A CONTACT MODEL FOR GEOMETRICALLY ACCURATE TREATMENT OF POLYTOPES IN SIMULATION

By

Jedediyah Freeman Williams

A Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: COMPUTER SCIENCE

Approved by the
Examining Committee:

_____
Jeffrey C. Trinkle, Thesis Adviser

_____
Srinivas Akella, Member

_____
Kurt S. Anderson, Member

_____
Barbara Cutler, Member

_____
Charles V. Stewart, Member

Rensselaer Polytechnic Institute
Troy, New York

August 2014
(For Graduation December 2014)

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENT

Thanks foremost to my advisor Jeff Trinkle, who has given me so many amazing opportunities and whose guidance has been invaluable. Thanks also to my doctoral committee, whose feedback and insight have been essential to the development of my work. Thanks in particular to Chuck and Barb, whose classes I had the great fortune to be able to take. I learned a great deal from them both, and had a most excellent time doing it. Many thanks and much appreciation also goes to the staff here in RPI CS, especially Pam, who in addition to being exceedingly helpful is exceedingly kind.

Thanks to my loving family for being totally rad and indefatigably supportive. Thanks to my second family, the Nilsons (and the subset Ripps), whose love and support mean the universe to me, and which I could not have done without.

To all of my lab mates, past and present, it has truly been a pleasure to share this research and learning experience with you. I will greatly miss working with this talented and capable group of people. I can't imagine I will ever again work in an environment where I can casually say "we should build a quad copter" and three weeks later we are tuning the flight controller to hover. Nor do I expect to hear a future coworker ask "wanna see the robot I built this weekend?"

Thanks to my amazing, brilliant, and beautiful dear friends. As David J. Buckley Esquire points out, it is remarkable that the same group of preschoolers should still be hanging out decades later.

ABSTRACT

Simulation can be an invaluable tool, particularly in fields such as robotics where physical experiments can be extremely expensive, time consuming, and even dangerous. However, the value of a simulator is directly related to its ability to accurately and reliably model physical phenomena such as intermittent contact. In virtually all multibody simulators available today, the standard methods of contact identification and response treat the free space between pairs of bodies as a convex set, when it is in fact non-convex. To reconcile, simulators typically use very small time steps and employ numerous *ad hoc* corrections, many of which have become commonplace and include allowing interpenetration to occur, arbitrarily limiting response forces, misrepresenting body geometries, and "freezing" bodies with relatively low velocities. Indeed, a vast body of literature exists addressing the many symptoms of dynamic instability due to poor contact determination.

I herein present a formulation of non-penetration constraints between pairs of bodies which accounts for all possible combinations of active contact. This is the first formulation that accurately models the body geometries near points of *potential* contact, simultaneously preventing interpenetration while allowing bodies to traverse accurately through the surrounding free space. Unlike the standard approach, this method does not need to guess at which contacts to enforce. This new formulation is easy to incorporate into existing simulation methods, improves accuracy by many orders of magnitude, and is stable for even large time steps.

Additionally, I present the RPI-MATLAB-Simulator (RPIsim), an open source tool for efficient research and practical teaching in multibody dynamics. RPIsim is designed to be easy to use and easily extended. Students being introduced to dynamics for the first time have no problem creating and running simulations, even with a limited programming background. Researchers can utilize the existing code base to support work on specific areas since it is easy to replace or extend individual modules of the simulator.

CHAPTER 1
INTRODUCTION

There is a wide spectrum of applications for simulation, but an understandable bias toward real-time interactivity has bred a culture that leaves concerns of physical accuracy in a dark corner. As a result, applications which require some level of physical fidelity from simulation are left with a relatively underdeveloped set of tools. The major focus of the work presented herein is on improving simulation accuracy, and subsequently stability, without increasing computational complexity.

This trade off of speed versus accuracy in simulation is a classic one, and many applications simply do not gain from improved physical accuracy. The gaming community is naturally concerned with methods that are fast and reliable, but not necessarily accurate [1–3], inspiring the term "game physics." Many games in fact intentionally incorporate non-realistic physics into their play, and no one would rely on Angry Birds, which uses Box2D [4], to predict avian flight patterns. The graphics community, which is constantly pushing the boundaries of realistic-looking visual effects, does not offer many improvements to physically accurate simulation. Although there are many examples in a long history of "physics based" methods in use in animation [5–9], there is really no need for such methods to do more than approximate some convenient alternative version of reality. Indeed, a more pressing concern in many graphics applications is to present the viewer with something that "feels" correct.

Particularly in the field of robotics, physical fidelity is a pressing concern and is especially challenging with regard to intermittent frictional contact. Robotics experiments tend to be exceedingly costly and time consuming, so that the ability to perform reliable experiments in simulation would be a great benefit. Although some work has been done regarding validation of simulation [10–13], the problem of measuring physical accuracy of simulation is a difficult one. Certainly we can say that convergence of a constraint solver is not a good metric alone, since the underlying constraints may be flawed. Comparison of a simulation with that of a

commercial simulator (an argument I have heard with unfortunate frequency) is meaningless without proof of the commercial simulator's accuracy. Even validation of a simulator with experiment can be misinterpreted when it takes the tuning of several simulation parameters to match a single experiment. Such an attempt at validation says little, if anything, of the simulator's ability to predict a different experiment.

Simulation is a powerfully useful tool if it can be shown to be accurate. A simulation requires accuracy in order to make any claims as to its ability to model behavior in the physical world, yet the growing number of variants on old simulation methods all have at least this in common: they sacrifice numerical and physical fidelity at multiple stages in order to achieve computational speed-up. The ideal simulation tool would be both accurate and fast, but when we sacrifice accuracy, we are necessarily sacrificing the usefulness of that tool.

1.1 Time-stepping

One category of simulation is the *event-driven* approach, which distinguishes between dynamic modes such as no contact, sticking contact, and slipping contact, and integrates over the current mode until a mode switching event occurs [14]. This may be preferable for small numbers of bodies and contacts, as it is continuous and computationally efficient in such a case, but becomes prohibitively expensive as the number of contacts increases [15] since the mode switching will occur more frequently.

Time-stepping is by far the most popular approach to simulating multiple bodies, and the most common representation of bodies are as polygons in 2D and polyhedra (often triangle meshes) in 3D. Throughout this document, I will be focusing on time-stepping methods which follow the pattern depicted in Figure 1.1. These methods involve maintaining the positions and velocities of a set of bodies and determining how to update these values over discrete time steps, starting from an initial valid configuration.

The first stage in Figure 1.1 of graphically rendering the scene, although not strictly necessary in the definition of this loop, is becoming more relevant as imple-

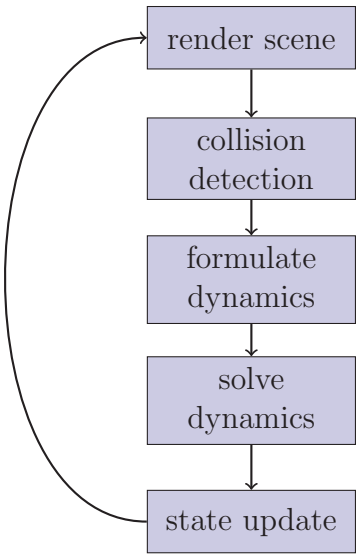


Figure 1.1: The main stages of the time-stepping simulation approach. At each iteration, a contact set is determined, a time-stepping problem representing the dynamic model is formulated, a solution to this problem is found which generates updated body properties, and finally these properties are used to update body positions.

mentations are being pushed onto graphics cards and graphics data structures are becoming more intimately tied with simulation, *e.g.,* CUDA's interopability. The next stage, collision detection, involves the identification of contact or "interference" points between pairs of bodies. From the set of contacts (and any additional constraints such as joints or force/torque control), the next stage formulates a time-stepping problem. This problem is solved in the next stage, providing an updated set of velocities for each body. In the final stage, the new velocities are integrated over the simulation time step to obtain the new body positions.

Of particular interest in this document are the stages of collision detection and dynamics formulation, and how they are connected. The big question is: during simulation, when given a pair of bodies composed of vertices, edges, and faces, that are relatively close such that they are or could be in contact within the next time step, what contacts should be enforced to best model their interaction with physical fidelity?

There are two major classes of methods for handling contact, *preventative*

methods [16–18], or *penalty*, methods [19–21]. Preventative methods attempt to identify contacts before they are violated, ideally preventing interpenetration of bodies. This approach has several benefits over penalty methods, particularly that it exhibits more stable behavior with regard to dynamics [22]. The major issue with preventative methods, which will be our starting off point of Chapter 4, is deciding which contacts to enforce when there are several to choose from. Enforcing too few contacts will increase the likelihood of interpenetration, and enforcing too many contacts will generate unnatural interactions.

Unlike preventative methods, penalty methods wait for contacts to be violated before enforcing them. Penalty methods are generally straight forward to implement and computationally less expensive than preventative methods, however they typically require very small time steps in order to avoid deep interpenetrations which result in dynamic instability. Traditional penalty methods model contact as a linear spring system [23], essentially pushing back on points of contact with penetration. As we shall see in Chapter 5, penalty methods can exhibit unpredictable behavior that is highly sensitive to time step size, and do not converge dynamically toward stability for small time steps.

1.2 State-of-the-art

At the time of this writing, the Defense Advanced Research Projects Agency (DARPA) Robotics Challenge (DRC) (`www.theroboticschallenge.org`) has completed its first two stages. The first stage was the Virtual Robotics Challenge (VRC) in which competitors implemented controllers and algorithms for enabling a virtual ATLAS robot to complete a set of predefined tasks meant to represent scenarios that could possibly occur at a disaster scene such as a nuclear meltdown. DRC tasks have included

1. Drive a utility vehicle
2. Traverse rubble
3. Remove debris blocking a doorway
4. Open a door and travel through it

5. Climb an industrial ladder

6. Pick up a tool and use it to cut through a wall

7. Locate and turn a set of valves

Winning teams of the VRC were given an ATLAS robot built by Boston Dynamics, with the idea that they would be able to port their algorithms from the VRC onto the real robot. Unfortunately, due to challenges with simulation, teams in the VRC were forced to exploit unrealistic modeling choices in the simulation software which did not translate into real robot control.

