Li et al. [2020] proposes to *lag* these values to their values at the beginning of the time-step (or optimization).

By setting N_k and T_k to some fixed value N_k^{lagged} and T_k^{lagged} we can define a dissipative potential as

$$D_k(x) = \mu N_k^{\text{lagged}} M_{\epsilon_v}(\|\Delta\tilde{\mathbf{q}}_k\|), \tag{298}$$

where $M'_{\epsilon_v}(s) = m_{\epsilon_v}(s)$ and the friction force is related to this potential by $F_k(x) = -\nabla D_k(x)$. This dissipative potential can then be added to the objective function in Equation 291. Again this leads to an unconstrained optimization that can be solved with Newton's method using a CCD-aware line-search.

For improved accuracy, the lagged values can be updated after the optimization and the new objective can be minimized again. This process repeats until a force balance at the updated positions is found. Unfortunately, IPC provides no guarantee that such convergence will occur. This is the major limitation of this approach, but in practice this rarely occurrence.

A limitation of this model is that for the static case the zero-force solution is preferred. In other approaches, one would try to use the static friction force that will make things stick [Bridson et al. 2002]. Additionally, the mollified friction models may suffer from a viscous drag. It can be controlled by the ϵ_v parameter but it can never be made to disappear. Geilinger et al. [2020] demonstrate this for a similar smooth friction model.

5.3.5 Further Topics. The above covers the basics of the IPC approach to simulating deformable bodies. Extensions to this method have been proposed for rigid [Ferguson et al. 2021], nearly rigid [Lan et al. 2022], reduced models [Lan et al. 2021], and more. We point the interested reader to these works for more information.

As a shameless plug, we point interested readers to the IPC Toolkit [Ferguson et al. 2020] for a reference implementation of all these components discussed here for use in any exist simulator or to PolyFEM [Schneider et al. 2019] for a full finite element simulation using IPC (through the IPC Toolkit).

6 ANISOTROPIC FRICTION MODELING

The isotropic Coulomb friction model is used by many rigid body simulations in the field of computer graphics [Bender et al. 2014], and indeed the focus of many works has been on the difficult problem of formulating and solving isotropic Coulomb frictional contact and its approximations. However, many real-world surfaces have characteristics that result in anisotropic frictional behavior. For instance, consider surfaces with ridges or grooves. The directional nature of the geometric features means that objects sliding on the surface will experience frictional resistance that is dependent on the sliding direction.

There has been comparatively less attention in computer graphics on anisotropic models of frictional contact. Nevertheless, various frameworks have been developed in recent years for simulating anisotropic friction in computer graphics. In this chapter, we dive into the details of a handful of approaches that address simulating anisotropic friction effects. Specifically, this chapter covers friction tensors [Pabst et al. 2009], the Matchstick model [Erleben et al. 2020], and a texture-based model where friction cones are computed from the normal map textures of each colliding surface [Andrews et al. 2021]. Additional implementation details about using these models with numerical solvers from Chapter 3 are also presented.

As we saw in Chapter 1, many physics simulators use friction models expressed as cones or generic sets of feasible friction forces. Combining feasible set descriptions with extra constraints, such as the principle of maximum dissipation, allows us to compute the friction forces at a given instant in time. The benefit of such friction descriptions is that they permit an easy implementation within a simulator. One can define a projection operator, and iteratively project the friction force onto the closest feasible value. This property is the governing principle about which many friction models can be defined. Let us now take a look at how these projection operators can used for anisotropy.

6.1 The Matchstick Model

The Matchstick model produces a cone based on surface structural directions from each body at the point of contact. Figure 35 shows simulation examples using this model. We begin by examining how to generate an anisotropic friction cone based on the kinematic state of two bodies, A and B. The main idea here is to create a convex cone based on the relative orientation between the two surfaces.

6.1.1 Friction Cone Modeling. An important difference of anisotropic friction compared to isotropic friction is that the local material frame of each surface is considered. Regarding the Matchstick model, the structural directions on each surface are defined with respect to some material frame. The directions can be interpreted as fiber directions or as micro-scale geometry features, such as grooves. While other materials (e.g., cloth with warp and weft directions) can be seen as having multiple material directions. For the sake of brevity, we limit our presentation to the case of a single material direction.

Fig. 35. The Matchstick model allows for control of frictional behavior in situations such as tire ground contact, a hopper, and soft robotic gripping, where each example shown here also uses different simulators with different contact solvers (Vortex, PROX, and Flex).

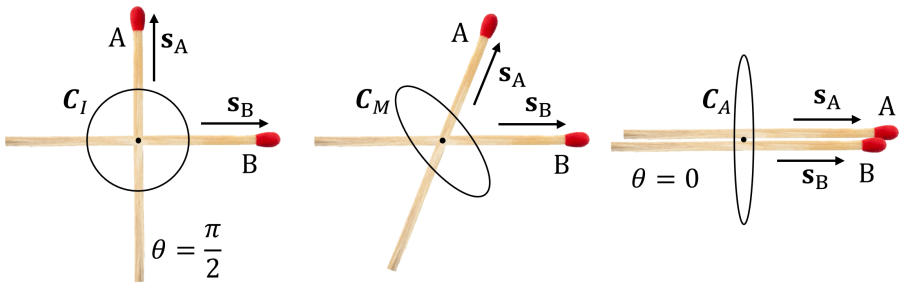


Fig. 36. The angle between matchsticks determines the friction cone. The Matchstick model interpolates between isotropic $\mathbf{C}_I$ and anisotropic $\mathbf{C}_A$ extremes (see Equations 302, 305, and 308).

For any given contact frame (see Figure 4), the material direction of each surface can be written as unit vectors $\mathbf{s}_\mathrm{A}$ and $\mathbf{s}_\mathrm{B}$, in the space of the tangent plane. These directions depend on the point of contact on each surface, and in a typical implementation they can be stored in a texture or generated procedurally. Given a simple parameterized cone generator, $\mathcal{G}_{\mathrm{A}\leftrightarrow\mathrm{B}}$, that computes a friction cone that describes the set of allowable friction forces based on the set of local parameters, we denote the generated cone by the symbol $\mathcal{F}_{\mathrm{A}\leftrightarrow\mathrm{B}}$. Relaxing the notation and not explicitly writing the object pairs, the generator can be defined as

$$\mathcal{F} \equiv \mathcal{G}\left(\lambda_{\hat{n}}, \mathbf{v}_{\hat{t}}, \mathbf{s}_\mathrm{A}(\mathbf{p}), \mathbf{s}_\mathrm{B}(\mathbf{p})\right) . \tag{299}$$

The normal force magnitude is given by $\lambda_{\hat{n}}$. The above parameterization could be extended with even more parameters to account for other dependencies. However, $\mathcal{F}$ here simply refers to any parametric friction cone model to enhance readability.

6.1.2 The Matchstick Model. Next, we derive the Matchstick model for the framework of anisotropic friction outlined in the previous section. The model is a phenomenological one, and it is based on several observations about how surfaces with structural features interact as they slide against each other (see Figure 36). Briefly, the frictional behavior appears isotropic for a pair of matchsticks when moving in different direction, but keeping structure directions orthogonal. Whereas the behavior is anisotropic for a pair of matchsticks when sliding in

different directions while keeping structure directions parallel. The model thus interpolates between isotropic and anisotropic Coulomb behaviors using the minimum angle between the structural directions.

With the simplifying assumption that the sign of the structure vector $\mathbf{s}$ can be ignored (i.e., if the friction only depends on the orientation and not the direction), then the Matchstick model interpolation parameter is based on the angle θ, and it is defined as

$$d \equiv 1 - \frac{2}{\pi} \underbrace{\cos^{-1} |\mathbf{s}_\mathrm{A} \cdot \mathbf{s}_\mathrm{B}|}_{\equiv\theta} . \tag{300}$$

Let the friction force in the world frame be given by $\mathbf{f}$ and let the coefficient of friction for a planar isotropic Coulomb friction cone be μ, equal to the tangent $\hat{t}$ and binormal $\hat{b}$ direction coefficients for the isotropic cone, i.e.,

$$\mu^I_{\hat{t}} = \mu^I_{\hat{b}} = \mu , \tag{301}$$

then the isotropic cone is defined by the relationship

$$\mathbf{f}^T \mathbf{R}^T \underbrace{\begin{bmatrix} \frac{1}{\mu}^2 & 0 \\ 0 & \frac{1}{\mu}^2 \end{bmatrix}}_{\equiv \mathbf{C}_I} \mathbf{R}\mathbf{f} \leq \lambda_{\hat{n}}{}^2 . \tag{302}$$

Here, $\mathbf{R}$ can be any rotation matrix for an isotropic cone, but for consistency we define $\mathbf{R}$ as the 2D rotation of $(\mathbf{s}_\mathrm{A} + \mathbf{s}_\mathrm{B})$ onto the $\hat{t}$ axis, assuming that $\mathbf{s}_\mathrm{A} \cdot \mathbf{s}_\mathrm{B}$ is positive (given that only the orientation is needed, we can swap the sign of one of the vectors to ensure positive dot product). This implies that the tangent plane vectors are defined as

$$\hat{t} \equiv \frac{\mathbf{s}_\mathrm{A} + \mathbf{s}_\mathrm{B}}{\|\mathbf{s}_\mathrm{A} + \mathbf{s}_\mathrm{B}\|} , \tag{303}$$

$$\hat{b} \equiv \hat{n} \times \hat{t} . \tag{304}$$

By construction, $\hat{t}$ and $\hat{b}$ are the major and minor axis of the anisotropic ellipse cone, respectively. And while the planar rotation matrix $\mathbf{R}$ is useful for explaining the model, the column vectors $\hat{t}$ and $\hat{b}$ are needed when assembling the Jacobian matrix.

For anisotropic Coulomb friction, the coefficients of friction are $\mu^A_{\hat{t}} \leq \mu^A_{\hat{b}}$, where the A superscript denotes anisotropy. Similar to the isotropic case, the anisotropic cone is defined by the quadratic inequality

$$\mathbf{f}^T \mathbf{R}^T \underbrace{\begin{bmatrix} \frac{1}{\mu^A_{\hat{t}}}^2 & 0 \\ 0 & \frac{1}{\mu^A_{\hat{b}}}^2 \end{bmatrix}}_{\equiv \mathbf{C}_A} \mathbf{R}\mathbf{f} \leq \lambda_{\hat{n}}{}^2 . \tag{305}$$

Data: Structure directions $\mathbf{s}_\mathrm{A}$, $\mathbf{s}_\mathrm{B}$, contact normal $\hat{n}$, isotropic friction μ, extreme friction coefficients $\mu^A_{\hat{t}}$, $\mu^A_{\hat{b}}$.