The simulator used for the VRC, and subsequently used by many teams during the DRC trials in December of 2013, was the Gazebo simulator [24], currently being developed by the Open Source Robotcs Foundation (OSRF). Figure 1.2 shows

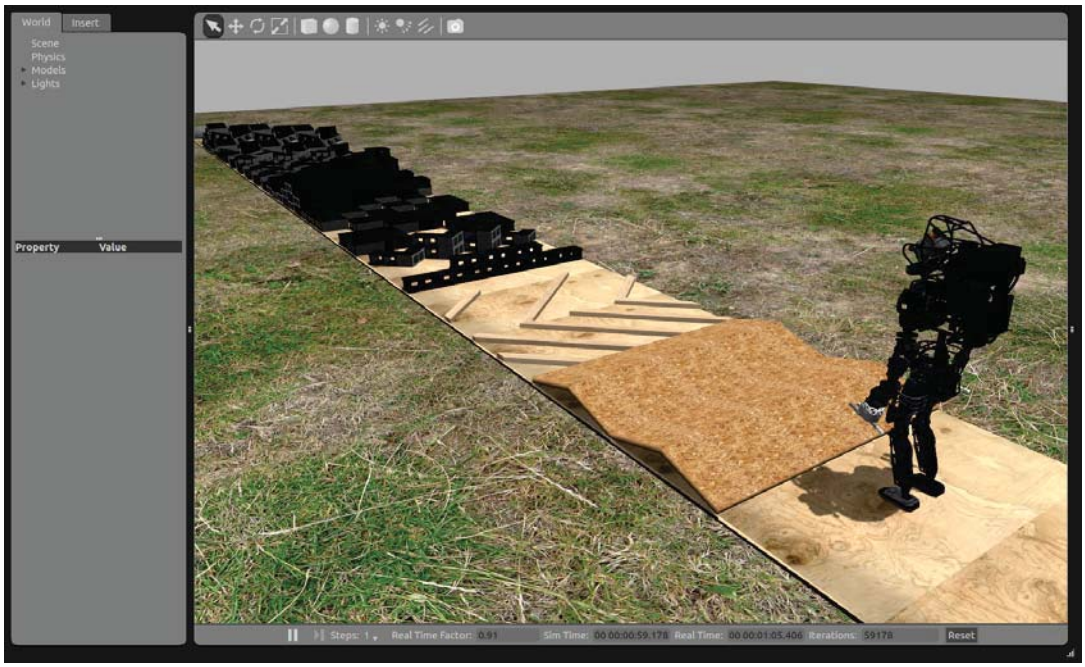


Figure 1.2: ATLAS standing at the start of the terrain task in the Gazebo Simulator.

ATLAS standing at the start of task 2, the terrain traversal task, as simulated by Gazebo.

The code segment in Figure 1.3 is from the file ODEPhysics.cc in the Gazebo simulator version 3.0.0, and includes a heuristic for limiting the number of contacts

between pairs of bodies. Prior to this point in the code, an initial set of contacts was determined by finding all contacts within a small distance ϵ between two bodies. Line 854 tests if the number of contacts in the initial set is over a given threshold. If

```
853   // Choose only the best contacts if too many were generated.
854   if (numc > maxCollide)
855   {
856     double max = _contactCollisions[maxCollide-1].depth;
857     for (unsigned int i = maxCollide; i < numc; ++i)
858     {
859       if (_contactCollisions[i].depth > max)
860       {
861         max = _contactCollisions[i].depth;
862         this->indices[maxCollide-1] = i;
863       }
864     }
865
866     // Make sure numc has the valid number of contacts.
867     numc = maxCollide;
868   }
```

Figure 1.3: Code segment from ODEPhysics.cc in the Gazebo 3.0.0. When the number of contacts between two bodies is over a certain threshold, only the contacts with the deepest violations are kept. This is a popular contact heuristic.

it is, then the subset of these contacts with deepest penetration are kept, while the other contacts are removed. This choice of heuristic, choosing to include contacts with deepest penetration, is a common one. It does seem quite sensible to think that correcting the deepest contact should take priority over less severe contacts. However, this simple heuristic can lead to instabilities.

Consider the case of a vertex of one body near a corner of another body as depicted in Figure 1.4. The bottom right vertex of body $\mathcal{A}$ has a small penetration with edge e_2 which could be corrected with a small force. The significant weakness of the deepest contact heuristic manifests when the vertex gets too close to e_1. As soon as this contact is considered, it will take priority since its penetration depth is significant. Subsequently, a non-physical and large force will be applied to "correct" this violation. It is tempting to immediately suggest an alternative heuristic to solve for this specific case, but such a heuristic would inevitably break for a different case.

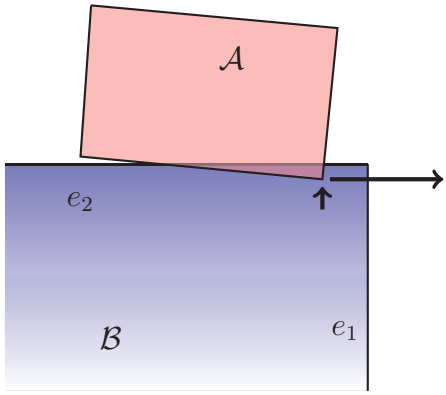
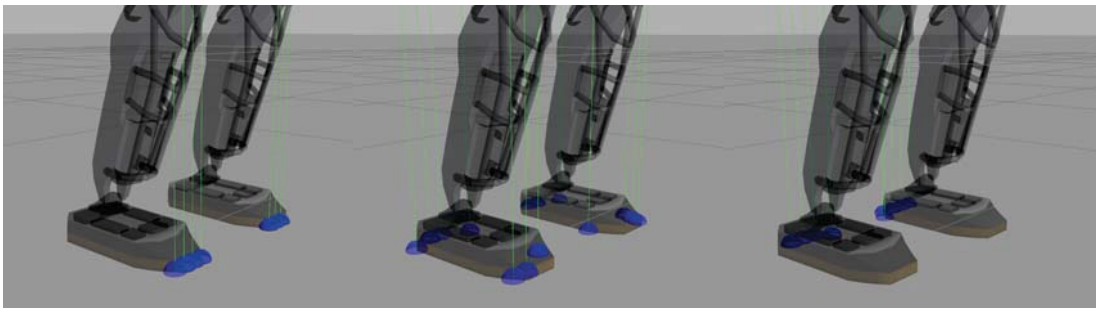


Figure 1.4: The bottom right vertex of body $\mathcal{A}$ has potential contact with two edges of $\mathcal{B}$. Including both contacts will generate a non-physical force sideways. The common heuristic of including contacts with deepest penetration guarantees this non-physical behavior in such cases.

Note in particular that it is entirely possible for the gap distance with e_1 to be smaller than the gap distance with e_2 in nearly the same configuration, but a sideways force would still be non-physical.

Figure 1.5 demonstrates another instability which is common in multibody simulators. In each frame, we see the set of contacts (blue spheres) and the direction of the contact normal (green lines) between the feet of ATLAS and the ground plane. In 1.5a, the collision detection in Gazebo has identified a set of contacts that has the



(a) Time $\sim$ 1.7 s **(b) Time $\sim$ 1.85 s** **(c) Time $\sim$ 2.0 s**

Figure 1.5: Visualization of contacts and contact normals between ATLAS's foot and the ground plane while "standing" in the Gazebo simulator. Contact instabilities have imposed additional challenges in the DARPA Robotics Challenge.

deepest penetration, and at this particular time step these contacts happen to all be

at the front of the feet. Given this contact set, and the center of mass of the robot being behind these contacts, ATLAS will rock backward. Less than half a second later we see that the heels now have the deepest penetration and the contacts at the toes are no longer included. As a result of choosing only a subset of the possible contacts, as well as using a heuristic of choosing contacts with deepest penetration, Gazebo and similar simulators tend to generate oscillatory motions of bodies that should stay dynamically still. A common approach to correcting this is to dampen body velocities and zero velocities below a certain threshold.

That state-of-the-art simulators employ such unsophisticated techniques for identifying contacts speaks to a general focus on speed over accuracy, and exemplifies the underdeveloped state of methods for contact determination. There are some heuristics that have shown improved stability, such as using grouping on normal directions [25] or maximizing the surface area between contacts. However, these heuristics are treating the symptoms and not the cause, and are susceptible to other instabilities.

Despite significant challenges lingering with regard to modeling multibody contact and dynamics, many have pushed past onto larger and more complex simulations. Unfortunately, simulations with large numbers of bodies result in a sort of overloading of human perception so that it is difficult for humans to perceive the large number of non-physical interactions, especially when these interactions may occur quickly over just a few time steps. This is unfortunate because it creates a sense of greater progress than there really is.

Since the Bullet physics engine [26] is used in many video games and for some special effects, it has little incentive to be physically accurate but great incentive to be fast. Yet, like the Open Dynamics Engine (ODE) [27] which is the default physics engine used in Gazebo, Bullet is utilized for robotics simulation. Figure 1.6 is a screen capture of the demo for Bullet 2.80 release, in which we can see many interpenetrations between bodies, even some bodies that are stuck inside others (front center). The video demonstrates the impressive speed of Bullet for many bodies, but also serves as an excellent example of sacrificing accuracy for speed.

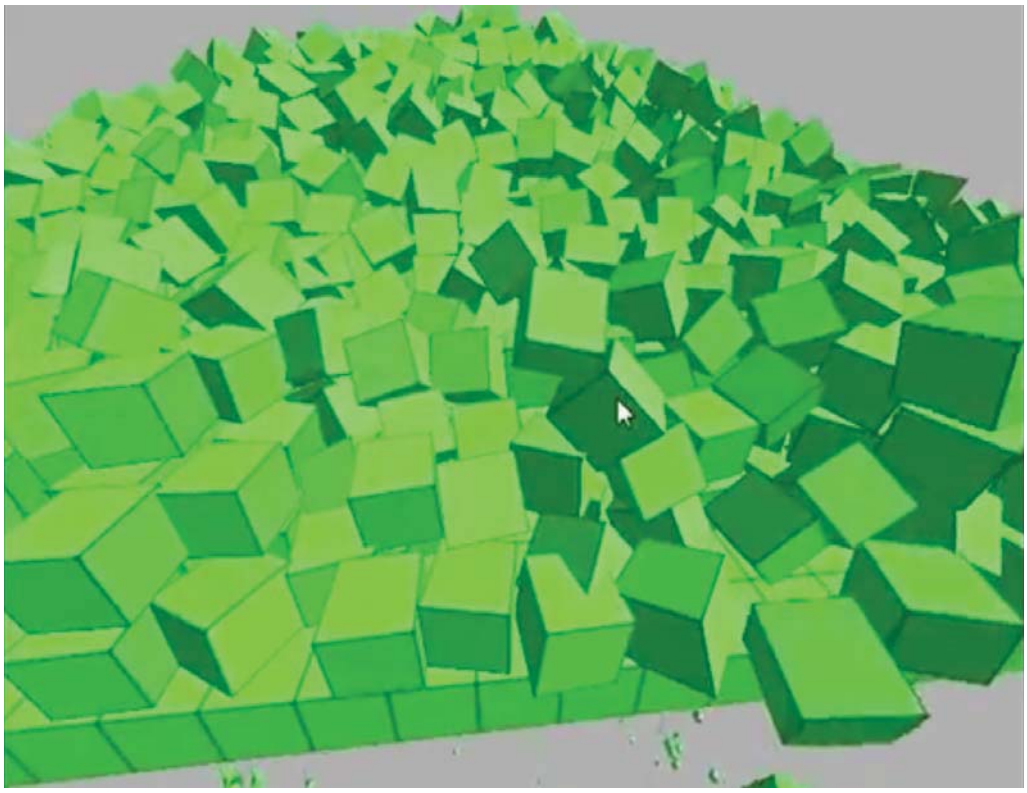


Figure 1.6: A screen capture of the Bullet 2.80 release video `www.youtube.com/watch?v=8jGZv1YYe2c` at 3:12. Despite all 110,000 bodies being identical (which makes collision detection more straight forward), this simulation has numerous dramatic and deep interpenetrations.

Figure 1.7 is a screen capture of a video from the Simulation Based Engineering Lab (SBEL) group at University of Wisconsin, Madison. This group has many impressive results not just for rigid bodies, but also for fluids and vehicles on granular terrain. Unfortunately, they too must deal with underdeveloped methods, particularly concerning collision detection and interference response for rigid body dynamics.