Result: Contact plane vectors $\hat{t}$, $\hat{b}$, and coefficients of friction $\mu_{\hat{t}}$, $\mu_{\hat{b}}$.

if $\mathbf{s}_\mathrm{A} \cdot \mathbf{s}_\mathrm{B} < 0$ **then**
  $\mathbf{s}_\mathrm{B} \leftarrow -\mathbf{s}_\mathrm{B}$;
end
$\theta \leftarrow \cos^{-1}(\mathbf{s}_\mathrm{A} \cdot \mathbf{s}_\mathrm{B})$;
$d \leftarrow 1 - \frac{2\,\theta}{\pi}$;
$\mu_{\hat{t}} \leftarrow d\,\mu^A_{\hat{t}} + (1-d)\,\mu$;
$\mu_{\hat{b}} \leftarrow d\,\mu^A_{\hat{b}} + (1-d)\,\mu$;
$\hat{t} \leftarrow \frac{\mathbf{s}_\mathrm{A}+\mathbf{s}_\mathrm{B}}{\|\mathbf{s}_\mathrm{A}+\mathbf{s}_\mathrm{B}\|}$;
$\hat{b} \leftarrow \hat{n} \times \hat{t}$;

Algorithm 21: MatchStickFrictionGenerator The generator gives both the world orientation of the friction cone as well as the coefficients of friction, which is an advantage when working with an analytic cone that is fully described by these parameters.

In the case of the Matchstick model, the friction cone generator $\mathcal{G}$ computes the coefficients by a spherical linear interpolation between the isotropic and anisotropic cones, such that

$$\mu_{\hat{t}} \equiv d\,\mu^A_{\hat{t}} + (1-d)\;\mu\,, \tag{306}$$

$$\mu_{\hat{b}} \equiv d\,\mu^A_{\hat{b}} + (1-d)\;\mu\,. \tag{307}$$

Thus, the actual Coulomb cone based on the kinematic configuration is given by:

$$\mathbf{f}^T\mathbf{R}^T \underbrace{\begin{bmatrix} \frac{1}{\mu_{\hat{t}}}^2 & 0 \\ 0 & \frac{1}{\mu_{\hat{b}}}^2 \end{bmatrix}}_{\equiv \mathbf{C}_M} \mathbf{R}\mathbf{f} \le \lambda_n^2\,. \tag{308}$$

The Matchstick friction model generator is outlined in Algorithm 21. A nice aspect about the model is that it has a small memory footprint and fast computational complexity for generating the cone and using it at run time. In the next section we will outline how the Matchstick model can be incorporated into different modeling approaches and solvers, before we conclude our presentation with some implementation notes. Thereby we demonstrate the Matchstick model to be a general model suitable for any type of simulator.

6.1.3 Fixed-point Solvers. We will now present how the Matchstick model can be used in a solver. Iterative methods for contact force computations are the natural choice for arbitrary friction cones. We will first consider the class of methods based on proximal operators as

described in Section 3.4. In methods based on proximal operators, the next feasible friction impulse iterate ${\boldsymbol{\lambda}_{\hat{t}}}^{k+1}$ is given by projecting the current guess ${\boldsymbol{\lambda}_{\hat{t}}}^{k}$ onto the friction cone, such that

$$ {\boldsymbol{\lambda}_{\hat{t}}}^{k+1} \leftarrow \text{prox}_{\mathcal{F}} \left({\boldsymbol{\lambda}_{\hat{t}}}^{k} - r_{\hat{t}} \, {\mathbf{v}_{\hat{t}}}^{k} \right) , \tag{309} $$

and tangential forces $\boldsymbol{\lambda}_{\hat{t}} = \begin{bmatrix} \lambda_{\hat{t}} & \lambda_{\hat{b}} \end{bmatrix}^T$ are projected into the anisotropic cone

$$ \mathcal{F} \equiv \left\{ \boldsymbol{\lambda}_{\hat{t}} \;\middle|\; \left(\frac{\lambda_{\hat{t}}^2}{\mu_{\hat{t}}^2} + \frac{\lambda_{\hat{b}}^2}{\mu_{\hat{b}}^2} \right) \leq {\lambda_{\hat{n}}}^2 \right\} . \tag{310} $$

Here, k is the iteration index and $r_{\hat{t}}$ is a scalar relaxation parameter known as the r-factor. The algorithm then proceeds using a sweeping process. First, Equation 299 is used to instantiate the current friction cone, $\mathcal{F}$, and then that cone is used in the proximal point update given by Equation 309. Algorithm 22 illustrates how a PROX-based block Gauss-Seidel variant is modified to accommodate such friction models. Observe that the only change is the addition of line 8 in the algorithm, before the friction proximal step, which instantiates the friction model.

6.1.4 LCP-based Solvers. For LCP based approaches, one may discretize a parameterized cone by shooting rays from the origin of the limit surface in various directions and use the limit surface points to build a polygonal approximation to the generated cone, such as the one shown in Figure 6. Each facet of the polyhedral cone will match one complementary constraint in the LCP model. While it is trivial to generate the polygonal facets, the main drawback is that one may need many facets to obtain a good approximation. The memory footprint of the polyhedral LCP has quadratic scaling with the number of constraints and the solver time will suffer accordingly. Hence, nonlinear complementary formulations can be more attractive for parametric cones.

6.1.5 NCP-based Solvers. Let us consider how to use a parameterized cone $\mathcal{F}$ in a Newton type framework by using non-smooth functions, such as the Fischer-Burmeister function [Macklin et al. 2019]. Without loss of generality, assume we have any type of complementary function, $\psi(a, b) : \mathbb{R} \times \mathbb{R} \mapsto \mathbb{R}$ such that

$$ 0 \leq a \perp b \geq 0 \quad \Leftrightarrow \quad \psi(a, b) = 0 \, . \tag{311} $$

Using an implicit limit surface of the cone we let η be the corresponding implicit function of $\mathcal{F}$,

$$ \eta \equiv \mathbf{f}^T \mathbf{R}^T \mathbf{C}_M \mathbf{R} \mathbf{f} - {\lambda_{\hat{n}}}^2 . \tag{312} $$

Then by the principle of maximal dissipation we can write

$$ \nabla \eta(\mathbf{f}) = -\beta \, \mathbf{v}_{\hat{t}} \, , \tag{313} $$

Data: Indices of all contacts K, indices of all bodies B, and $\mathbf{J}$, $\mathbf{M}$, $\mathbf{b}$, r, $\boldsymbol{\lambda}^0$, ν.
Result: $\boldsymbol{\lambda}^k$

$\left(k, \boldsymbol{\lambda}^k, \epsilon^k\right) \leftarrow \left(0, \boldsymbol{\lambda}^0, \infty\right)$;
while *not converged* **do**
  $\mathbf{w} \leftarrow \mathbf{M}^{-1}\mathbf{J}^T\boldsymbol{\lambda}^k$;
  foreach $i \in \mathsf{K}$ **do**
    $\mathsf{I} \equiv$ indices of block $\mathsf{N}_i, \mathsf{F}_i$;
    $\mathbf{z}_\mathsf{I} \leftarrow \boldsymbol{\lambda}^k_\mathsf{I} - r\left(\mathbf{J}_{\mathsf{IB}}\,\mathbf{w} + \mathbf{b}_\mathsf{I}\right)$;
    $\lambda_n^{k+1} \leftarrow \mathrm{prox}_{\mathcal{N}_i}\left(\mathbf{z}_n\right)$;
    $\mathcal{F} \leftarrow \mathcal{G}(\lambda_{\hat{n}_i}^{k+1}, \ldots)$;
    $\boldsymbol{\lambda}_{\hat{t}_i}^{k+1} \leftarrow \mathrm{prox}_{\mathcal{F}}\left(\mathbf{z}_{\mathsf{F}_i}\right)$;
    $\mathbf{w} \leftarrow \mathbf{w} + \left(\mathbf{M}^{-1}\mathbf{J}^T\right)_{\mathsf{BI}}\left(\boldsymbol{\lambda}^{k+1}_\mathsf{I} - \boldsymbol{\lambda}^k_\mathsf{I}\right)$;
  end
  $\epsilon^{k+1} = \left\|\boldsymbol{\lambda}^{k+1} - \boldsymbol{\lambda}^k\right\|_\infty$;
  if $\epsilon^{k+1} > \epsilon^k$ **then**
    $r \leftarrow \nu r$;
  else
    $\left(\boldsymbol{\lambda}^k, \epsilon^k, k\right) \leftarrow \left(\boldsymbol{\lambda}^{k+1},\ \epsilon^{k+1},\ k+1\right)$;
  end
end

Algorithm 22: ProxGaussSeidel The PROX Gauss-Seidel variant with an adaptive global r-Factor strategy and parametric friction cones. The product $\mathbf{M}^{-1}\mathbf{J}^T$ may be precomputed.

where $\beta \geq 0$ is an auxiliary scalar variable. We can now restate the model with the help of the complementary function,

$$\psi(\beta, -\eta(\mathbf{f})) = 0\,, \tag{314}$$

$$\psi(\mathbf{v}_{\hat{t}}^T\mathbf{v}_{\hat{t}}, \mathbf{w}^T\mathbf{w}) = 0\,, \tag{315}$$

where we now introduce $\mathbf{w} = \nabla\eta(\mathbf{f}) + \beta\,\mathbf{v}_{\hat{t}}$. The above model gives us a root finding problem and can be solved with a Newton type of method. For this purpose we must obtain the generalized Jacobian of these equations. The differential becomes

$$d\psi(\beta, -\eta(\mathbf{f})) = \partial_a\psi\, d\beta - \partial_b\psi\, \nabla\eta^T d\mathbf{f}\,, \tag{316}$$

$$d\psi(\mathbf{v}_{\hat{t}}^T\mathbf{v}_{\hat{t}}, \mathbf{w}^T\mathbf{w}) = 2\partial_a\psi\, \mathbf{v}_{\hat{t}}^T d\mathbf{v}_{\hat{t}} + 2\partial_b\psi\, \mathbf{w}^T d\mathbf{w}\,, \tag{317}$$

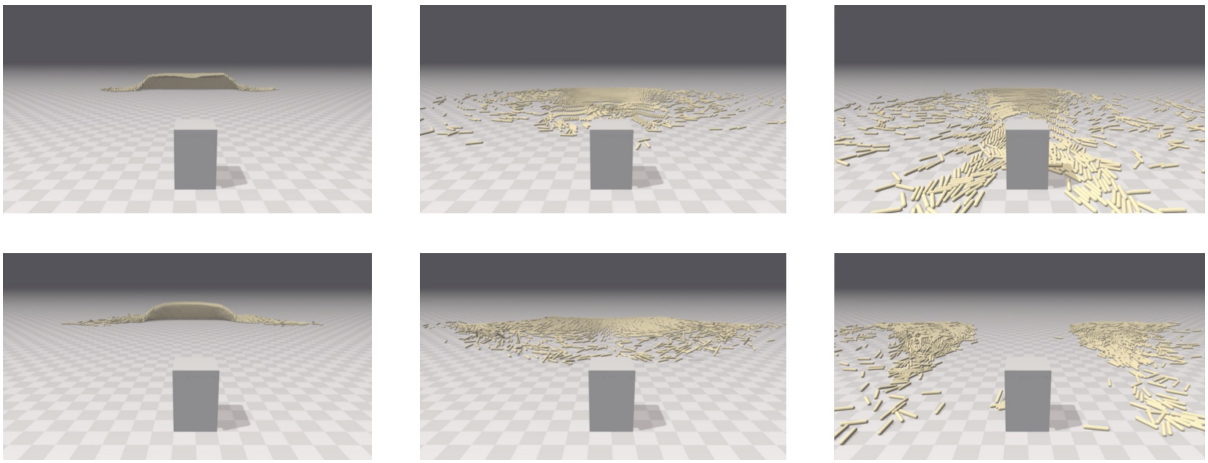


Fig. 37. A structure field on a ravine slope can be used to steer a log slide towards (top) or around (bottom) a cube-shaped building.

where

$$d\mathbf{w} = \mathbf{v}_{\hat{t}}\, d\beta + \nabla^2\eta\, d\mathbf{f} + \beta\, d\mathbf{v}_{\hat{t}}\,. \tag{318}$$

Here we use ∇^2 to denotes the Hessian of η. Assembling all parts we can write

$$\begin{bmatrix} d\psi(\beta, -\eta(\mathbf{f})) \\ d\psi(\mathbf{v}_{\hat{t}}{}^T\mathbf{v}_{\hat{t}}, \mathbf{w}^T\mathbf{w}) \end{bmatrix} = \mathbf{J}_\psi \begin{bmatrix} d\beta \\ d\mathbf{f} \\ d\mathbf{v}_{\hat{t}} \end{bmatrix} \tag{319}$$

where $\mathbf{J}_\psi$ is the Jacobian one will need for implementing a Newton method, and is computed as

$$\mathbf{J}_\psi \equiv \begin{bmatrix} \partial_a\psi & -\partial_b\psi\,\nabla\eta^T & 0 \\ 2\left(\partial_b\psi\mathbf{w}^T\mathbf{v}_{\hat{t}}\right) & 2\left(\partial_b\psi\mathbf{w}^T\nabla^2\eta\right) & 2\left(\partial_a\psi\mathbf{v}_{\hat{t}}{}^T + \partial_b\psi\mathbf{w}^T\beta\right) \end{bmatrix}.$$

Figure 37 shows examples of Flex which uses a fully implicit solver based on a Newton method using a Fischer-Burmeister formulation.