At this point we may make a general observation on demonstrations of state-of-the-art simulators: they tend to involve a large number of similar, if not identical, bodies. Although this may give an innocent enough impression in that it is easier to generate identical bodies as opposed to several thousand unique bodies, it drastically reduces the conclusions one can make from such a demonstration. In particular, giving every body the same set of attributes is an extremely special case and allows

Figure 1.7: A screen capture of video `vimeo.com/31810546` at 0:47. Although the vast number of bodies makes it difficult to discern, there are numerous deep interpenetrations between bodies.

for many assumptions to be made during simulation which would not be safe in general. Using identical bodies also avoids many issues with collision detection since precisely the same set of tests will be applied between every pair of potentially colliding bodies. Having identical body attributes also helps to avoid issues that would arise when trying to solve multibody dynamics, since having all masses being equal greatly improves the numerical properties of the dynamics problems, and subsequently improves the stability of solution methods.

There are several closed source and for-profit simulators, but they do not seem to show improvements in physical fidelity. There are several packages intended for game physics, such as Havok Physics [28] and PhysX [29]. There are also packages such as the widely popular Adams simulator [30], which purports to allow the user to simulate "Real World" physics, but we should make special note their use of quotation marks regarding what type of physics they simulate. Adams uses a penalty method, and consequently exhibits many of the issues and instabilities associated with such methods. Other simulation software packages, such as Cinema 4D Dynamics for MAXON, or Massive for Maya, are intended for use in cinematography

and need only provide eye-pleasing results.

In addition to robotic simulation, there are many other

Some of the challenges with state-of-the-art simulation include:

1. Penalty methods are popular for their ease of implementation and speed, but are inherently flawed (non-physical) and require fine tuning to achieve stability.

2. Many problems in collision detection are poorly defined, and simulators tend to require custom collision detection.

3. It is common in popular collision detection libraries to use some form of broad-phase collision detection, and then GJK (see 2.6) for narrow phase. The problem with using only GJK at narrow-phase is that it returns only a single contact. A single contact is not generally good enough to provide accuracy and stability.

4. Simulators scale poorly. Many contain hard coded parameters that are only appropriate for certain problems. Others deal well only with particular body types.

5. Solvers are slow (inefficient)

6. Solvers struggle with some numerical properties of simulation problems, such as large mass ratios and complex friction models.

7. Little has been done to analyze the performance of the various solvers commonly used for simulation.

8. The trade off of speed for accuracy results in non-physical simulation. In particular, accuracy is often sacrificed in friction and proper collision response.

9. Commonly, joint constraint correction involves applying an impulse *after* a general dynamics solution. These corrections ignore other constraints such as non-penetration and friction, voiding guarantees of accuracy.

1.3 Contributions

There are many applications for which simulation only becomes useful when it accurately represents real-world phenomena. The goal of my research has been to develop mathematical methods and simulation tools to push the state-of-the-art performance in multibody dynamics simulation toward accuracy and dynamic stability, with a particular focus on applications in robotics. The three major contributions presented herein are as follows.

1. **Contact determination**

Chapter 3 details a set of geometric tests for robust identification of potential contacts for polygons in 2D and polyhedra in 3D. While many simulators wait for interpenetration to occur before identifying contact (due in part to the nature of the collision detection tools available), it is necessary to detect potential contacts before they are violated in order to avoid interpenetration. In doing so, we are able to avoid the many instabilities that inevitably arise if we were to attempt to enforce constraints only after they have been violated.

2. **Modeling of contact interdependencies**

During simulation, it is possible to reach a state in which a feature of one body has multiple contacts with several features of another body. The traditional approach is to make some heuristic choices regarding which of these contacts to include in the "active" set. Unfortunately, it is always possible to provide examples which break this approach. In Chapter 4, I introduce by means of derivation a contact model which is capable of taking into account all potential contacts, and show how to formulate a time stepping problem which takes into account the interdependencies between them. This new contact model permits all geometrically accurate configurations of bodies in 2D and 3D. Chapter 5 compares the performance of this new model to popular and competitive alternative models.

3. **RPI-MATLAB-Simulator**

In Chapter 6, I describe the RPI-MATLAB-Simulator (RPIsim), an open source simulation framework for research and education in multibody dynamics.

RPIsim is designed and organized to be extended. Its modular design allows users to edit or add new components without worrying about extra implementation details. RPIsim has two main goals:

1. Provide an intuitive platform that is easy to extend for research and education in multibody dynamics,

2. Maintain an evolving code base of useful algorithms and analysis tools for multibody dynamics problems.

CHAPTER 2
BACKGROUND

In this chapter, we cover much of the background of rigid body kinematics and dynamics, as well as basic ideas in collision detection. We will see how rigid bodies are commonly represented in simulation, and how to formulate sets of dynamic constraints, including those for contact, friction, and joints. We also introduce methods for discretizing these constraints into formulations for time-stepping simulation schemes. Lastly, we briefly introduce some of the popular collision detection libraries for multibody simulation.

2.1 Kinematics

Kinematics refers to the mathematical descriptions of positions and orientations of bodies, or points on those bodies, as they move through space over time.

2.1.1 Spatial representation of rigid bodies

A rigid body's frame of reference is completely described with respect to the world frame by a position and a rotation (orientation). In three dimensional space, let the position of a body's frame $\{B\}$ at a point $\mathbf{x}$ be represented by the column vector $\mathbf{p} = \begin{bmatrix} p_x & p_y & p_z \end{bmatrix}^T$ where p_x, p_y, and p_z represent the x, y, and z values of a 3D coordinate. The rotation of the body from frame $\{A\}$ to frame $\{B\}$ may be represented by a rotation matrix $\mathbf{R}_{AB} = \begin{bmatrix} \hat{\mathbf{x}}_{AB} & \hat{\mathbf{y}}_{AB} & \hat{\mathbf{z}}_{AB} \end{bmatrix}$, where $\hat{\mathbf{x}}_{AB}$, $\hat{\mathbf{y}}_{AB}$, and $\hat{\mathbf{z}}_{AB}$ are the unit vectors representing the principal axes of frame $\{B\}$ in frame $\{A\}$ coordinates [31]. $\mathbf{R}_{AB}$ is said to be the rotation matrix "from A to B." The rotation matrix is composed of nine scalar values r_{ij} where

$$\mathbf{R}_{AB} = \begin{bmatrix} \hat{\mathbf{x}}_B \cdot \hat{\mathbf{x}}_A & \hat{\mathbf{y}}_B \cdot \hat{\mathbf{x}}_A & \hat{Z}_B \cdot \hat{\mathbf{x}}_A \\ \hat{\mathbf{x}}_B \cdot \hat{\mathbf{y}}_A & \hat{\mathbf{y}}_B \cdot \hat{\mathbf{y}}_A & \hat{Z}_B \cdot \hat{\mathbf{y}}_A \\ \hat{\mathbf{x}}_B \cdot \hat{\mathbf{z}}_A & \hat{\mathbf{y}}_B \cdot \hat{\mathbf{z}}_A & \hat{Z}_B \cdot \hat{\mathbf{z}}_A \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

Upon inspection of this matrix describing $\{B\}$ with respect to $\{A\}$, we notice that each row i is indeed a principal axis of $\{A\}$ with respect to $\{B\}$. In other words, the transpose of rotation matrix gives the rotation matrix in the reverse order, or $\mathbf{R}_{AB}^T = \mathbf{R}_{BA}$. Further, we note that multiplying $\mathbf{R}_{AB}\mathbf{R}_{BA}$ results in the identity I_3. This is precisely the definition of a matrix inverse, which leads us to write

$$\mathbf{R}_{AB} = \mathbf{R}_{BA}^T = \mathbf{R}_{BA}^{-1}$$

and declare that the inverse of a rotation matrix is equal to its transpose. As a final observation on the properties of rotation matrices, we note that a rotation matrix is said to be "orthonormal" since it is both orthogonal and composed of unit vectors. It can be shown that for a given rotation matrix $\mathbf{R}$, $det(\mathbf{R}) = 1$ and $||\mathbf{R}|| = 1$.

2.1.2 Mapping and transformation

Mapping refers to the conversion, or transformation, of a point or vector from one reference frame to another. Consider the following example: Given frames $\{A\}$ and $\{B\}$, the vector $\mathbf{p}_{AB}$ from $\{A\}$ to $\{B\}$, and the vector $\mathbf{p}_{BC}$ what is $\mathbf{p}_{AC}$? To convert a vector from frame $\{B\}$ to frame $\{A\}$, we simply multiply the rotation matrix $\mathbf{R}_{AB}$ by the vector's representation in $\{B\}$. Here we begin see a pattern in the notation. The solution is $\mathbf{p}_{AC} = \mathbf{p}_{AB} + \mathbf{R}_{AB}\mathbf{p}_{BC}$. Indeed, if we were to consider a similar example finding $\mathbf{p}_{AZ}$, the solution would simply be

$$\mathbf{p}_{AZ} = \mathbf{p}_{AB} + \mathbf{R}_{AB}\mathbf{p}_{BC} + \mathbf{R}_{AC}\mathbf{p}_{CD} + \mathbf{R}_{AE}\mathbf{p}_{EF} + ... + \mathbf{R}_{AY}\mathbf{p}_{YZ}$$

where for example $\mathbf{R}_{AD}$ is given by $\mathbf{R}_{AD} = \mathbf{R}_{AB}\mathbf{R}_{BC}\mathbf{R}_{CD}$.

It is common to combine the rotation and translation from one frame to another into a single matrix $\mathbf{T}$ we call a *transformation matrix*. Given the rotation matrix $\mathbf{R}_{AB}$ and the column vector $\mathbf{p}_{AB}$, we define

$$\mathbf{T}_{AB} = \left[\begin{array}{ccc|c} & \mathbf{R}_{AB} & & \mathbf{p}_{AB} \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

Examining this definition while considering the example above finding P_{AC}, we see

that we can rewrite the solution compactly as $\bar{\mathbf{p}}_{AC} = \mathbf{T}_{AB}\bar{\mathbf{p}}_{BC}$ where $\bar{\mathbf{p}}$ is a homogenous vector $[p_x\ p_y\ p_z\ 1]^T$. If we consider the example finding $\bar{\mathbf{p}}_{AZ}$, the notation really goes to work for us with the solution $\bar{\mathbf{p}}_{AZ} = \mathbf{T}_{AB}\mathbf{T}_{BC}\mathbf{T}_{CD}\ldots\mathbf{T}_{XY}\bar{\mathbf{p}}_{YZ}$.

Note that the reverse transformation matrix is not so neat as the rotation matrix alone,

$$\mathbf{T}_{BA} = \left[\begin{array}{c|c} \mathbf{R}_{AB}^T & -\mathbf{R}_{AB}^T\mathbf{p}_{AB} \\ \hline 0\ \ 0\ \ 0 & 1 \end{array}\right]$$

2.1.3 Alternative rotation representations

There are many ways of representing a 3D rotation [32, 33]. The following are brief descriptions of the most popular rotation representations.

Euler Angles

Rotating with Euler angles involves three rotations by angles of α, β, and γ about three primary axes. The axes used can be any combination of X, Y, and Z, as long as no redundant rotations occur, e.g. XXY. This leaves 12 total possible rotation combinations. Sometimes a distinction is made between "classical" Euler angles whose first and third rotations are about the same axis, and Cardan or Tait-Bryan angles, whose rotations use all three primary axes.

If the orientation of the axes of rotation change with the system, the rotations are said to be *intrinsic*. If the rotations occur about axes fixed with respect to the world frame, they are said to be *extrinsic*.

Axis-Angle and Exponential Notation

The aptly named axis-angle rotation representation defines rotation by an angle θ about an axis defined by unit vector $\hat{\mathbf{k}}$. This is directly related to exponential notation where $e^{[\hat{\mathbf{k}}]_\times\theta}$ corresponds to the rotation by θ radians about unit vector $\hat{\mathbf{k}}$ where $[\ ]_\times$ denotes the cross product matrix

$$[\mathbf{k}]_\times = \begin{bmatrix} 0 & -k_z & k_y \\ k_z & 0 & -k_x \\ -k_y & k_x & 0 \end{bmatrix}. \tag{2.1}$$

The definition is found in the Euler-Rodrigues formula

$$\mathbf{R} = e^{[\hat{\mathbf{k}}]_\times \theta} = \mathbf{I}_3 cos(\theta) + [\mathbf{k}]_\times \sin(\theta) + (1 - \cos(\theta))\mathbf{k}\mathbf{k}^T \tag{2.2}$$

which is derived by expanding $e^{[\hat{\mathbf{k}}]_\times \theta}$ using Taylor series expansion.

Quaternions

Strictly speaking, it is unit quaternions that are used for spacial rotation. The quaternion takes the form of a four-tuple and may be written as $\{a, b, c, d\}$ or $a + bi + cj + dk$ where i, j, and k follow Hamilton's rules:

$$\begin{gathered} i^2 = j^2 = k^2 = ijk = -1 \\ ij = -ji = k \\ jk = -kj = i \\ ki = -ik = j. \end{gathered}$$

Given the quaternion $q = a + bi + cj + dk$, one can calculate the rotation matrix R corresponding to q with

$$\mathbf{R} = \begin{bmatrix} 1 - 2(c^2 + d^2) & 2(bc - ad) & 2(ac + bd) \\ 2(bc + ad) & 1 - 2(b^2 + d^2) & 2(cd - ab) \\ 2(bd - ac) & 2(ab + cd) & 1 - 2(b^2 + c^2) \end{bmatrix}$$

Quaternions are often favored because they do not suffer from Gimbal lock, they require less storage space since they are composed of only four real numbers, and are numerically more stable since they are easily normalized and have fewer sources of error than rotation matrices when being integrated.

2.2 Dynamics

Dynamics, or classical mechanics, is the branch of physics that deals with forces and torques on bodies, and the behavior consequences of those forces. We focus particularly on those forces generated by contact and friction.

2.2.1 Twist, wrench, and inertia

Consider a world frame $\{W\}$ in which we place a rigid body with an inertial frame $\{B\}$, fixed with respect to the body and with its origin coincident with the body's center of mass (this coincidence is the convention, and is mathematically convenient, but not required). The orientation of the body is given by the position $\mathbf{u} \in \mathbb{R}^3$ and a rotation matrix $\mathbf{R}_{WB} = \begin{bmatrix} \hat{\mathbf{x}}_B \cdot \hat{X}_W & \hat{\mathbf{y}}_B \cdot \hat{\mathbf{x}}_W & \hat{\mathbf{z}}_B \cdot \hat{\mathbf{x}}_W \\ \hat{\mathbf{x}}_B \cdot \hat{Y}_W & \hat{\mathbf{y}}_B \cdot \hat{\mathbf{y}}_W & \hat{\mathbf{z}}_B \cdot \hat{\mathbf{y}}_W \\ \hat{\mathbf{x}}_B \cdot \hat{Z}_W & \hat{\mathbf{y}}_B \cdot \hat{\mathbf{z}}_W & \hat{\mathbf{z}}_B \cdot \hat{\mathbf{z}}_W \end{bmatrix} \in \mathbb{R}^{3\times3}$. The translational velocity $\mathbf{v} \in \mathbb{R}^3$ contains scalar components representing the velocity in the direction of the primary axes. The angular velocity $\boldsymbol{\omega} \in \mathbb{R}^3$ contains scalar compenets representing the angular velocity about the primary axes in the right-handed sense. The generalized velocity of of the body is the stacked column vector $\boldsymbol{\nu} = \begin{bmatrix} \mathbf{v} \\ \boldsymbol{\omega} \end{bmatrix}$. For an arbitrary point with respect to the body, $\boldsymbol{\nu}$ is composed of the translational and rotational velocity of that point expressed in $\{W\}$, and is called the twist [34]. From here on, when we refer to velocity, we are speaking of the generalized velocity.

Consider a point c_i on the body given by a vector $\mathbf{r} \in \mathbf{R}^3$ from the body's center of mass to the point, with a fixed frame $\{C\}$. The velocity of the point c_i with respect to the world frame is given by

$$\begin{bmatrix} \mathbf{v}^W_{i_{obj}} \\ \boldsymbol{\omega}^W_{obj} \end{bmatrix} = \mathbf{P}_i^T \boldsymbol{\nu}, \tag{2.3}$$

where

$$\mathbf{P}_i = \begin{bmatrix} \mathbf{I}_{3\times3} & \mathbf{0} \\ [\mathbf{r}]_\times & \mathbf{I}_{3\times3} \end{bmatrix}.$$

If we want to know the twist of the origin of $\{\mathbf{C}\}_i$ given in frame $\{\mathbf{C}\}_i$, we can transform the left hand side of equation (2.3). Let $\mathbf{R}_i$ represent the rotation matrix

from $\{\mathbf{W}\}$ to $\{\mathbf{C}\}_i$. The twist represented in $\{\mathbf{C}\}_i$ is given by

$$\begin{bmatrix} \mathbf{R}_i^T & \mathbf{0} \\ \mathbf{0} & \mathbf{R}_i^T \end{bmatrix} \begin{bmatrix} \mathbf{v}_{i_obj}^W \\ \boldsymbol{\omega}_{obj}^W \end{bmatrix}. \tag{2.4}$$

The inertia tensor $\bar{\mathbf{J}}$ is a 3×3 matrix of moments of inertia and relates a body's angular momentum to its angular velocity. A body with volume V has angular momentum $\mathbf{L}$ which may be written as $\mathbf{L} = \bar{\mathbf{J}}\boldsymbol{\omega}$ or

$$\begin{bmatrix} L_x \\ L_y \\ L_z \end{bmatrix} = \begin{bmatrix} \int (y^2 + z^2)\, dV & -\int xy\, dV & -\int xz\, dV \\ -\int xy\, dV & \int (z^2 + x^2)\, dV & -\int yz\, dV \\ -\int zx\, dV & -\int yz\, dV & \int (x^2 + y^2)\, dV \end{bmatrix} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}. \tag{2.5}$$

We can transform $\bar{\mathbf{J}}$ into the world frame inertia tensor $\mathbf{J}$ by

$$\mathbf{J} = \mathbf{R}\bar{\mathbf{J}}\mathbf{R}^T. \tag{2.6}$$

2.2.2 Newton-Euler equations

The Newton-Euler equations are simply the combination of Newton's second [35, 36] law with Euler's second law [37].

$$\begin{aligned} \text{Newton's 2nd Law:} \quad & \mathbf{f} = m\dot{\mathbf{v}} \\ \text{Euler's 2nd Law:} \quad & \boldsymbol{\tau} = \mathbf{J}\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times \mathbf{J}\boldsymbol{\omega} \end{aligned}$$

where $\mathbf{f}$ is force, m is mass, $\dot{\mathbf{v}}$ is the translational acceleration, $\boldsymbol{\tau}$ is torque, $\mathbf{J}$ is inertia, and $\boldsymbol{\omega}$ is angular velocity. We can combine these into a single equation

$$\mathbf{M}(\mathbf{q})\dot{\boldsymbol{\nu}} = \mathbf{G}(\mathbf{q})\boldsymbol{\lambda}_{app} + \boldsymbol{\lambda}_{ext} \tag{2.7}$$

where $\mathbf{q}$ is the generalized position $\mathbf{q} = \begin{bmatrix} \mathbf{u} \\ q \end{bmatrix}$ given quaternion q, $\mathbf{M}(\mathbf{q})$ is the mass-inertia matrix with the form $\mathbf{M} = \begin{bmatrix} m\mathbf{I}_3 & \mathbf{0} \\ \mathbf{0} & \mathbf{J} \end{bmatrix}$, $\dot{\boldsymbol{\nu}}$ is the derivative of the generalized

velocity vector, and $\boldsymbol{\lambda}$ is the generalized force containing the torque $\boldsymbol{\lambda} = \begin{bmatrix} \mathbf{f} \\ \boldsymbol{\tau} \end{bmatrix}$. We have also broken up the forces into an applied force $\boldsymbol{\lambda}_{app}$ and external force $\boldsymbol{\lambda}_{ext}$. The matrix $\mathbf{G}(\mathbf{q})$ is a Jacobian matrix and contains information about the point on the body at which $\boldsymbol{\lambda}_{app}$ is being applied. It essentially translates the applied force into its translational and rotational components with respect to the body frame. When associated with a force $\boldsymbol{\lambda}$ in direction $\hat{\mathbf{n}}$,

$$\mathbf{G}(\mathbf{q}) = \begin{bmatrix} \hat{\mathbf{n}} \\ \mathbf{r} \times \hat{\mathbf{n}} \end{bmatrix} \tag{2.8}$$

where $\mathbf{r}$ is the vector from the center of mass to the point of application of $\boldsymbol{\lambda}$. In equation (2.7), we could have also multiplied a $\mathbf{G}$ by $\boldsymbol{\lambda}_{ext}$, but it would have simplified to an identity since the point of application of the external force is through the center of mass.

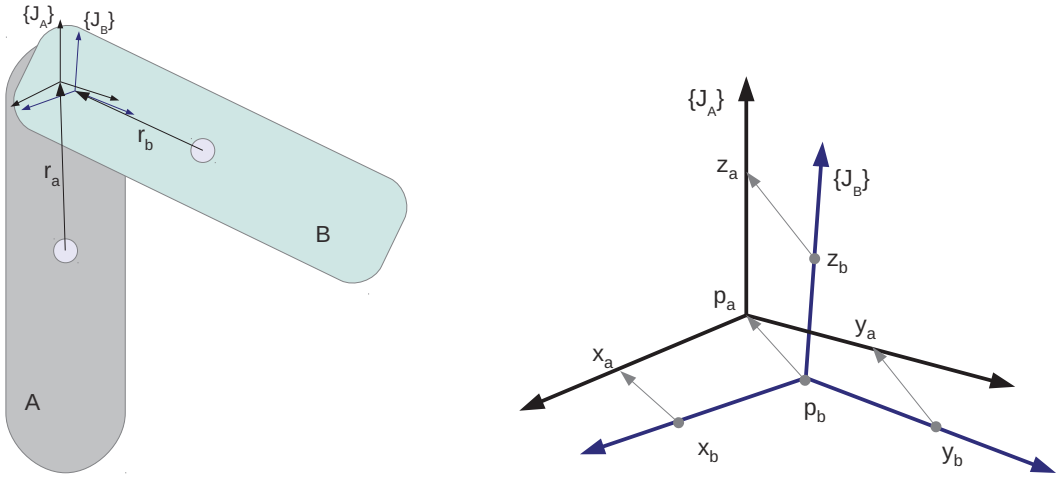
2.2.3 Bilateral constraint

Joints may be modelled as bilateral constraints as

$$\mathbf{0} = \boldsymbol{\psi}_b(\mathbf{q}, t) \tag{2.9}$$

where $\boldsymbol{\psi}_b(\mathbf{q}, t)$ is the constraint violation as a function of configuration $\mathbf{q}$ and time t. Consider the joint depicted in Figure 2.1.