6.1.6 Implementation Notes. Notice that a friction cone generator $\mathcal{G}$, such as the one in Algorithm 21, returns not only a friction cone $\mathcal{F}$, but also the cone orientation $\mathbf{C} \equiv \begin{bmatrix} \hat{n} & \hat{t} & \hat{b} \end{bmatrix}$. Here, $\mathbf{C}$ is needed for the assembly of the contact Jacobian, as is evident from Equation 22. Hence, one may wish to invoke the generator when assembling the contact Jacobian or split the generator implementation into two sub-routines– one for computing the cone and one for determining the contact frame. The choice is intimately related to how the time-discretization of the friction cone is implemented. Imagine that the cone generator parametrically depends on the sliding velocity $\mathbf{v}_{\hat{t}}$ and that a fully implicit integration scheme is wanted. In this case, the generator truly needs to be invoked each time before projecting $\boldsymbol{\lambda}_{\hat{t}}$ to the cone. However,

in most cases the positions and orientations are not updated inside the solver algorithm as shown in Algorithm 22, and both the cone orientation and limit surface can be computed outside the solver for improved computational efficiency. However, for the Newton type solver outlined above, one must update cone orientations continuously as positions and velocities are solved in a fully coupled way, which gives a fully implicit time-integration method.

We note that an additional torque can be included in the friction model, which introduces an angular moment about the contact normal based on an additional friction coefficient, μ_τ. The value of this coefficient is computed similar to $\mu_{\hat{t}}$ and $\mu_{\hat{b}}$, and this type of torsional friction is also known as a Coulomb–Contensou friction model. Inclusion of this torque is optional from a modeling perspective, but does result in a higher-dimensional limit surface, and thus requires a third diagonal term in $\mathbf{C}$ as $\frac{1}{\mu_\tau}^2$, thus promoting $\mathbf{R}$ to a 3D rotation matrix. The proximal operator for this surface can then be solved numerically, as outlined in Section 3.4.3, whereas omitting this additional term gives a planar surface and the projection can be solved analytically.

6.2 Friction Tensors

The friction tensor model was introduced to the graphics community by Pabst et al. [2009]. Material parameters of the model are described by one scalar and two tensors: $\kappa \in \mathbb{R}_+$ and $\mathbf{Q}_\mathrm{A}, \mathbf{Q}_\mathrm{B} \in \mathbb{R}^{2\times 2}$. This gives a total of 9 parameters for planar friction, which can be encoded as texture maps in a practical implementation. The model defines dynamic friction, $\mathbf{f}$, by a direct evaluation of

$$\mathbf{f} \equiv -\lambda_{\hat{n}}\,\kappa\,\left(\mathbf{Q}_\mathrm{A} + \mathbf{R}\mathbf{Q}_\mathrm{B}\mathbf{R}^T\right)\mathbf{v}_{\hat{t}}\,, \tag{320}$$

where $\mathbf{v}_{\hat{t}}$ is the unit-length tangential relative sliding vector, $\lambda_{\hat{n}}$ is the magnitude of the normal force. Observe that it is necessary to first align the tensor from the second surface B with the first surface A before combining them. This is achieved by a rotation around the contact normal $\hat{n}$, which aligns the tangential bases vectors $\hat{t}_\mathrm{A}$ and $\hat{t}_\mathrm{B}$ from each surface, such that

$$\begin{aligned}\theta &\equiv \operatorname{acos}\left(\hat{t}_\mathrm{A}\cdot\hat{t}_\mathrm{B}\right),\\ \mathbf{R} &\equiv \mathbf{R}(\hat{n},\phi)\,,\end{aligned}$$

and $\mathbf{R}(\hat{n},\phi)$ is the rotation around $\hat{n}$ with angle θ. The work of Pabst et al. [2009] provides many pointers on how to generate and setup these parameters in an implementation.

The friction tensor model is quite easy to add to solvers based on a "lagged" friction model where direction evaluation of the friction force is sufficient. In some cases, the model can be converted into a cone description. We therefore present the cone reformation in full detail, since this gives more insight into the nature of the model.

We define the linear operator $\mathbf{L}$ to derive the cone counterpart of the friction tensor model:

$$\begin{aligned}\mathbf{f} &= -\lambda_{\hat{n}}\, \underbrace{\kappa\, \left(\mathbf{Q}_{\mathrm{A}} + \mathbf{R}\mathbf{Q}_{\mathrm{B}}\mathbf{R}^T\right)}_{\equiv \mathbf{L}} \mathbf{v}_{\hat{t}} \\ &= -\lambda_{\hat{n}}\, \mathbf{L}\, \mathbf{v}_{\hat{t}}\end{aligned}$$

Assuming that $\mathbf{L}$ is non-singular, we may now write:

$$\mathbf{f} = -\lambda_{\hat{n}}\, \mathbf{L}\, \mathbf{v}_{\hat{t}} \tag{321}$$

$$\Rightarrow \mathbf{L}^{-1}\mathbf{f} = -\lambda_{\hat{n}}\, \mathbf{v}_{\hat{t}} \tag{322}$$

$$\Rightarrow \left\|\mathbf{L}^{-1}\mathbf{f}\right\|^2 = \lambda_{\hat{n}}^2 \tag{323}$$

$$\Rightarrow \mathbf{f}^T \underbrace{\left(\mathbf{L}^{-T}\mathbf{L}^{-1}\right)}_{\equiv \mathbf{C}_P} \mathbf{f} = \lambda_{\hat{n}}^2 \tag{324}$$

Observe that this is the same as an elliptical-cone Coulomb friction model we used in Section 6.1.2, and an eigenvalue decomposition will recover the usual cone description

$$\mathbf{C}_p \rightarrow \mathbf{R}^T \begin{bmatrix} \frac{1}{\mu_{\hat{t}}^2} & 0 \\ 0 & \frac{1}{\mu_{\hat{b}}^2} \end{bmatrix} \mathbf{R}\,. \tag{325}$$

Next, let us investigate the dissipation of the model. Writing the instantaneous power,

$$\mathbf{v}_{\hat{t}} \cdot \mathbf{f} = -\lambda_{\hat{n}} \frac{\mathbf{v}_{\hat{t}}{}^T \mathbf{L} \mathbf{v}_{\hat{t}}}{\|\mathbf{v}_{\hat{t}}\|}\,, \tag{326}$$

we observe that if $\mathbf{Q}_{\mathrm{A}}$ and $\mathbf{Q}_{\mathrm{B}}$ are symmetric positive definite, then there are sufficient conditions to state that $\mathbf{L}$ will always be symmetric and positive definite. In this case, the friction force will always be dissipative.

Recall from the principle of maximum dissipation that the negative sliding velocity must be in the normal cone of the friction cone, or $-\mathbf{v}_{\hat{t}} \in \mathcal{N}_{\mathbf{C}_P}(\mathbf{f})$, if $\mathbf{f}$ is the maximal dissipative force (see Section 1.5.1). We may now use an implicit function to express the friction cone version of the friction tensor model:

$$\psi(\mathbf{f}) \equiv \mathbf{f}^T \left(\mathbf{L}^{-T}\mathbf{L}^{-1}\right) \mathbf{f} - \lambda_{\hat{n}}^2\,. \tag{327}$$

To prove that $\mathbf{f}$ is maximal dissipating, there must exist some value $\beta > 0$ such that

$$-\beta\, \mathbf{v}_{\hat{t}} = \nabla_{\mathbf{f}}\, \psi(\mathbf{f})\,. \tag{328}$$

It is straightforward to find an expression for $\nabla_{\mathbf{f}}\,\psi(\mathbf{f})$:

$$d_{\mathbf{f}}\,\psi = d\mathbf{f}^T\left(\mathbf{L}^{-T}\mathbf{L}^{-1}\right)\mathbf{f} + \mathbf{f}^T\left(\mathbf{L}^{-T}\mathbf{L}^{-1}\right)d\mathbf{f} \tag{329}$$

$$= \underbrace{2\mathbf{f}^T\left(\mathbf{L}^{-T}\mathbf{L}^{-1}\right)}_{\equiv(\nabla_{\mathbf{f}}\,\psi)^T} d\mathbf{f}\,. \tag{330}$$

Hence,

$$-\beta\,\mathbf{v}_{\hat{t}} = 2\left(\mathbf{L}^{-T}\mathbf{L}^{-1}\right)\mathbf{f}\,. \tag{331}$$

Finally, redefining $\beta \leftarrow \frac{\beta}{2\lambda_{\hat{n}}}$ and recalling that $\mathbf{f} = -\lambda_{\hat{n}}\mathbf{L}\mathbf{v}_{\hat{t}}$, we have:

$$\beta\,\mathbf{v}_{\hat{t}} = \mathbf{L}^{-T}\,\mathbf{v}_{\hat{t}} \tag{332}$$

This shows that the friction tensor model is only maximal dissipating when the direction of sliding is an eigenvector of $\mathbf{L}^{-T}$. This also shows that there can be single cases (two for planar sliding) where the model is maximally dissipative, but in general $\mathbf{v}_{\hat{t}}$ can have any direction. Thus there are infinitely many cases where the principle of maximum dissipation will not hold.

In summary, the friction tensor model cannot be directly plugged into an existing simulator based on the concept of cones as it requires $\mathbf{L}$ to be non-singular and needs the computation of $\mathbf{C}_p$ and its eigenvalue decomposition. Further, when $\mathbf{L}$ is symmetric positive definite the model is always dissipating, but not necessarily maximally dissipating. Hence, the friction tensor model is best suited for approaches using a direct evaluation of the friction force, and in these cases it provides a very high level of expressiveness due to using 9 parameters to parameterize the friction behavior at a single point of contact. Pabst et al. [2009] show that friction tensors can be extended to support asymmetric behaviors too by adding a selection function to choose from different underlying tensor parameters based on the current sliding direction.