Joint constraints attempt to constrain the relative positions of joint frames of either one body to the world frame or two bodies relative to one another. Given two bodies A and B, each has their own joint frame $\{J_a\}$ and $\{J_b\}$ fixed relative to their respective bodies. Let $\mathbf{r}_a$ be the vector from the center of mass of body A to the joint frame $\{J_a\}$, and let $\hat{\mathbf{x}}_a$, $\hat{\mathbf{y}}_a$, and $\hat{\mathbf{z}}_a$ be the unit vectors corresponding to the primary axes of $\{J_a\}$, all expressed in the world frame. Body b has analogously defined vectors for its joint frame $\{J_b\}$. We may write the joint constraint Jacobian as the stacked matrix $\mathbf{G}_b = \begin{bmatrix} \mathbf{G}_A \\ \mathbf{G}_B \end{bmatrix}$. If we were to constrain both position and



(a) Two joined bodies and their respective joint frame coordinate systems $\{J_A\}$ and $\{J_B\}$.

(b) The set of joint axes and possible errors in origins as well as unit points along constraint directions.

Figure 2.1: A joint constraint between two bodies $\mathcal{A}$ and $\mathcal{B}$. Every joints maintains two joint frames, one per body involved. The discrepancies in these two frames correspond to the joint error.

rotation about all three axes, we would have a rigid or "fixed" joint where

$$\mathbf{G}_A = \begin{bmatrix} \hat{\mathbf{x}}_a & \hat{\mathbf{y}}_a & \hat{\mathbf{z}}_a & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{r}_a \times \hat{\mathbf{x}}_a & \mathbf{r}_a \times \hat{\mathbf{y}}_a & \mathbf{r}_a \times \hat{\mathbf{z}}_a & \hat{\mathbf{x}}_a & \hat{\mathbf{y}}_a & \hat{\mathbf{z}}_a \end{bmatrix}$$

$$\mathbf{G}_B = \begin{bmatrix} -\hat{\mathbf{x}}_a & -\hat{\mathbf{y}}_a & -\hat{\mathbf{z}}_a & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{r}_b \times -\hat{\mathbf{x}}_a & \mathbf{r}_b \times -\hat{\mathbf{y}}_a & \mathbf{r}_b \times -\hat{\mathbf{z}}_a & -\hat{\mathbf{x}}_a & -\hat{\mathbf{y}}_a & -\hat{\mathbf{z}}_a \end{bmatrix} \tag{2.10}$$

We can think of the six columns of equation (2.10) as respectfully constraining the x position, y position, z position, x rotation, y rotation, and z rotation with regard to the assumed initially coincident joint frames $\{J_a\}$ and $\{J_b\}$. From here, we can create any common joint we wish by simply masking out the columns for which we would like to introduce a degree of freedom. A revolute joint about the $\hat{\mathbf{z}}$ axis would include columns one through five, but eliminate column six, thus allowing non-zero rotational velocity about the joint's $\hat{\mathbf{z}}$ direction. Table 2.1 contains the Jacobian definitions (for $\mathbf{G}_A$) for several common joint types.

Table 2.1: Joint constraint Jacobian definitions.

Joint Type	Jacobian
Rigid (fixed)	$\mathbf{G}_A = \begin{bmatrix} \hat{\mathbf{x}}_a & \hat{\mathbf{y}}_a & \hat{\mathbf{z}}_a & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{r}_a \times \hat{\mathbf{x}}_a & \mathbf{r}_a \times \hat{\mathbf{y}}_a & \mathbf{r}_a \times \hat{\mathbf{z}}_a & \hat{\mathbf{x}}_a & \hat{\mathbf{y}}_a & \hat{\mathbf{z}}_a \end{bmatrix}$
Revolute	$\mathbf{G}_A = \begin{bmatrix} \hat{\mathbf{x}}_a & \hat{\mathbf{y}}_a & \hat{\mathbf{z}}_a & \mathbf{0} & \mathbf{0} \\ \mathbf{r}_a \times \hat{\mathbf{x}}_a & \mathbf{r}_a \times \hat{\mathbf{y}}_a & \mathbf{r}_a \times \hat{\mathbf{z}}_a & \hat{\mathbf{x}}_a & \hat{\mathbf{y}}_a \end{bmatrix}$
Prismatic	$\mathbf{G}_A = \begin{bmatrix} \hat{\mathbf{x}}_a & \hat{\mathbf{y}}_a & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{r}_a \times \hat{\mathbf{x}}_a & \mathbf{r}_a \times \hat{\mathbf{y}}_a & \hat{\mathbf{x}}_a & \hat{\mathbf{y}}_a & \hat{\mathbf{z}}_a \end{bmatrix}$
Cylindrical	$\mathbf{G}_A = \begin{bmatrix} \hat{\mathbf{x}}_a & \hat{\mathbf{y}}_a & \mathbf{0} & \mathbf{0} \\ \mathbf{r}_a \times \hat{\mathbf{x}}_a & \mathbf{r}_a \times \hat{\mathbf{y}}_a & \hat{\mathbf{x}}_a & \hat{\mathbf{y}}_a \end{bmatrix}$
Spherical	$\mathbf{G}_A = \begin{bmatrix} \hat{\mathbf{x}}_a & \hat{\mathbf{y}}_a & \hat{\mathbf{z}}_a \\ \mathbf{r}_a \times \hat{\mathbf{x}}_a & \mathbf{r}_a \times \hat{\mathbf{y}}_a & \mathbf{r}_a \times \hat{\mathbf{z}}_a \end{bmatrix}$

2.3 Contact and the complementarity problem

Consider two bodies $\mathcal{A}$ and $\mathcal{B}$ near contact as depicted in Figure 2.2. A contact between two bodies may be represented as a 5-tuple $C = \{\mathcal{A}_{id}, \mathcal{B}_{id}, \mathbf{p}_a, \hat{\mathbf{n}}, \psi_n\}$, where $\mathcal{A}_{id}$ and $\mathcal{B}_{id}$ are body indices or identifiers for $\mathcal{A}$ and $\mathcal{B}$ respectively, $\mathbf{p}_a$ is the contact point on $\mathcal{A}$ in world coordinates, $\hat{\mathbf{n}}$ is the contact normal vector in the direction of $\mathcal{A}$ onto $\mathcal{B}$ by convention, and ψ_n is the signed gap distance between the two bodies. The point of contact $\mathbf{p}_b$ on $\mathcal{B}$ is available by $\mathbf{p}_b = \mathbf{p}_a + \psi_n \hat{\mathbf{n}}$. Vectors $\hat{\mathbf{t}}$ and $\hat{\mathbf{o}}$ form an orthonormal basis with $\hat{\mathbf{n}}$. The vectors $\mathbf{r}_a$ and $\mathbf{r}_b$ span from the center of mass of each body to the point of contact on that body. Although not required, we use the convention that a body's reference frame is coincident with its center of mass. There is the potential for a contact force $\boldsymbol{\lambda}_n$ along the normal vector which is intended to prevent ψ_n from becoming negative.

The complementarity problem offers a convenient way to model contact. There is a vast body of research regarding complementarity problems and methods for their solution [38, 39]. Given a mapping $\mathbf{f}(\mathbf{z}) : \mathbb{R}^n \to \mathbb{R}^n$, the complementarity problem

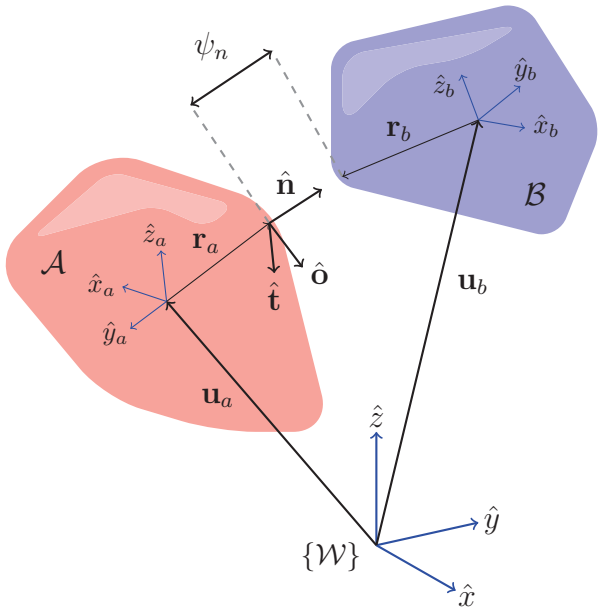


Figure 2.2: Bodies $\mathcal{A}$ and $\mathcal{B}$ in 3D, and contact vectors in the world frame $\{\mathcal{W}\}$. A potential contact force is enacted along the normal direction $\hat{\mathbf{n}}$. Frictional forces occur in the plane spanned by $\hat{\mathbf{t}}$ and $\hat{\mathbf{o}}$, which form an orthonormal basis with $\hat{\mathbf{n}}$.

(CP) seeks to find a solution vector $\mathbf{z} \in \mathbb{R}^n$ to satisfy

$$\begin{aligned} \mathbf{z} &\geq \mathbf{0} \\ \mathbf{f}(\mathbf{z}) &\geq \mathbf{0} \\ \mathbf{z}^T\mathbf{f}(\mathbf{z}) &= \mathbf{0} \end{aligned} \tag{2.11}$$

which may be expressed equivalently as

$$\mathbf{0} \leq \mathbf{f}(\mathbf{z}) \perp \mathbf{z} \geq 0 \tag{2.12}$$

There are many solvers available for CPs [40], and in particular the linear CP (LCP) [41, 42] where $\mathbf{f}(\mathbf{z})$ is a linear function. There are also specialized solvers which attempt to exploit properties specifically related to multibody dynamics problems [43].

2.3.1 Contact constraint

A unilateral contact constraint is intended to prevent interpenetration of bodies given the contact information depicted in Figure 2.2 (we will see in 5.3 why this

is subject to error). The standard approach of this constraint [44] requires

$$\begin{aligned} \lambda_n &\geq 0 \\ \psi(\mathbf{q},t) &\geq 0 \\ \lambda_n \psi(\mathbf{q},t) &= 0 \end{aligned} \tag{2.13}$$

or following the format of (2.12),

$$0 \leq \psi(\mathbf{q},t) \perp \lambda_n \geq 0 \tag{2.14}$$

demonstrating how useful the complementarity problem is for modeling contact.