6.3 Texture Friction

Like the Matchstick model, the texture friction model proposed by Andrews et al. [2021] computes the limit surface on-the-fly using a surface encoding of geometric details. However, rather than assuming there are specific structural directions aligned with the material, a normal map texture is used to evaluate the friction cone for surface patches from each colliding surface. These are the same textures used to render visual appearance of roughness in many computer graphics applications, and so there is consistency between the frictional behavior and the visual appearance of the objects.

The normal textures encode the facet orientations on the surface of simulation objects due to meso-scale asperities. When asperities, from one surface collide with another surface, there is elastic deformation, fracture, and even plowing [Sheng Chen and Liu 2016]. This last phenomenon contributes to friction by asperities from one surface plowing into the another surface. This is where the texture-based friction model begins.

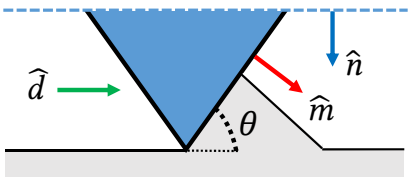


Fig. 38. A surface asperity shown in blue has meso-facet direction $\hat{m}$ and mean surface direction $\hat{n}$. It is being pushed in the direction $\hat{d}$, and thus plows into the opposing surface creating friction.

6.3.1 The Plowing Model. Friction due plowing is based on the angle of a meso-facet θ, such that

$$\mu = \frac{2}{\pi}\tan\theta, \tag{333}$$

and $\theta = \text{acos}(\hat{n} \cdot \hat{m})$ is the angle between the mean surface normal $\hat{n}$ and the meso-facet normal $\hat{m}$. The principal concepts of the plowing model are illustrated in Figure 38.

From Equation 333, it is clear that the plowing model is capable of generating an isotropic limit surface based on the orientation of a meso-facet, since we have already seen how this is done from a single friction coefficient in earlier chapters. However, consider for a moment the analogy that a meso-facet behaves like a plow blade that grinds against another surface. It is easy to conceive that the resistance to motion will increase if the plowing direction is aligned with $\hat{m}$. The friction can therefore be limited by considering the direction of sliding in the tangent plane of the contact surface, or $\hat{d}$. The modified version of the friction model can be written as:

$$\mu_{\hat{d}} = \left(\hat{m}' \cdot \hat{d}\right) \frac{2}{\pi}\,\theta\,, \tag{334}$$

Here, $\hat{m}'$ is the facet orientation $\hat{m}$ projected on the tangent space of the contact plane. This version of the model is now capable of exhibiting anisotropy, since μ depends on the sliding direction.

The model is further extended by noting that the meso-facet may be oriented in the direction opposite plowing. For instance, consider that the asperity shown in Figure 38 begins moving in the opposite direction, in which case no friction should be generated by the facet on the right-hand side of the asperity. This one-sided behavior is achieved by the max function:

$$\mu_{\hat{d}} = \max\left(0, \hat{m}' \cdot \hat{d}\right) \frac{2}{\pi}\,\theta \tag{335}$$

This version of the model is now anisotropic and asymmetric. Since there are two surfaces A and B colliding, directional friction coefficients for both can be computed and combined, such

that

$$\mu_{\hat{d}} = \underbrace{\max\left(0, \hat{m}'_{\mathrm{A}} \cdot \hat{d}\right) \frac{2}{\pi} \theta_{\mathrm{A}}}_{\mu_{\hat{d},\mathrm{A}}} + \underbrace{\max\left(0, \hat{m}'_{\mathrm{B}} \cdot \hat{d}\right) \frac{2}{\pi} \theta_{\mathrm{B}}}_{\mu_{\hat{d},\mathrm{B}}}, \tag{336}$$

where θ_{A},$\hat{m}'_{\mathrm{A}}$ and θ_{B},$\hat{m}'_{\mathrm{B}}$ are coincident features from surfaces A and B, respectively.

6.3.2 Texture Friction for Surface Patches. Next, we consider how this model can used to approximate the limit surface for small regions of colliding surfaces. Since the meso-facet orientations are encoded on surfaces using normal map textures, an efficient way to evaluate the model at many different points on the surface is by using a GPU-based rendering pipeline. This also has the benefit that efficient routines for clipping and culling surface geometry are automatically leveraged.

A patch surrounding each contact point is rendered using a special fragment shader, and the model in Equation 335 is evaluated for each pixel. The results are rendered into framebuffers I_{A} and I_{B}, one for each surface. This is done by constructing a small viewing volume around the contact point that is aligned with the contact frame. The pixels in each framebuffer store the directional friction coefficients, and computing the aggregate friction coefficients for each patch involves integrating the coefficients over the patch. The authors Andrews et al. [2021] propose averaging the coefficients, such that

$$\mu_{\hat{d}_k} = \frac{1}{|\mathcal{P}|} \sum I_{\mathrm{A}(i,j)\in\mathcal{P}} + \frac{1}{|\mathcal{P}|} \sum I_{\mathrm{B}(i,j)\in\mathcal{P}},$$

where $\mathcal{P}$ is set of all pixel coordinates (i, j) and coordinate k selects a direction. This implies that a discrete number of plowing directions may be rendered using this pipeline.

For example, to generate a limit surface that is compatible with a BLCP formulation, we can compute directional friction coefficients in the four sliding directions aligned with the tangent and binormal directions $\{-\hat{t}, -\hat{b}, \hat{t}, \hat{b}\}$. This results in directional friction coefficients $\{\mu_{-\hat{t}}, \mu_{-\hat{b}}, \mu_{\hat{t}}, \mu_{\hat{b}}\}$. Figure 39 shows normal textures from two surface patches rendering using the process described above, and the resulting directional coefficient images. This approach is not limited to just four plowing directions. It is trivial to use a larger number of tangent directions and evaluate the model for each of these directions, for instance to compute coefficients for a linear polyhedral cone presented in Section 1.6.

Having computed the coefficients $\{\mu_{-\hat{t}}, \mu_{-\hat{b}}, \mu_{\hat{t}}, \mu_{\hat{b}}\}$, it is now trivial to include them in a BLCP formulation and solve for frictional forces using a fixed-point iterative solver. For instance, the projection presented in Section 3.2.2 is modified with friction bounds computed using the directional coefficients:

$$\boldsymbol{\lambda}_{\hat{t}} \leftarrow \min\left(\mu_{\hat{t}}\boldsymbol{\lambda}_{\hat{n}},\ \max\left((\mu_{-\hat{t}}\boldsymbol{\lambda}_{\hat{n}}),\ \boldsymbol{\lambda}_{\hat{t}}\right)\right) \tag{337}$$

$$\boldsymbol{\lambda}_{\hat{b}} \leftarrow \min\left(\mu_{\hat{b}}\boldsymbol{\lambda}_{\hat{n}},\ \max\left((\mu_{-\hat{b}}\boldsymbol{\lambda}_{\hat{n}}),\ \boldsymbol{\lambda}_{\hat{b}}\right)\right) \tag{338}$$

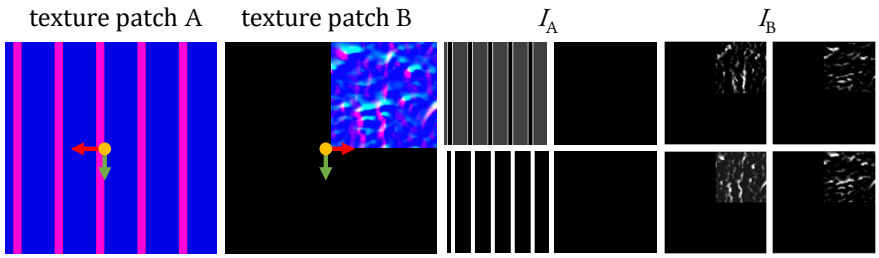


Fig. 39. Patches rendered from surfaces with a sawtooth texture (A) and bumpy texture (B). The resulting coefficient images for four axis-aligned plowing directions are also shown for both surfaces. Note that only part of surfaces overlap, and hence only the upper-left corner of each coefficient image will used to compute the aggregate directional coefficients.

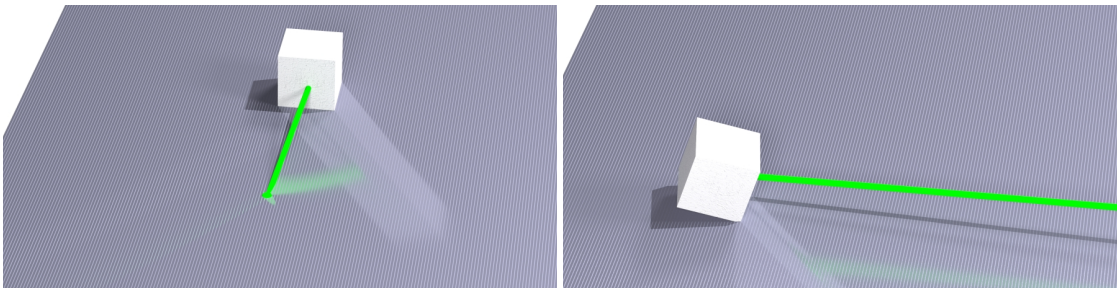


Fig. 40. A cube sliding on a textured plane. When the cube is pulled in a direction aligned with the ridges of a sawtooth texture, it slides easily (left). The same cube is pulled against the ridges and large friction forces are generated, causing it to stick.

6.4 Other Anisotropic Approaches

Anisotropic Coulomb friction is supported by many physics engines [CM Labs Simulations 2017; Coumans 2005; Todorov et al. 2012]. An appealing characteristic of the models presented in this chapter is that material parameters can be stored separately for each object, and then either the friction forces or a limit surface are computed on-the-fly by combining local information from each colliding surface. However, more pragmatic approaches than the ones presented in this chapter are often used. A popular choice is to use a material-pair look-up table storing one or more Coulomb coefficients. Alternatively, friction coefficients are assigned to each object, but then averaged when objects collide to compute aggregate coefficients.

Another challenging aspect when dealing with anisotropic friction models is that a common material frame must be specified in order to compute the direction dependent friction forces. Many physics engines use the current sliding direction to determine the first principal axis of the material frame. Another common approach is to assign the material frame from the local coordinate frame of one of the colliding objects.

Table 5. Overview of common dynamic friction models used in computer graphics. ANISO denotes aniosotropic, PMD denotes principle of maximum dissipation, INT denotes intrinsically defined in the contact frame, $\lambda_{\hat{n}}$ is the normal force/impulse, and $\mathbf{v}$ is the relative contact velocity.