2.3.2 Friction constraint

There are several choices of friction model, but a typical choice is Coulomb's dry friction model [45]. Coulomb's model is expressed as

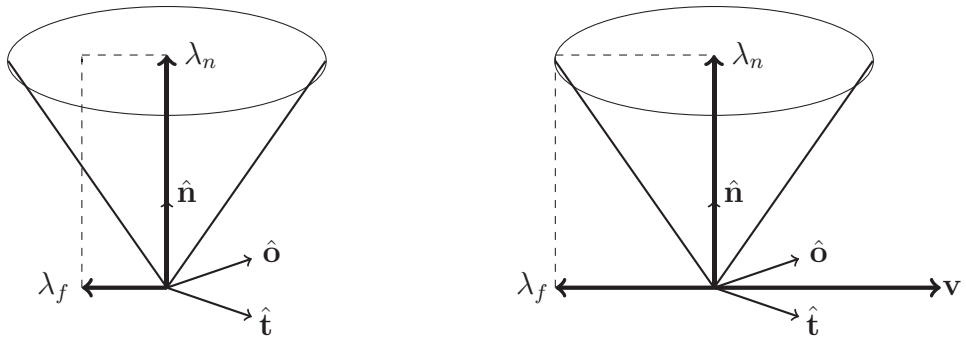
$$||\boldsymbol{\lambda}_f|| \leq \mu ||\boldsymbol{\lambda}_n|| \tag{2.15}$$

where $\boldsymbol{\lambda}_f$ is the friction force and μ is the coefficient of friction between the two bodies at the point of contact. This inequality corresponds to a friction cone as depicted in Figure 2.3. The contact may be in one of two states in terms of friction. When the relative velocity of the contact points is zero, the contact is said to be sticking or rolling, otherwise the contact is said to be sliding. In the case of sticking, the magnitude of the friction force λ_f is bounded by

$$0 \leq \lambda_f \leq \mu \lambda_n$$

and in the case of sliding the magnitude of the friction force is given by

$$\lambda_f = \mu \lambda_n.$$



(a) Sticking case. **(b) Sliding case.**

Figure 2.3: The friction cone of Coulomb's dry friction model. The cone represents to the possible frictional force λ_f corresponding to the normal force λ_n. The size of the base of the cone (top) is determined by the friction coefficient μ and λ_n. When sticking, λ_f will project to the interior of the base, and when sliding, λ_f is constrained to the perimeter of the base.

The direction of $\boldsymbol{\lambda}_f$ is opposite that of the velocity vector, and may be in any direction in the case of sticking. We define the friction cone $\mathcal{F}(\mu, \lambda_n)$ as

$$\mathcal{F}(\mu, \lambda_n) = \{(\lambda_t, \lambda_o : \mu^2\lambda_o^2 - \lambda_t^2 - \lambda_o^2 \geq 0\} \tag{2.16}$$

where λ_t and λ_o are the magnitudes of vectors $\boldsymbol{\lambda}_t$ and $\boldsymbol{\lambda}_o$ which form an orthogonal basis with $\boldsymbol{\lambda}_n$ [46]. The maximum dissipation principle allows us to write Coulomb's law as

$$(\lambda_t, \lambda_o) \in \underset{(\lambda_t,\lambda_o)\in\mathcal{F}}{\arg\max}(-\lambda_t v_t - \lambda_o v_o) \tag{2.17}$$

where v_t and v_o are the velocity vector components in the $\hat{\mathbf{t}}$ and $\hat{\mathbf{o}}$ directions.

Following the derivations in [46, 11], we can write Coulomb's law as a nonlinear problem with multiple contacts as

$$\mathbf{0} = (\mathbf{U}\boldsymbol{\lambda}_n) \circ (\mathbf{v}_t) + \boldsymbol{\lambda}_t \circ \boldsymbol{\sigma} \tag{2.18}$$

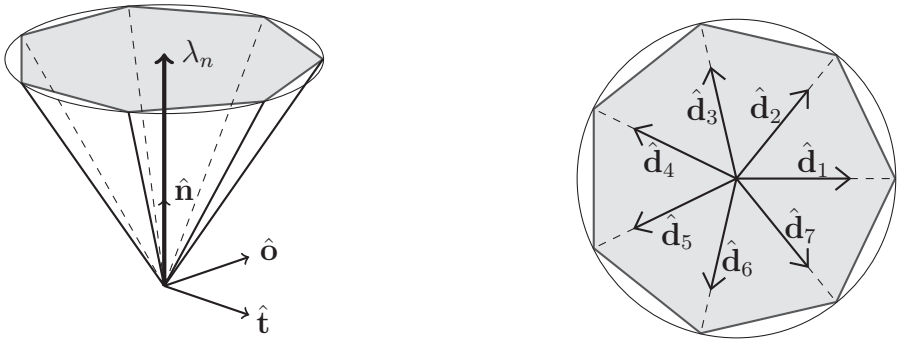
$$\mathbf{0} = (\mathbf{U}\boldsymbol{\lambda}_n) \circ (\mathbf{v}_o) + \boldsymbol{\lambda}_o \circ \boldsymbol{\sigma} \tag{2.19}$$

$$\mathbf{0} \leq \boldsymbol{\sigma} \perp (\mathbf{U}\boldsymbol{\lambda}_n) \circ (\mathbf{U}\boldsymbol{\lambda}_n) - \boldsymbol{\lambda}_t \circ \boldsymbol{\lambda}_t - \boldsymbol{\lambda}_o \circ \boldsymbol{\lambda}_o \geq 0 \tag{2.20}$$

where for each contact j, $\mathbf{U}$ is a diagonal matrix composed of coefficients of friction μ_j and $\boldsymbol{\sigma}$ is the slip speed. The $\circ$ operator denotes the Hadamard product for vectors and matrices where for example

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \circ \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix} = \begin{pmatrix} a_{11}\,b_{11} & a_{12}\,b_{12} & a_{13}\,b_{13} \\ a_{21}\,b_{21} & a_{22}\,b_{22} & a_{23}\,b_{23} \\ a_{31}\,b_{31} & a_{32}\,b_{32} & a_{33}\,b_{33} \end{pmatrix}$$

We can linearize the friction constraint by approximating the friction cone as a polygonal cone as depicted in Figure 2.4. This allows us to write the friction constraint as a linear complementarity problem which is convenient. The linear



(a) Side view. **(b) View from above.**

Figure 2.4: The friction cone and its polygonal approximation for $n_d = 7$ friction directions as viewed from the side and top. In this linear model, the friction occurs along one of the n_d directions, but models the same friction law. Some error is introduced by the simplification, but solutions are easier to find and allow us to forgo non-linear solvers.

friction constraint takes the form

$$\begin{aligned} \mathbf{0} &\leq \mathbf{G}_f \boldsymbol{\nu}^{\ell+1} + \mathbf{E}\mathbf{s}^{\ell+1} \perp \mathbf{p}_f^{\ell+1} \geq \mathbf{0} \\ \mathbf{0} &\leq \mathbf{U}\mathbf{p}_n^{\ell+1} - \mathbf{E}^T\mathbf{p}_f^{\ell+1} \perp \mathbf{s}^{\ell+1} \geq \mathbf{0} \end{aligned} \tag{2.21}$$

where $\mathbf{s}$ corresponds to the relative sliding speed, $\mathbf{U}$ is a diagonal matrix composed of friction coefficients, $\mathbf{E}$ is a selection matrix given by

$$\mathbf{E} = blockdiag(\mathbf{e}_1, \ldots, \mathbf{e}_{n_c}), \text{ where } \mathbf{e}_i = ones(n_d, 1)$$

for n_c contacts, and the friction constraint Jacobian $\mathbf{G}_f$ is composed of submatrices $\mathbf{G}_{f_{ij}}$ for n_d friction directions in the linearized friction cone with

$$\mathbf{G}_{f_{ij}} = \begin{bmatrix} \hat{\mathbf{d}}_{i_1} & ... & \hat{\mathbf{d}}_{i_{nd}} \\ (\mathbf{r}_{ij} \times \hat{\mathbf{d}}_{i_1}) & ... & (\mathbf{r}_{ij} \times \hat{\mathbf{d}}_{i_{nd}}) \end{bmatrix} \tag{2.22}$$

where $\hat{\mathbf{d}}_{ik}$ is the k^{th} vector representing the friction cone.

2.4 Time stepping

Time stepping describes the process in which we keep track of all bodies, including their position, orientation, and other physical properties such as velocity, and formulate a time-stepping subproblem of which the solution gives the updated values for velocity at the end of the time step. Time-stepping simulation methods utilizing the complementarity problem were introduced by Moreau [17], and there was significant work regarding constraint based multibody dynamics in the 1980s [47–49].Using the backward Euler method to approximate the derivatives from step ℓ to $\ell + 1$ of size h, we have

$$\dot{\boldsymbol{\nu}}^{\ell+1} \approx \frac{\boldsymbol{\nu}^{\ell+1} - \boldsymbol{\nu}^{\ell}}{h} \tag{2.23}$$

and

$$\dot{\mathbf{q}}^{\ell+1} \approx \frac{\mathbf{q}^{\ell+1} - \mathbf{q}^{\ell}}{h} \tag{2.24}$$

If the mass matrix $\mathbf{M}$ and the constraint Jacobian $\mathbf{G}$ are to be evaluated at $\ell + 1$, then the time-stepping scheme is said to be fully-implicit. A semi-implicit scheme, such as Stewart-Trinkle [18], is one in which $\mathbf{M}$ and $\mathbf{G}$ are formulated at ℓ but constrained to be true at $\ell + 1$.

2.4.1 Discretization of Newton-Euler equations and constraints

We discretize constraints over a time step h in seconds from step ℓ to $\ell + 1$ as a time-stepping subproblem using the Taylor series approximation of a gap distance

$$\psi^{\ell+1} = \psi^{\ell} + \mathbf{G}^T(\mathbf{q})\boldsymbol{\nu}^{\ell+1}h + \frac{\partial \psi^{\ell}}{\partial t}h \tag{2.25}$$

for generalized velocity $\boldsymbol{\nu}_i = \begin{bmatrix} \mathbf{v}_i^T & \boldsymbol{\omega}_i^T \end{bmatrix}^T$, and where $\mathbf{G}(\mathbf{q})$ is the constraint Jacobian given for body i and constraint j by

$$\mathbf{G}_{ij} = \begin{bmatrix} \hat{\mathbf{n}}_{ij} \\ \mathbf{r}_{ij} \times \hat{\mathbf{n}}_{ij} \end{bmatrix} \tag{2.26}$$

where $\mathbf{r}$ is the vector from the center of mass of body i to the point of application of the force corresponding to constraint j on body i. The non-penetration constraint of equation (2.14) is then discretized to

$$\mathbf{0} \leq \boldsymbol{\psi}_n^\ell + \mathbf{G}_n^T \boldsymbol{\nu}^{\ell+1} h + \frac{\partial \boldsymbol{\psi}_n^\ell}{\partial t} h \perp \mathbf{p}_n^{\ell+1} \geq \mathbf{0} \tag{2.27}$$

where $\mathbf{p}_n$ are impulses, *i.e.,* $\mathbf{p} = h\boldsymbol{\lambda}$ for time step size h.

The bilateral constraint of equation (2.9) discretizes to

$$\mathbf{0} = \boldsymbol{\psi}_b^\ell + \mathbf{G}_b^T \boldsymbol{\nu}^{\ell+1} h + \frac{\partial \boldsymbol{\psi}_b^\ell}{\partial t} h \tag{2.28}$$

where $\boldsymbol{\psi}_b$ is the distance violation for each constrained dimension. It is not uncommon to see the terms $\boldsymbol{\psi}_b^\ell + \frac{\partial \boldsymbol{\psi}_b^\ell}{\partial t} h$ in equation (2.28) dropped, allowing the bilateral constraint to "drift" over time. Joint stabilization can then be utilized to correct this drift at the end of the time step after other constraints have been satisfied.