Model Name	Shape	# Params	Frame Orientation	Scaling	ANISO	PMD	INT
Isotropic Coulomb [Baraff 1989]	Circle	1	None	Linear in $\lambda_{\hat{n}}$	No	Yes	Yes(1)
Anisotropic Coulomb(2)	Ellipse	2	Fixed on one object or by sliding velocity	Linear in $\lambda_{\hat{n}}$	Yes	Yes	Yes(1)
Limit Surfaces [Goyal et al. 1989]	Any Cone	∞	Not specified	Linear in $\lambda_{\hat{n}}$	Yes	Yes(3)	Yes
Friction Tensors [Pabst et al. 2009]	Affine Map	9	Fixed to one of the structure fields	Linear in $\lambda_{\hat{n}}$	Yes	No(4)	No(5)
Cloth Friction [Chen et al. 2013]	Load force curve	> 6(6)	Fixed to material space	Nonlinear in $\lambda_{\hat{n}}$, quadratic in $\mathbf{v}$	No	Yes	Yes
Matchstick [Erleben et al. 2020]	Ellipse	3	Mean of both structure field directions	Linear in $\lambda_{\hat{n}}$	Yes	Yes	Yes
Texture Friction [Andrews et al. 2021]	Polyhedral	2	Replaced by sampling	Linear in $\lambda_{\hat{n}}$	Yes	Yes	No

(1) Many engines either average friction coefficients assigned to objects, or use a material-pair look-up table.
(2) Anisotropic Coulomb friction is used in many physics engines, where the sliding direction is used to determine the first major principal axis of the contact coordinate system. However, the sliding and least direction of friction are not always parallel in reality.
(3) Supports non-convex cones giving rise to non-uniqueness even when PMD is applied.
(4) PMD is fulfilled only under restriction of symmetric positive definite friction tensors.
(5) Two separate friction tensors combine to give the intrinsic friction map.
(6) Three nonlinear function are fit to data using linear regression. Hence, more than 6 parameters seems reasonable as otherwise the 3 nonlinear functions would be no better than a linear fitting.

Table 5 presents a brief overview and comparison of work on friction models used in the field of computer graphics. The different traits in the table can help guide the selection of a model for a given application.

A TUTORIAL: PROGRAMMING A RIGID BODY SIMULATOR WITH FRICTIONAL CONTACT

This appendix covers the main steps for developing a simple rigid body simulator with frictional contact. Specifically, this tutorial encompasses the following concepts:

- Semi-implicit integration of rigid bodies (Section 1.2)
- Collision detection between analytic shapes (Section 2.1)
- Building the contact Jacobian for boxed LCPs (Section 1.7)
- Solving boxed LCPs using the PGS method (Section 3.2)

Getting started. As a first step, visit the "Rigid Body Simulations with Frictional Contact" tutorial website:

https://github.com/siggraphcontact/rigidBodyTutorial

Follow the instructions in the README to build the project files and load the code using your favorite development environment. Compile and execute the `rigidBodyTutorial` project and you should see the simulation viewer appear on your screen as shown in Figure 41.

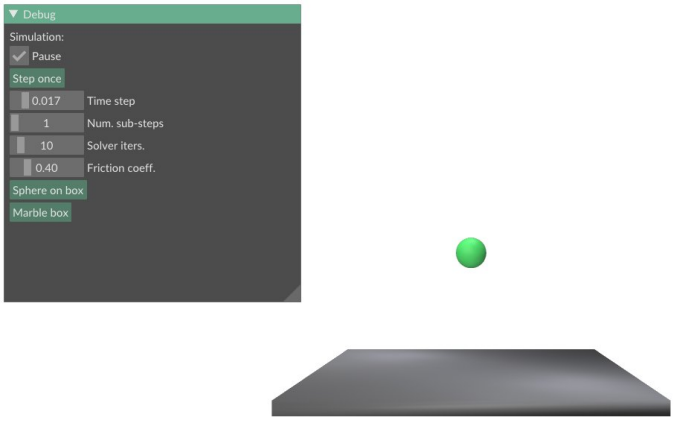


Fig. 41. The simulation viewer for the rigid body tutorial. You should see a similar view when you run the tutorial code for the first time.

The starter code is written in C++ and uses Eigen for linear algebra operations. Therefore, we transition to a less formal presentation of technical information that is closer to the code you will write for the implementation. Locations in the code where you should implement the objectives described below are annotated with a `//TODO` comment. They indicate where you should implement the objectives described below. Let's get started!

Step 1: Numerical integration of rigid bodies. The first objective in simulating contact is having bodies that move and collide. Therefore, for this step, you will update the positions and

orientations of bodies using a numerical integrator. Implement your changes in the function `step` in `RigidBodySystem.cpp`.

Begin by looping over each `RigidBody` in the member array `m_bodies` and use the semi-implicit time integration scheme described in Section 1.2 to update the positions and orientations of the rigid bodies. Compute the updated velocities of each rigid body using the forces (`f`) and torques (`tau`). For instance, the code below is a realization of Equation 3 that includes gyroscopic forces:

```
body->xdot  += dt * (1/body->m) * body->f
body->omega += dt * body->Iinv *
            (body->tau - body->omega.cross(I*body->omega))
```

The world space inertia and inverse inertia matrices, `I` and `Iinv` respectively, are already computed for you. ANote that bodies with the `fixed` flag set to true should not move, and their velocities must be set to zero.

The orientation of each body is stored as a quaternion `q`, which must be updated using a special kinematic mapping that maps angular velocities `omega` to a quaternion differential. The time rate of change can be computed as:

```
qdot = 0.5 * H * b->omega
```

Here, `H` is 4×3 matrix defined as follows:

$$\mathtt{H} = \begin{bmatrix} \mathtt{-q.x()} & \mathtt{-q.y()} & \mathtt{-q.z()} \\ \mathtt{q.w()} & \mathtt{q.z()} & \mathtt{-q.y()} \\ \mathtt{-q.z()} & \mathtt{q.w()} & \mathtt{q.x()} \\ \mathtt{q.y()} & \mathtt{-q.x()} & \mathtt{q.w()} \end{bmatrix}$$

Then, an update to the orientation can be computed as Δ `q = h* qdot`. Take care to renormalize each quaternion, since they may not be unit length after the integration update. The code to compute the gravitational force is already provided for you. It may be also a good idea to re-read Section 1.10.1 when completing this step.

If updates to the rigid bodies are computed correctly, objects in the scene should begin to fall when you start the simulation.

Step 2: Sphere-OBB Collision Detection. The sphere shown in the test scene (Figure 41) should eventually fall and collide with the static box underneath. Implement the code for a sphere-sphere collision test in the function `collisionDetectSphereBox`, which can be found in the file `collision/CollisionDetect.cpp`. The code for sphere-sphere collision detection is provided in `collisionDetectSphereSphere` as a guide for your implementation.

Use the process outlined in Section 2.1.2. First, transform the sphere into the local coordinate system of the box. Then, perform tests between the local sphere center and each face of the box to determine if the sphere is colliding with the box, and if so, which face of the box.

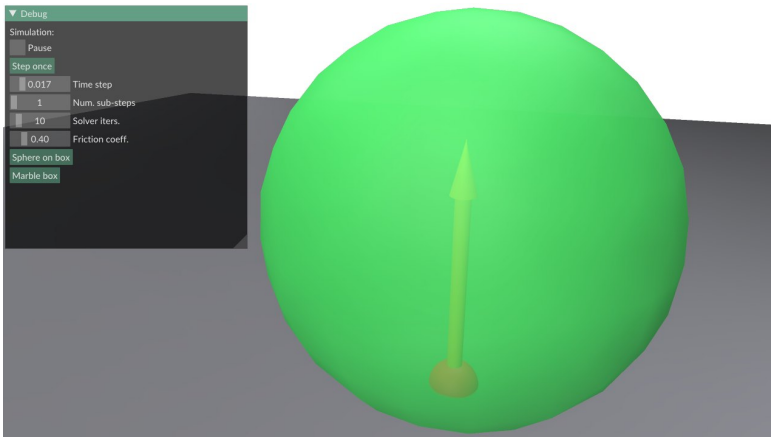


Fig. 42. The collision point and contact normal are visualized by the viewer. This is the result when your Sphere-OBB collision detection is working.

If the two shapes are colliding, the function should create an instance of `Contact` and set the collision point, the normal, and the penetration depth in the constructor. The normal vector and collision point should be specified relative to the global coordinate frame, and so additional transformations are required to map the local normal and collision point to world space.

Add the new contact to the member array `m_contacts`, which is later used to compute the contact Jacobian.

Step 3: Compute the contact Jacobian. As we saw in Chapter 1, there are several ways to create the contact Jacobian for point-based contact, which dependings on the friction model. However, for this exercise, let's stick with one of the simplest models to implement– the boxed LCP. This requires computing a Jacobian with three rows.

First, using the information contained in the Contact class, compute the contact frame that includes the normal `n` and the two tangent directions, `t` and `b`. The normal has already been computed during the collision detection phase. However, the tangent directions are computed to form an orthonormal bases, such that $\mathtt{n} \perp \mathtt{t} \perp \mathtt{b}$. There are many valid bases here, but it is convenient to choose directions that are principally aligned with the global coordinate frame. For example:

```
t = n.cross( Vector3f(1, 0, 0) )
```

The above cross product will not work if `n` and `Vector3f(1,0,0)` are co-linear, in which case another axis aligned direction can be used. Recall that your implementation should ensure that `t` and `b` are unit length.

Next, compute the constraint Jacobian based on Equation 43. Use the function `computeJacobian` for your implementation. The Jacobian is stored in two 3×6 matrices– `JA` and `JB`– that correspond to the first and second body, respectively. Since both the contact point `p` and the

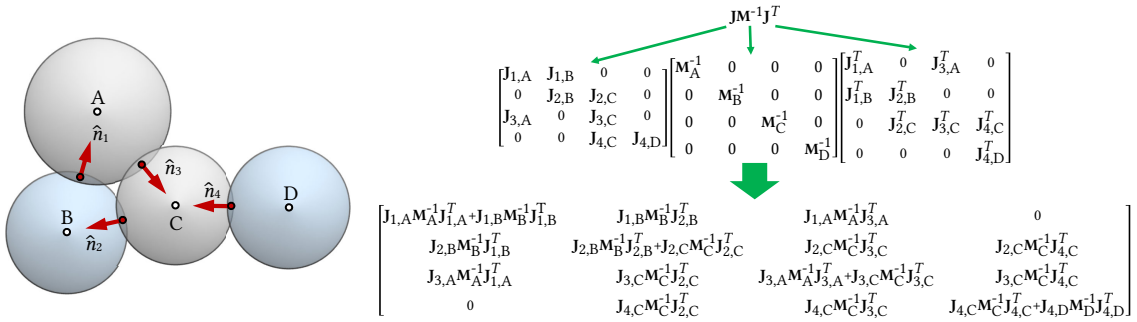


Fig. 43. A rigid body system with four contacts is shown. The system matrix is formed from the Delassus operator involving the global constraint Jacobian $\mathbf{J}$ and mass matrix $\mathbf{M}$. Upon inspection, there is a clear block structure that may be exploited for a matrix-free implementation of the PGS method.

body positions are represented in global coordinate space, the contact vector arm for the first and second body are conveniently computed by:

```
rA = p - bodyA->x
rB = p - bodyB->x
```

Once the vectors of the contact frame and vectors `rA` and `rB` have been computed, assemble `JA` and `JB` according to Equation 43.

Step 4: Iterative BLCP solver. For this last objective, you will implement the projected Gauss-Seidel method in `solvers/SolverBoxPGS.h`. Specifically, the blocked PGS method (see Algorithm 8). However, rather than assembling the system matrix $\mathbf{A}$, you will instead implement a matrix-free version of the algorithm that directly uses the Jacobian matrices computed in the previous step, as well as the inverse mass matrix of each body, to compute the terms for each PGS update.

Observe from the example shown in Figure 43 that there are $K = 4$ contacts and $N = 4$ bodies. The lead matrix is the product of three matrices– the global Jacobian matrix, the inverse mass matrix, and the transposed Jacobian– or $\mathbf{A} = \mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T$. However, upon further inspection, it is easy to see that there is a pattern to how this matrix is built.

Diagonal blocks are specific to a contact, and block (i, i) corresponds to the i-th contact with bodies p and q. It is computed as the sum of matrices $\mathbf{J}_{i,p}\mathbf{M}_p^{-1}\mathbf{J}_{i,p}^T + \mathbf{J}_{i,q}\mathbf{M}_q^{-1}\mathbf{J}_{i,q}^T$. The values of non-diagonal blocks with coordinates (i, j) are non-zero only if the i-th and j-th contact share a common body, in which case (i, j) contains $\mathbf{J}_{i,p}\mathbf{M}_p^{-1}\mathbf{J}_{j,p}^T$. . Otherwise, the non-diagonal block is zero.

We can take advantage of this structure for the PGS algorithm. Writing the update for variables of the i-th contact, $\mathbf{x}_i$:

$$\mathbf{x}_i = \underbrace{\left(\mathbf{J}_{i,p}\mathbf{M}_p^{-1}\mathbf{J}_{i,p}^T + \mathbf{J}_{i,q}\mathbf{M}_q^{-1}\mathbf{J}_{i,q}^T\right)}_{\texttt{diagA[i]}}{}^{-1} \left(\mathbf{b}_i - \sum_{j \neq i} \left(\underbrace{\mathbf{J}_{i,p}\mathbf{M}_p^{-1}\mathbf{J}_{j,p}^T}_{\text{coupling body } p} + \underbrace{\mathbf{J}_{i,q}\mathbf{M}_q^{-1}\mathbf{J}_{j,q}^T}_{\text{coupling body } q} \right) \mathbf{x}_j \right)$$

The term `diagA[i]` forms a 3×3 matrix, and implies that a small linear system is solved for the block update of each $\mathbf{x}_i$. The solution is then projected to the bounds defined by the boxed friction model and stored in the vector of constraint impulses `lambda`.

Also note that the terms $\mathbf{J}_{i,p}\mathbf{M}_p^{-1}\mathbf{J}_{j,p}^T$ and $\mathbf{J}_{i,q}\mathbf{M}_q^{-1}\mathbf{J}_{j,q}^T$ couple contacts i and j through bodies p and q respectively. However, there are many instances where these terms are will be zero if all contacts are considered. Therefore, rather than looping over all contacts, a more efficient way to evaluate the sum on the right-hand side is to visit the neighboring contacts (e.g., that have a common body with contact i). A list of contact constraints involving each body is maintained by the `RigidBody` class to make this task easier).

Below, a sketch of the main steps of the matrix-free block PGS solver is provided:

```
for (int iter = 0; iter < maxIter; ++iter) {
    for (int i = 0; i < numContacts; ++i) {
        // Initialize x = rhs vector.
        Eigen::VectorXf x = b[i];

        // Accumulate contributions from other contacts
        // coupled with bodyA
        accumulateCoupledContacts(contacts[i],
            contacts[i]->JAMinv,
            contacts[i]->bodyA, x);

        // Accumulate contributions from other contacts
        // coupled with bodyB
        accumulateCoupledContacts(contacts[i],
            contacts[i]->JBMinv,
            contacts[i]->bodyB, x);

        // Solve the 3x3 system and stored the result in lambda.
        //   diagA * contacts[i]->lambda = x
        //
```

```
            // Recall that for isotropic BLCPs:
            //       Normal impulse: 0 <= lambda[0] < infinity
            //       Friction impules:
            //       -mu*lambda[0] <= lambda[1] < mu*lambda[0]
            //       -mu*lambda[0] <= lambda[2] < mu*lambda[0]
            solveContact(diagA[i], x, contacts[i]->lambda,
                contacts[i]->mu);
        }
    }
```

The code above assumes that `diagA[i]` and `b[i]` have been constructed during an initialization step. The function `accumulateCoupledContacts()` loops over all other contacts that share the body and computes modifications to the right-hand side vector:

```
x -= (JMinv*Jother.transpose()) * lambda_other
```

The function `solveContact()` solves the block 3-by-3 system that represents the BLCP of a single contact. The function also projects the constraint impulses in `lambda` to the bounds specified by the contact model.

All done? Congratulations! You have implemented a rigid body simulator with frictional contact. There are many ways to extend the simulator. For instance, by using other numerical methods to solve the BLCP, or by adding support for collision handling between other shapes pairs, or using a non-linear friction cone. You are also encouraged to improve the performance of the simulator, particularly the PGS solver loop (e.g., by pre-computing the Jacobian "sandwich" transforms at the start of each time step, or using a gather-scatter strategy for computing the coupling impulses)

REFERENCES

V. Acary, F. Cadoux, C. Lemaréchal, and J. Malick. 2011. A formulation of the linear discrete Coulomb friction problem via convex optimization. *ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik* 91, 2 (2011), 155–175.

S. Ainsley, E. Vouga, E. Grinspun, and R. Tamstorf. 2012. Speculative parallel asynchronous contact mechanics. *ACM Trans. Graph.* 31, 6, Article 151 (Nov. 2012), 8 pages. https://doi.org/10.1145/2366145.2366170

S. Andrews, K. Erleben, P. G. Kry, and M. Teichmann. 2017. Constraint reordering for iterative multi-body simulation with contact. In *ECCOMAS '17: Proc. of the Multibody Dynamics 2007 ECCOMAS Thematic Conference*. ECCOMAS, Prague, Czech Republic.

S. Andrews, L. Nassif, K. Erleben, and P. G. Kry. 2021. Coupling Friction with Visual Appearance. *Proc. of the ACM on Computer Graphics and Interactive Techniques* 4, 3 (2021), 20. https://doi.org/10.1145/3480138

M. Anitescu and G. D. Hart. 2004. A constraint-stabilized time-stepping approach for rigid multibody dynamics with joints, contact and friction. *Intl. Journal for Numerical Methods in Engineering* 60, 14 (2004), 2335–2371. https://doi.org/10.1002/nme.1047

M. Anitescu and F. A. Potra. 1997. Formulating Dynamic Multi-Rigid-Body Contact Problems with Friction as Solvable Linear Complementarity Problems. *Nonlinear Dynamics* 14 (1997), 231–247. Issue 3. https://doi.org/10.1023/A:1008292328909

M. Anitescu and A. Tasora. 2010. An iterative approach for cone complementarity problems for nonsmooth dynamics. *Computational Optimization and Applications* 47, 2 (October 2010), 207–235. https://doi.org/10.1007/s10589-008-9223-4

L. Armijo. 1966. Minimization of functions having Lipschitz continuous first partial derivatives. *Pacific J. Math.* 16, 1 (1966), 1 – 3.

D. Baraff. 1989. Analytical Methods for Dynamic Simulation of Non-Penetrating Rigid Bodies. In *Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '89)*. Association for Computing Machinery, New York, NY, USA, 223–232. https://doi.org/10.1145/74333.74356

D. Baraff. 1993. Issues in computing contact forces for nonpenetrating rigid bodies. *Algorithmica. An Intl. Journal in Computer Science* 10, 2-4 (1993), 292–352. https://doi.org/10.1007/BF01891843

D. Baraff. 1994. Fast contact force computation for nonpenetrating rigid bodies. In *Proc. of the 21st Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '94)*. ACM, New York, NY, USA, 23–34. https://doi.org/10.1145/192161.192168

D. Baraff and A. Witkin. 1998. Large Steps in Cloth Simulation. In *Proc. of the 25th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '98)*. ACM, New York, NY, USA, 43–54. https://doi.org/10.1145/280814.280821

A. W. Bargteil, T. Shinar, and P. G. Kry. 2020. An Introduction to Physics-Based Animation. In *SIGGRAPH Asia 2020 Courses* (Virtual Event) *(SA '20)*. ACM, New York, NY, USA, Article 5, 57 pages. https://doi.org/10.1145/3415263.3419147

C. Batty, F. Bertails, and R. Bridson. 2007. A fast variational framework for accurate solid-fluid coupling. *ACM Trans. Graph.* 26, Article 100 (July 2007). Issue 3. https://doi.org/10.1145/1276377.1276502

J. Baumgarte. 1972. Stabilization of constraints and integrals of motion in dynamical systems. *Computer Methods in Applied Mechanics and Engineering* 1, 1 (1972), 1–16. https://doi.org/10.1016/0045-7825(72)90018-7

J. Bender, K. Erleben, and J. Trinkle. 2014. Interactive Simulation of Rigid Body Dynamics in Computer Graphics. *Comp. Graph. Forum* 33, 1 (2014), 246–270.

S. Bouaziz, S. Martin, T. Liu, L. Kavan, and M. Pauly. 2014. Projective Dynamics: Fusing Constraint Projections for Fast Simulation. *ACM Trans. Graph.* 33, 4 (July 2014), 154:1–154:11.

S. Boyd and L. Vandenberghe. 2004. *Convex Optimization*. Cambridge University Press, Cambridge. https://doi.org/10.1017/CBO9780511804441

R. Bridson, R. Fedkiw, and J. Anderson. 2002. Robust Treatment of Collisions, Contact and Friction for Cloth Animation. *ACM Trans. Graph.* 21, 3 (July 2002), 594–603. https://doi.org/10.1145/566654.566623

T. Brochu, E. Edwards, and R. Bridson. 2012. Efficient Geometrically Exact Continuous Collision Detection. *ACM Trans. Graph.* 31, 4, Article 96 (July 2012), 7 pages.

D. T. Chen and D. Zeltzer. 1992. Pump It up: Computer Animation of a Biomechanically Based Model of Muscle Using the Finite Element Method. In *Proc. of the 19th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '92)*. ACM, New York, NY, USA, 89–98. https://doi.org/10.1145/133994.134016

Y. Chen, M. Li, L. Lan, H. Su, Y. Yang, and C. Jiang. 2022. A Unified Newton Barrier Method for Multibody Dynamics. *ACM Trans. Graph. (SIGGRAPH)* 41, 4, Article 66 (2022).

Z. Chen, R. Feng, and H. Wang. 2013. Modeling Friction and Air Effects Between Cloth and Deformable Bodies. *ACM Trans. Graph.* 32, 4, Article 88 (July 2013), 8 pages.

M. B. Cline and D. K. Pai. 2003. Post-stabilization for rigid body simulation with contact and constraints. In *2003 IEEE Intl. Conference on Robotics and Automation*, Vol. 3. IEEE, Taipei, Taiwan, 3744–3751. https://doi.org/10.1109/ROBOT.2003.1242171

CM Labs Simulations. 2017. *Theory Guide: Vortex Software's Multibody Dynamics Engine.* Technical Report. https://www.cm-labs.com/vortexstudiodocumentation/Vortex_User_Documentation/Content/Concepts/theoryguide.html

R. Cottle. 1968. Complementary Pivot Theory of Mathematical Programming. *Linear Algebra and Its Applications* 1 (1968), 103–125. https://doi.org/10.1016/0024-3795(68)90052-9

R. Cottle, J.-S. Pang, and R. E. Stone. 1992. *The Linear Complementarity Problem.* Academic Press, Boston.

E. Coumans. 2005. The Bullet Physics Library. http://www.pybullet.org.

H. Courtecuisse, J. Allard, C. Duriez, and S. Cotin. 2011. Preconditioner-Based Contact Response and Application to Cataract Surgery. In *Proceedings of the 14th International Conference on Medical Image Computing and Computer-Assisted Intervention - Volume Part I* (Toronto, Canada) *(MICCAI'11)*. Springer-Verlag, Berlin, Heidelberg, 315–322.