We enforce the Newton-Euler equation with all of our constraints

$$\mathbf{M}\dot{\boldsymbol{\nu}} = \mathbf{G}_b \boldsymbol{\lambda}_b + \mathbf{G}_n \boldsymbol{\lambda}_n + \mathbf{G}_f \boldsymbol{\lambda}_f + \boldsymbol{\lambda}_{ext} \tag{2.29}$$

where $\mathbf{M}$ is the mass-inertia matrix diagonally composed of $\mathbf{M}_i = \begin{bmatrix} m_i \mathbf{I}_3 & \mathbf{0} \\ \mathbf{0} & \mathbf{J}_i \end{bmatrix}$ for each i^{th} body with mass m_i and inertia tensor $\mathbf{J}_i$, $\dot{\boldsymbol{\nu}}$ is the first derivative of the generalized velocities, each $\boldsymbol{\lambda}$ is a column vector of forces resulting from constraints imposed upon the bodies, $\mathbf{G}$ is the corresponding constraint Jacobian, and $\boldsymbol{\lambda}_{ext}$ is a column vector of the external forces applied to the bodies, *e.g.,* gravity. We

discretize equation (2.29) as

$$\mathbf{M}\boldsymbol{\nu}^{\ell+1} = \mathbf{M}\boldsymbol{\nu}^{\ell} + \mathbf{G}_b\mathbf{p}_b^{\ell+1} + \mathbf{G}_n\mathbf{p}_n^{\ell+1} + \mathbf{G}_f\mathbf{p}_f^{\ell+1} + \mathbf{p}_{ext}^{\ell+1} \tag{2.30}$$

where $\mathbf{p}_{ext}$ are the external impulses, *e.g.,* gravity.

We can write this dynamic model in the form of a mixed LCP (MCP) by letting

$$\boldsymbol{\rho}_n^{\ell+1} = \mathbf{G}_n^T\boldsymbol{\nu}^{\ell+1} + \frac{\boldsymbol{\psi}_n}{h} + \frac{\partial\boldsymbol{\psi}_{\boldsymbol{n}}}{\partial t} \tag{2.31}$$

$$\boldsymbol{\rho}_f^{\ell+1} = \mathbf{G}_f^T\boldsymbol{\nu}^{\ell+1} + \frac{\partial\boldsymbol{\psi}_{\boldsymbol{f}}}{\partial t} \tag{2.32}$$

$$\boldsymbol{\sigma}^{\ell+1} = \mathbf{U}\mathbf{p}_n^{\ell+1} - \mathbf{E}^T\mathbf{p}_f^{\ell+1} \tag{2.33}$$

Then we have

$$\begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \boldsymbol{\rho}_n^{\ell+1} \\ \boldsymbol{\rho}_f^{\ell+1} \\ \boldsymbol{\sigma}^{\ell+1} \end{bmatrix} = \begin{bmatrix} -\mathbf{M} & \mathbf{G}_b & \mathbf{G}_n & \mathbf{G}_f & \mathbf{0} \\ \mathbf{G}_b^T & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{G}_n^T & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{G}_f^T & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{E} \\ \mathbf{0} & \mathbf{0} & \mathbf{U} & -\mathbf{E}^T & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\nu}^{\ell+1} \\ \boldsymbol{p}_b^{\ell+1} \\ \boldsymbol{p}_n^{\ell+1} \\ \boldsymbol{p}_f^{\ell+1} \\ \mathbf{s}^{\ell+1} \end{bmatrix} + \begin{bmatrix} \mathbf{M}\boldsymbol{\nu}^{\ell} + \mathbf{p}_{ext} \\ \frac{\boldsymbol{\psi}_b}{h} + \frac{\partial\boldsymbol{\psi}_{\boldsymbol{b}}}{\partial t} \\ \frac{\boldsymbol{\psi}_n}{h} + \frac{\partial\boldsymbol{\psi}_{\boldsymbol{n}}}{\partial t} \\ \frac{\partial\boldsymbol{\psi}_{\boldsymbol{f}}}{\partial t} \\ \mathbf{0} \end{bmatrix} \tag{2.34}$$

$$\mathbf{0} \leq \begin{bmatrix} \boldsymbol{\rho}_n^{\ell+1} \\ \boldsymbol{\rho}_f^{\ell+1} \\ \boldsymbol{\sigma}^{\ell+1} \end{bmatrix} \perp \begin{bmatrix} \mathbf{p}_n^{\ell+1} \\ \mathbf{p}_f^{\ell+1} \\ \mathbf{s}^{\ell+1} \end{bmatrix} \geq \mathbf{0} \tag{2.35}$$

which corresponds to the Stewart-Trinkle (ST) time-stepping method [18]. If we change the unilateral constraint terms in the right most portion of equation (2.34) to be zero, *i.e.,* replace $\frac{\psi_n}{h} + \frac{\partial\psi_n}{\partial t}$ with 0, we achieve the Anitescu-Potra method [50, 51].

At the end of each time step, body configurations are kinematically updated. Bodies that were not included in the time-stepping subproblem, *i.e.,* those that were unconstrained during the time step, have velocities updated by

$$\boldsymbol{\nu}^{\ell+1} \leftarrow \boldsymbol{\nu}^{\ell} + \frac{\lambda_{ext}}{m}h$$

Given a quaternion representation of rotation of $q = a + bi + cj + dk$, all dynamic

bodies are then updated according to

$$\mathbf{q}^{\ell+1} \leftarrow \mathbf{q}^{\ell} + \begin{bmatrix} \mathbf{I}_3 & \mathbf{0} \\ \mathbf{0} & \frac{1}{2}\begin{pmatrix} -b & -c & -d \\ a & d & -c \\ -d & a & b \\ c & -b & a \end{pmatrix} \end{bmatrix} \boldsymbol{\nu}^{\ell+1} h \tag{2.36}$$

As an alternate dynamics formulation to the MCP of equation (2.34), we can write the LCP in a form that can be passed to a solver. We can write the dynamics model as

$$\mathbf{A} = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{E} \\ \mathbf{U} & -\mathbf{E}^T & \mathbf{0} \end{bmatrix} - \begin{bmatrix} \mathbf{G}_n^T & \mathbf{0} \\ \mathbf{G}_f^T & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} -\mathbf{M} & \mathbf{G}_b \\ \mathbf{G}_b & \mathbf{0} \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{G}_n & \mathbf{G}_f & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{2.37}$$

$$\mathbf{b} = \begin{bmatrix} \frac{\boldsymbol{\psi}_n}{h} + \frac{\partial \boldsymbol{\psi}_n}{\partial t} \\ \frac{\partial \boldsymbol{\psi}_f}{\partial t} \\ \mathbf{0} \end{bmatrix} - \begin{bmatrix} \mathbf{G}_n^T & \mathbf{0} \\ \mathbf{G}_f^T & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} -\mathbf{M} & \mathbf{G}_b \\ \mathbf{G}_b & \mathbf{0} \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{M}\boldsymbol{\nu}^{\ell} + \mathbf{p}_{ext} \\ \frac{\boldsymbol{\psi}_b}{h} + \frac{\partial \boldsymbol{\psi}_b}{\partial t} \end{bmatrix} \tag{2.38}$$

From the solution $\mathbf{z}$, we determine $\boldsymbol{\nu}^{\ell+1}$ with

$$\boldsymbol{\nu}^{\ell+1} = \begin{bmatrix} -\mathbf{M} & \mathbf{G}_b \\ \mathbf{G}_b & \mathbf{0} \end{bmatrix}^{-1} \left(- \begin{bmatrix} \mathbf{G}_n & \mathbf{G}_f \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \mathbf{z} - \begin{bmatrix} \mathbf{M}\boldsymbol{\nu}^{\ell} + \mathbf{p}_{ext} \\ \frac{\boldsymbol{\psi}_b}{h} + \frac{\partial \boldsymbol{\psi}_b}{\partial t} \end{bmatrix} \right) \tag{2.39}$$

2.4.2 Impulse constraint correction

As mentioned, joint constraints are prone to drift within this maximum coordinate formulation. A joint correction algorithm is given in algorithm 1 where $\mathbf{C}$ is the bilateral constraint violation and $\mathbf{V}$ is the kinematic mapping onto $\mathbf{q}$. This algorithm is conceptually similar to that described by Bender [52], but is more general in that it could be used for any equality constraint for which we could determine $\mathbf{C}$.

Algorithm 1 Iterative joint constraint correction method

while $(norm(\mathbf{C}^{(\ell+1)}) > \epsilon_p)$

$\mathbf{p} = -(\mathbf{G}^{(\ell)T}\mathbf{M}^{-1}\mathbf{G}^{(\ell)})^{-1}\ \frac{\mathbf{C}^{(\ell+1)}}{h}$

$\Delta\boldsymbol{\nu} = \mathbf{M}^{-1}\ \mathbf{G}^{(\ell)}\ \mathbf{p}$

$\mathbf{q}^{(\ell+1)} = \mathbf{q}^{(\ell+1)} + \mathbf{V}\ \Delta\boldsymbol{\nu}\ h$

$\boldsymbol{\nu}^{(\ell+1)} = \boldsymbol{\nu}^{(l+1)} + \Delta\boldsymbol{\nu}$

Recalculate $\mathbf{C}^{(\ell+1)}$

$\mathbf{G}^{(\ell+1)} \leftarrow \mathbf{G}^{(\ell)}$

while $(norm(\dot{\mathbf{C}}^{(\ell+1)}) > \epsilon_{\boldsymbol{\nu}})$

$\mathbf{p} = -(\mathbf{G}^{(\ell+1)T}\mathbf{M}^{-1}\mathbf{G}^{(\ell+1)})^{-1}\ \dot{\mathbf{C}}^{(\ell+1)}$

$\Delta\boldsymbol{\nu} = \mathbf{M}^{-1}\ \mathbf{G}^{(l+1)}\ \mathbf{p}$

$\boldsymbol{\nu}^{(\ell+1)} = \boldsymbol{\nu}^{(\ell+1)} + \Delta\boldsymbol{\nu}$

Recalculate $\dot{\mathbf{C}}^{(\ell+1)}$

The joint constraint Jacobians of Algorithm 1 are different than the bilateral Jacobian definitions of equation (2.10). Here, we define these constraint Jacobians in Table 2.2 in terms of a primary bilateral vector, which we choose by convention to be the $\hat{\mathbf{z}}$ axis of the joint.

Table 2.2: Joint correction constraint Jacobian definitions.