G. Daviet. 2020. Simple and Scalable Frictional Contacts for Thin Nodal Objects. *ACM Trans. Graph.* 39, 4, Article 61 (July 2020), 16 pages. https://doi.org/10.1145/3386569.3392439

G. Daviet, F. Bertails-Descoubes, and L. Boissieux. 2011. A hybrid iterative solver for robustly capturing coulomb friction in hair dynamics. *ACM Trans. Graph.* 30, 6, Article 139 (Dec. 2011), 12 pages. https://doi.org/10.1145/2070781.2024173

A. Enzenhoefer, N. Lefebvre, and S. Andrews. 2019. Efficient Block Pivoting for Multibody Simulations with Contact. In *Proc. of the 2019 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (Montreal, Quebec, Canada) *(I3D '19)*. ACM, New York, NY, USA, Article 2, 9 pages. https://doi.org/10.1145/3306131.3317019

K. Erleben. 2005. *Stable, Robust, and Versatile Multibody Dynamics Animation.* Ph.D. Dissertation. Department of Computer Science, University of Copenhagen (DIKU).

K. Erleben. 2007. Velocity-Based Shock Propagation for Multibody Dynamics Animation. *ACM Trans. Graph.* 26, 2 (June 2007), 12–es. https://doi.org/10.1145/1243980.1243986

K. Erleben. 2017. Rigid Body Contact Problems Using Proximal Operators. In *Proc. of the 2017 ACM SIGGRAPH / Eurographics Symposium on Computer Animation* (Los Angeles, California) *(SCA '17)*. ACM, New York, NY, USA, Article 13, 12 pages. https://doi.org/10.1145/3099564.3099575

K. Erleben, M. Andersen, N. S., and S. M. 2011. num4lcp. https://github.com/erleben/num4lcp.

K. Erleben, M. Macklin, S. Andrews, and P. G. Kry. 2020. The Matchstick Model for Anisotropic Friction Cones. *Comp. Graph. Forum* 39, 1 (2020), 450–461. https://doi.org/10.1111/cgf.13885

Z. Ferguson et al. 2020. *IPC Toolkit.* https://ipc-sim.github.io/ipc-toolkit/

Z. Ferguson, M. Li, T. Schneider, F. Gil-Ureta, T. Langlois, C. Jiang, D. Zorin, D. M. Kaufman, and D. Panozzo. 2021. Intersection-Free Rigid Body Dynamics. *ACM Trans. Graph. (SIGGRAPH)* 40, 4, Article 183 (jul 2021), 16 pages.

M. C. Ferris and T. S. Munson. 1999. Interfaces to PATH 3.0: Design, Implementation and Usage. *Comput. Optim. Appl.* 12 (January 1999), 207–227. Issue 1-3. https://doi.org/10.1023/A:1008636318275

A. Fischer. 1992. A special newton-type optimization method. *Optimization* 24, 3-4 (1992), 269–284. https://doi.org/10.1080/02331939208843795

M. Foerg, T. Geier, L. Neumann, and H. Ulbrich. 2006. r-Factor Strategies for the Augmented Lagrangian Approach in Multi-Body Contact Mechanics. In *III European Conference on Computational Mechanics*. Springer Netherlands, Dordrecht, NL, 316. https://doi.org/10.1007/1-4020-5370-3_316

M. Fratarcangeli and F. Pellacini. 2015. Scalable Partitioning for Parallel Position Based Dynamics. *Comput. Graph. Forum* 34, 2 (May 2015), 405–413. https://doi.org/10.1111/cgf.12570

S. F. Frisken, R. N. Perry, A. P. Rockwood, and T. R. Jones. 2000. Adaptively Sampled Distance Fields: A General Representation of Shape for Computer Graphics. In *Proc. of the 27th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '00)*. ACM Press/Addison-Wesley Publishing Co., USA, 249–254. https://doi.org/10.1145/344779.344899

M. Geilinger, D. Hahn, J. Zehnder, M. Bächer, B. Thomaszewski, and S. Coros. 2020. ADD: Analytically Differentiable Dynamics for Multi-Body Systems with Frictional Contact. *ACM Trans. Graph.* 39, 6, Article 190 (Nov. 2020), 15 pages. https://doi.org/10.1145/3414685.3417766

E. G. Gilbert, D. W. Johnson, and S. S. Keerthi. 1988. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal on Robotics and Automation* 4, 2 (1988), 193–203. https://doi.org/10.1109/56.2083

H. Goldstein, C. Poole, and J. Safko. 2002. *Classical mechanics*. Addison-Wesley, USA. 638 pages.

S. Goyal, A. Ruina, and J. Papadopoulos. 1989. Limit Surface and Moment Funktion Descriptions of Planar Sliding. In *Proc. of the 1989 IEEE Intl. Conference on Robotics and Automation (Vol. 2)*. IEEE, Scottsdale, AZ, 794–799.

D. Harmon, E. Vouga, B. Smith, R. Tamstorf, and E. Grinspun. 2009. Asynchronous Contact Mechanics. *ACM Trans. Graph.* 28, 3, Article 87, 12 pages. https://doi.org/10.1145/1531326.1531393

D. Harmon, E. Vouga, R. Tamstorf, and E. Grinspun. 2008. Robust Treatment of Simultaneous Collisions. *ACM Trans. Graph.* 27, 3 (aug 2008), 1–4. https://doi.org/10.1145/1360612.1360622

S. Hasegawa, N. Fujii, Y. Koike, and M. Sato. 2003. Real-Time Rigid Body Simulation Based on Volumetric Penalty Method. In *HAPTICS '03: Proc. of the 11th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*. IEEE Computer Society, Los Alamitos, CA, USA, 326. https://doi.org/10.1109/HAPTIC.2003.1191304

J. Jansson and J. S. M. Vergeest. 2002. A discrete mechanics model for deformable bodies. *Computer-Aided Design* 34, 12 (2002), 913–928.

J. Jansson and J. S. M. Vergeest. 2003. Combining Deformable- and Rigid-Body Mechanics Simulation. *Vis. Comput.* 19, 5 (Aug. 2003), 280–290. https://doi.org/10.1007/s00371-002-0187-6

M. Jean. 1999. The non-smooth contact dynamics method. *Computer Methods in Applied Mechanics and Engineering* 177, 3–4 (July 1999), 235–257. https://doi.org/10.1016/S0045-7825(98)00383-1

M. W. Jones, J. A. Baerentzen, and M. Sramek. 2006. 3D distance fields: A survey of techniques and applications. *IEEE Transactions on visualization and Computer Graphics* 12, 4 (2006), 581–599.

F. Jourdan, P. Alart, and M. Jean. 1998. A Gauss-Seidel like algorithm to solve frictional contact problems. *Computer Methods in Applied Mechanics and Engineering* 155, 1 (1998), 31 – 47.

J. J. Júdice and F. M. Pires. 1994. A block principal pivoting algorithm for large-scale strictly monotone linear complementarity problems. *Computers & operations research* 21, 5 (1994), 587–596.

C. Kane, E. A. Repetto, M. Ortiz, and J. E. Marsden. 1999. Finite element analysis of nonsmooth contact. *CMAME* 180, 1-2 (1999).

D. M. Kaufman, S. Sueda, D. L. James, and D. K. Pai. 2008. Staggered Projections for Frictional Contact in Multibody Systems. *ACM Trans. Graph.* 27, 5, Article Article 164 (Dec. 2008), 11 pages. https://doi.org/10.1145/1409060.1409117

T. Kim and D. Eberle. 2020. Dynamic Deformables: Implementation and Production Practicalities. In *ACM SIGGRAPH 2020 Courses* (Virtual Event, USA) *(SIGGRAPH '20)*. ACM, New York, NY, USA, Article 23, 182 pages. https://doi.org/10.1145/3388769.3407490

Y. J. Kim, M. A. Otaduy, M. C. Lin, and D. Manocha. 2002. Fast Penetration Depth Computation for Physically-Based Animation. In *Proc. of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (San Antonio, Texas) *(SCA '02)*. ACM, New York, NY, USA, 23–31. https://doi.org/10.1145/545261.545266

D. Koschier, C. Deul, and J. Bender. 2016. Hierarchical Hp-Adaptive Signed Distance Fields. In *Proc. of the 2016 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Zurich, Switzerland) *(SCA '16)*. Eurographics Association, Goslar, DEU, 189–198.

C. Lacoursiere and M. Linde. 2011. *Spook: a variational time-stepping scheme for rigid multibody systems subject to dry frictional contacts.* Technical Report. HPC2N and Department of Computer Science, Umeaa University, Sweeden.

L. Lan, D. M. Kaufman, M. Li, C. Jiang, and Y. Yang. 2022. Affine Body Dynamics: Fast, Stable & Intersection-free Simulation of Stiff Materials. *ACM Trans. Graph. (SIGGRAPH)* 41, 4, Article 67 (2022).

L. Lan, Y. Yang, D. Kaufman, J. Yao, M. Li, and C. Jiang. 2021. Medial IPC: Accelerated Incremental Potential Contact with Medial Elastics. *ACM Trans. Graph. (SIGGRAPH)* 40, 4, Article 158 (July 2021), 16 pages.

R. I. Leine and C. Glocker. 2003. A set-valued force law for spatial coulomb-contensou friction. *European Journal of Mechanics - A/Solids* 22, 2 (2003), 193–216. https://doi.org/10.1016/S0997-7538(03)00025-1

M. Li, Z. Ferguson, T. Schneider, T. Langlois, D. Zorin, D. Panozzo, C. Jiang, and D. M. Kaufman. 2020. Incremental Potential Contact: Intersection- and Inversion-free Large Deformation Dynamics. *ACM Trans. Graph. (SIGGRAPH)* 39, 4 (2020).

M. Li, D. M. Kaufman, and C. Jiang. 2021. Codimensional Incremental Potential Contact. *ACM Trans. Graph. (SIGGRAPH)* 40, 4, Article 170 (July 2021), 24 pages.

X. Li, Y. Fang, M. Li, and C. Jiang. 2022a. BFEMP: Interpenetration-free MPM–FEM coupling with barrier contact. *Computer Methods in Applied Mechanics and Engineering* 390 (2022), 114350.

X. Li, M. Li, and C. Jiang. 2022b. Energetically Consistent Inelasticity for Optimization Time Integration. *ACM Trans. Graph. (SIGGRAPH)* 41, 4, Article 52 (2022).

J. Lloyd. 2005. Fast Implementation of Lemke's Algorithm for Rigid Body Contact Simulation. In *ICRA '05: Proc. of the 2005 IEEE Intl. Conference on Robotics and Automation*. IEEE, Barcelona, Spain, 4538–4543. https://doi.org/10.1109/ROBOT.2005.1570819

P. Lötstedt. 1984. Numerical Simulation of Time-Dependent Contact and Friction Problems in Rigid Body Mechanics. *SIAM journal on scientific and statistical computing* 5, 2 (1984), 370–393.

M. Macklin, K. Erleben, M. Müller, N. Chentanez, S. Jeschke, and Z. Corse. 2020a. Local Optimization for Robust Signed Distance Field Collision. *Proc. of the ACM on Computer Graphics and Interactive Techniques* 3, 1 (2020), 1–17.

M. Macklin, K. Erleben, M. Müller, N. Chentanez, S. Jeschke, and T. Y. Kim. 2020b. Primal/Dual Descent Methods for Dynamics. In *Proc. of the 2020 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Virtual Event, Canada) *(SCA '20)*. Eurographics Association, Goslar, DEU, Article 9, 12 pages. https://doi.org/10.1111/cgf.14104

M. Macklin, K. Erleben, M. Müller, N. Chentanez, S. Jeschke, and V. Makoviychuk. 2019. Non-Smooth Newton Methods for Deformable Multi-Body Dynamics. *ACM Trans. Graph.* 38, 5, Article Article 140 (Oct. 2019), 20 pages. https://doi.org/10.1145/3338695

H. Mazhar, T. Heyn, D. Negrut, and A. Tasora. 2015. Using Nesterov's Method to Accelerate Multibody Dynamics with Friction and Contact. *ACM Trans. Graph.* 34, 3, Article 32 (May 2015), 14 pages. https://doi.org/10.1145/

2735627

M. McKenna and D. Zeltzer. 1990. Dynamic simulation of autonomous legged locomotion. In *Proc. of the 17th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '90)*. ACM, Dallas, TX, USA, 29–38. https://doi.org/10.1145/97879.97882

X. Merlhiot. 2007. A robust, efficient and time-stepping compatible collision detection method for non-smooth contact between rigid bodies of arbitrary shape. In *ECCOMAS '07: Proc. of the 2007 Multibody Dynamics ECCOMAS Thematic Conference*. ECCOMAS, Milano, Italy, 20.

M. Moore and J. Wilhelms. 1988. Collision detection and response for computer animationr3. In *Proc. of the 15th annual conference on Computer graphics and interactive techniques (SIGGRAPH '88)*. ACM, New York, NY, USA, 289–298. https://doi.org/10.1145/54852.378528

J. J. Moreau. 1999. Numerical aspects of the sweeping process. *Computer Methods in Applied Mechanics and Engineering* 177, 3–4 (July 1999), 329–349.

M. Müller, N. Chentanez, T.-Y. Kim, and M. Macklin. 2015. Air Meshes for Robust Collision Handling. *ACM Trans. Graph. (SIGGRAPH)* 34, 4, Article 133 (July 2015).

T. S. Munson, F. Facchinei, M. C. Ferris, A. Fischer, and C. Kanzow. 2001. The Semismooth Algorithm for Large Scale Complementarity Problems. *INFORMS Journal on Computing* 13, 4 (2001), 294–311. https://doi.org/10.1287/ijoc.13.4.294.9734

K. G. Murty. 1974. Note on a Bard-type scheme for solving the complementarity problem. *Opsearch* 11, 2-3 (1974), 123–130.

K. G. Murty and F.-T. Yu. 1988. *Linear Complementarity, Linear and Nonlinear Programming*. Vol. 3. Helderman-Verlag, Berlin, Germany. 629 pages.

R. Narain, M. Overby, and G. E. Brown. 2016. ADMM ⊇ Projective Dynamics: Fast Simulation of General Constitutive Models. In *Proc. of the 2016 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Zurich, Switzerland) *(SCA '16)*. Eurographics Association, 21–28.

X. Ni, L. V. Kale, and R. Tamstorf. 2015. Scalable Asynchronous Contact Mechanics Using Charm++. In *2015 IEEE International Parallel and Distributed Processing Symposium*. 677–686.

S. M. Niebe. 2014. *Rigid Bodies in Contact: and Everything in Between.* Ph.D. Dissertation. Department of Computer Science, Faculty of Science, University of Copenhagen.

J. Nocedal and S. J. Wright. 2006. *Numerical optimization.* Springer-Verlag, New York. 664 pages. https://doi.org/10.1007/978-0-387-40065-5

M. Otaduy, R. Tamstorf, D. Steinemann, and M. Gross. 2009a. Implicit Contact Handling for Deformable Objects. *Comp. Graph. Forum* 28 (04 2009).

M. A. Otaduy, R. Tamstorf, D. Steinemann, and M. Gross. 2009b. Implicit Contact Handling for Deformable Objects. *Comp. Graph. Forum (Proc. of Eurographics)* 28, 2 (apr 2009). http://www.gmrv.es/Publications/2009/OTSG09

S. Pabst, B. Thomaszewski, and W. Straßer. 2009. Anisotropic Friction for Deformable Surfaces and Solids. In *Proc. of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (New Orleans, Louisiana) *(SCA '09)*. ACM, New York, NY, USA, 149–154. https://doi.org/10.1145/1599470.1599490

N. Parikh and S. Boyd. 2014. Proximal Algorithms. *Found. Trends Optim.* 1, 3 (Jan. 2014), 127–239.

A. Peiret, S. Andrews, J. Kovecses, P. G. Kry, and M. Teichmann. 2019. Schur Complement-based Substructuring of Stiff Multibody Systems with Contact. *ACM Trans. Graph.* 38, 5, Article 150 (2019), 17 pages. https://doi.org/10.1145/3355621

M. Poulsen, S. Niebe, and K. Erleben. 2010. Heuristic Convergence Rate Improvements of the Projected Gauss-Seidel Method for Frictional Contact Problems. In *WSCG '10: In Proc. of the 18th Intl. Conf. in Central Europe on Computer Graphics, Visualization and Computer Vision*. University of West Bohemia, Plzen, Czech Republic, 135–142.

S. Redon, A. Kheddar, and S. Coquillart. 2002. Fast Continuous Collision Detection between Rigid Bodies. *Computer Graphics Forum* 21 (May 2002).

T. Schneider, J. Dumas, X. Gao, D. Zorin, and D. Panozzo. 2019. *Polyfem*.

G. Sheng Chen and X. Liu. 2016. Chapter 3 - Friction. In *Friction Dynamics*, Gang Sheng Chen and Xiandong Liu (Eds.). Woodhead Publishing, Cambridge, UK, 91–159. https://doi.org/10.1016/B978-0-08-100285-8.00003-1

E. Sifakis and J. Barbic. 2012. FEM Simulation of 3D Deformable Solids: A Practitioner's Guide to Theory, Discretization and Model Reduction. In *ACM SIGGRAPH 2012 Courses* (Los Angeles, California) *(SIGGRAPH '12)*. ACM, New York, NY, USA, Article 20, 50 pages. https://doi.org/10.1145/2343483.2343501

M. Silcowitz, S. Niebe, and K. Erleben. 2009. Nonsmooth Newton Method for Fischer Function Reformulation of Contact Force Problems for Interactive Rigid Body Simulation. In *Proc. of the 6th Workshop on Virtual Reality Interaction and Physical Simulation* (Karlsruhe, DE) *(VRIPHYS '09)*. The Eurographics Association, Karlsruhe, DE, 105–114. https://doi.org/10.2312/PE/vriphys/vriphys09/105-114

M. Silcowitz, S. Niebe, and K. Erleben. 2010a. A nonsmooth nonlinear conjugate gradient method for interactive contact force problems. *The Visual Computer* 26, 6 (2010), 893–901. https://doi.org/10.1007/s00371-010-0502-6

M. Silcowitz, S. Niebe, and K. Erleben. 2010b. Projected Gauss-Seidel Subspace Minimization Method for Interactive Rigid Body Dynamics. In *VISIGRAPP '10: Proc. of the 5th Intl. Conference on Computer Graphics Theory and Applications*. Springer Berlin Heidelberg, Angers, France, 218–229. https://doi.org/10.1007/978-3-642-25382-9_15

M. Silcowitz-Hansen. 2010. Jinngine: a Physics Engine Written In Java. https://github.com/rzel/jinngine

J. M. Snyder. 1992. Interval Analysis for Computer Graphics. In *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '92)*. Association for Computing Machinery, New York, NY, USA, 121–130.

D. E. Stewart and J. C. Trinkle. 1996. An Implicit Time-Stepping Scheme for Rigid Body Dynamics with Inelastic Collisions and Coulomb Friction. *Intl. Journal of Numerical Methods in Engineering* 39, 15 (1996), 2673–2691.

C. W. Studer. 2008. *Augmented Time-stepping Integration of Non-smooth Dynamical Systems*. Ph.D. Dissertation. ETH Zurich. https://doi.org/10.3929/ethz-a-005556821

I. E. Sutherland and G. W. Hodgman. 1974. Reentrant polygon clipping. *Commun. ACM* 17, 1 (1974), 32–42. https://dl.acm.org/doi/10.1145/360767.360802

M. Tang, R. Tong, Z. Wang, and D. Manocha. 2014. Fast and Exact Continuous Collision Detection with Bernstein Sign Classification. *ACM Trans. Graph.* 33 (Nov. 2014), 186:1–186:8. Issue 6.

A. Tasora, D. Mangoni, S. Benatti, and R. Garziera. 2021. Solving variational inequalities and cone complementarity problems in nonsmooth dynamics using the alternating direction method of multipliers. *Intl. Journal for Numerical Methods in Engineering* 122, 16 (2021). https://doi.org/10.1002/nme.6693

J. Teran, E. Sifakis, G. Irving, and R. Fedkiw. 2005. Robust Quasistatic Finite Elements and Flesh Simulation. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Los Angeles, California) *(SCA '05)*. Association for Computing Machinery, New York, NY, USA, 181–190.

D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer. 1987. Elastically Deformable Models. In *Proc. of the 14th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '87)*. Association for Computing Machinery, New York, NY, USA, 205–214. https://doi.org/10.1145/37401.37427

E. Todorov, T. Erez, and Y. Tassa. 2012. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 5026–5033. https://doi.org/10.1109/IROS.2012.6386109

G. Van Den Bergen. 2003. *Collision detection in interactive 3D environments*. CRC Press, San Francisco, USA. 277 pages.

M. Verschoor and A. C. Jalba. 2019. Efficient and accurate collision response for elastically deformable models. *ACM Trans. Graph.* 38, 2 (2019), 1–20.

B. Wang, Z. Ferguson, X. Jiang, M. Attene, D. Panozzo, and T. Schneider. 2022. Fast and Exact Root Parity for Continuous Collision Detection. *Computer Graphics Forum (Proceedings of Eurographics)* 41, 2 (2022), 9.

B. Wang, Z. Ferguson, T. Schneider, X. Jiang, M. Attene, and D. Panozzo. 2021. A Large-Scale Benchmark and an Inclusion-Based Algorithm for Continuous Collision Detection. *ACM Trans. Graph.* 40, 5, Article 188 (sep 2021), 16 pages.

H. Wang. 2015. A Chebyshev Semi-iterative Approach for Accelerating Projective and Position-based Dynamics. *ACM Trans. Graph.* 34, 6 (Oct. 2015), 246:1–246:9.

H. Xu and J. Barbič. 2014. Signed distance fields for polygon soup meshes. In *Proc. of the 40th Graphics Interface Conference (GI '14)*. Canadian Information Processing Society, Montreal, Canada, 35–41. https://dl.acm.org/doi/pdf/10.5555/2619648.2619655

H. Xu, Y. Zhao, and J. Barbič. 2014. Implicit Multibody Penalty-based Distributed Contact. *IEEE Transactions on Visualization and Computer Graphics* 20, 9 (Sep. 2014), 1266–1279. https://doi.org/10.1109/TVCG.2014.2312013

Y. Zhao, J. Choo, Y. Jiang, M. Li, C. Jiang, and K. Soga. 2022. A barrier method for frictional contact on embedded interfaces. *Computer Methods in Applied Mechanics and Engineering* 393 (2022), 114820.