Joint Type	**Jacobian**
Rigid (fixed)	$\mathbf{G} = \begin{bmatrix} \hat{\mathbf{x}}_a & \hat{\mathbf{y}}_a & \hat{\mathbf{z}}_a & \hat{\mathbf{x}}_a & \hat{\mathbf{y}}_a & \hat{\mathbf{y}}_a \\ \mathbf{r}_a \times \hat{\mathbf{x}}_a & \mathbf{r}_a \times \hat{\mathbf{y}}_a & \mathbf{r}_a \times \hat{\mathbf{z}}_a & \mathbf{r}_{z_a} \times \hat{\mathbf{x}}_a & \mathbf{r}_{\mathbf{z}_a} \times \hat{\mathbf{y}}_a & \mathbf{r}_{\mathbf{x}_a} \times \hat{\mathbf{y}}_a \end{bmatrix}$
Revolute	$\mathbf{G} = \begin{bmatrix} \hat{\mathbf{x}}_a & \hat{\mathbf{y}}_a & \hat{\mathbf{z}}_a & \hat{\mathbf{x}}_a & \hat{\mathbf{y}}_a \\ \mathbf{r}_a \times \hat{\mathbf{x}}_a & \mathbf{r}_a \times \hat{\mathbf{y}}_a & \mathbf{r}_a \times \hat{\mathbf{z}}_a & \mathbf{r}_{z_a} \times \hat{\mathbf{x}}_a & \mathbf{r}_{z_a} \times \hat{\mathbf{y}}_a \end{bmatrix}$
Prismatic	$\mathbf{G} = \begin{bmatrix} \hat{\mathbf{x}}_a & \hat{\mathbf{y}}_a & \hat{\mathbf{x}}_a & \hat{\mathbf{y}}_a & \hat{\mathbf{y}}_a \\ \mathbf{r}_a \times \hat{\mathbf{x}}_a & \mathbf{r}_a \times \hat{\mathbf{y}}_a & \mathbf{r}_{z_a} \times \hat{\mathbf{x}}_a & \mathbf{r}_{z_a} \times \hat{\mathbf{y}}_a & \mathbf{r}_{x_a} \times \hat{\mathbf{y}}_a \end{bmatrix}$
Cylindrical	$\mathbf{G} = \begin{bmatrix} \hat{\mathbf{x}}_a & \hat{\mathbf{y}}_a & \hat{\mathbf{x}}_a & \hat{\mathbf{y}}_a \\ \mathbf{r}_a \times \hat{\mathbf{x}}_a & \mathbf{r}_a \times \hat{\mathbf{y}}_a & \mathbf{r}_{z_a} \times \hat{\mathbf{x}}_a & \mathbf{r}_{z_a} \times \hat{\mathbf{y}}_a \end{bmatrix}$
Spherical	$\mathbf{G} = \begin{bmatrix} \hat{\mathbf{x}}_a & \hat{\mathbf{y}}_a & \hat{\mathbf{z}}_a \\ \mathbf{r}_a \times \hat{x}_a & \mathbf{r}_a \times \hat{\mathbf{y}}_a & \mathbf{r}_a \times \hat{\mathbf{z}}_a \end{bmatrix}$

Table 2.3: Joint correction constraint violation definitions.

Joint Type	Error
Rigid (fixed)	$\mathbf{C} = \begin{bmatrix} \mathbf{p}_{a_x} - \mathbf{p}_{b_x} \\ \mathbf{p}_{a_y} - \mathbf{p}_{b_y} \\ \mathbf{p}_{a_z} - \mathbf{p}_{b_z} \\ \mathbf{z}_{a_x} - \mathbf{z}_{b_x} \\ \mathbf{z}_{a_y} - \mathbf{z}_{b_y} \\ \mathbf{x}_{a_y} - \mathbf{x}_{b_y} \end{bmatrix}$
Revolute	$\mathbf{C} = \begin{bmatrix} \mathbf{p}_{a_x} - \mathbf{p}_{b_x} \\ \mathbf{p}_{a_y} - \mathbf{p}_{b_y} \\ \mathbf{p}_{a_z} - \mathbf{p}_{b_z} \\ \mathbf{z}_{a_x} - \mathbf{z}_{b_x} \\ \mathbf{z}_{a_y} - \mathbf{z}_{b_y} \end{bmatrix}$
Prismatic	$\mathbf{C} = \begin{bmatrix} \mathbf{p}_{a_x} - \mathbf{p}_{b_x} \\ \mathbf{p}_{a_y} - \mathbf{p}_{b_y} \\ \mathbf{z}_{a_x} - \mathbf{z}_{b_x} \\ \mathbf{z}_{a_y} - \mathbf{z}_{b_y} \\ \mathbf{z}_{a_z} - \mathbf{z}_{b_z} \end{bmatrix}$
Cylindrical	$\mathbf{C} = \begin{bmatrix} \mathbf{p}_{a_x} - \mathbf{p}_{b_x} \\ \mathbf{p}_{a_y} - \mathbf{p}_{b_y} \\ \mathbf{z}_{a_x} - \mathbf{z}_{b_x} \\ \mathbf{z}_{a_y} - \mathbf{z}_{b_y} \end{bmatrix}$
Spherical	$\mathbf{C} = \begin{bmatrix} \mathbf{p}_{a_x} - \mathbf{p}_{b_x} \\ \mathbf{p}_{a_y} - \mathbf{p}_{b_y} \\ \mathbf{p}_{a_z} - \mathbf{p}_{b_z} \end{bmatrix}$

A revolute joint example was run for the pendulum configuration depicted in Figure 2.5 where a dynamic body (blue) is constrained to a static body (gray) and set to swing. Two simulations were executed: one with joint correction, and one without. The error over the first 4000 iterations of these simulations is shown in Figure 6.4. We can see that the error is periodic in both corrected and non-corrected

runs, but the position error in the non-corrected run increases over time. This drift is stabilized in the corrected run using Algorithm 1. Note that both position and velocity error for the corrected run are stably maintained within values of 10^{-5} since this was the epsilon value used for both the position ϵ_p and velocity ϵ_v.

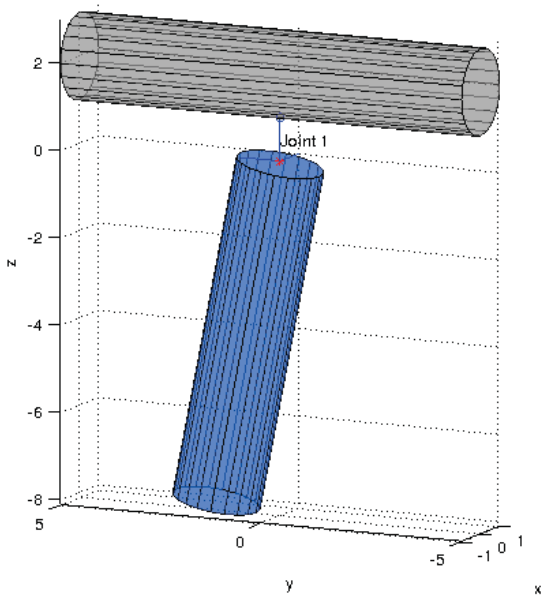
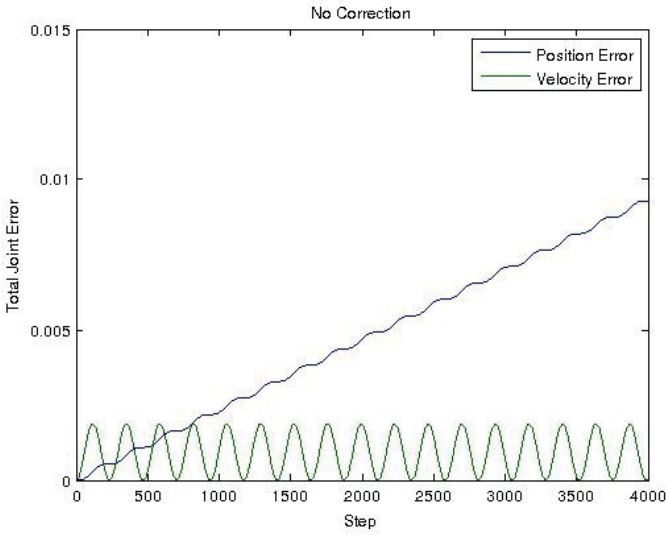


Figure 2.5: A hanging pendulum. A single joint constrains the motion of the dynamic body (blue) with respect to the the static body (gray).

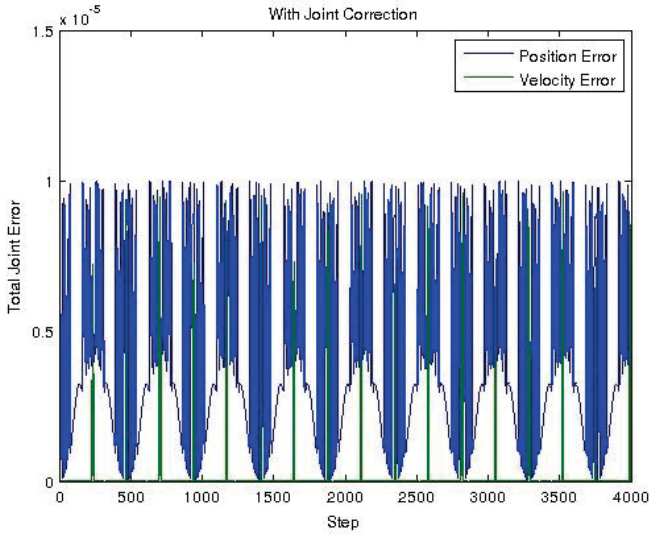
This type of correction to bilateral constraint violation is useful and shown to be quick, but because it is executed at the end of the time step after all other constraints have been satisfied, it ignores all other constraints such as contact. As a result, it is possible to violate the constraints that were just implicitly solved for in the time-stepping subproblem. Because of this, this method of joint stabilization requires small time steps in order to avoid violating constraints too deeply to recover from in the next time step.

2.5 Solution methods

There are several numerical methods for solving problems in the forms of LCP, mLCP, and NCP, all with varying advantages and disadvantages. One set of



(a) Joint error with no correction.



(b) Stabalized joint error with Correction.

Figure 2.6: Position and velocity error for pendulum simulations.

popular methods for solving systems of linear equations are variants of the iterative Gauss-Seidel method [53–55].

The standard Gauss-Seidel scheme for solving $\mathbf{Ax} = \mathbf{b}$ is loosely derived as follows. Decompose $\mathbf{A}$ into its lower triangular and strictly upper triangular components $\mathbf{A} = \mathbf{L} + \mathbf{U}$. We may then write $\mathbf{Lx} = \mathbf{b} - \mathbf{Ux}$, or as a fixed-point iteration $\mathbf{x}^{k+1} = \mathbf{L}^{-1}(\mathbf{b} - \mathbf{U}\mathbf{x}^{(k)})$. It follows that the element-wise iterative scheme for solving

$\mathbf{x}$ with sequential forward substitution is given for i from 1 to N as

$$x_i^{(k+1)} \leftarrow \frac{1}{A_{ii}}(b_i - \sum_{j<i} A_{ij}x_j^{(k+1)} - \sum_{j>i} A_{ij}x_j^{(k)}). \tag{2.40}$$

If this method is instead executed without sequentially forward substituting, it is the Jacobi method [56] and is written

$$x_i^{(k+1)} \leftarrow \frac{1}{A_{ii}}(b_i - \sum_{i\neq j} A_{ij}x_j^{(k)}). \tag{2.41}$$

Because the Jacobi method does not rely on the results of other rows, it is easily parallelized, however it converges much more slowly than GS, if at all.

It should be noted that there is a division in equation (2.40) by each diagonal element of $\mathbf{A}$. Indeed, for guaranteed convergence, $\mathbf{A}$ must be either symmetric positive-definite, or diagonally dominant. Unfortunately, formulations like those in equation (2.34) have zeros in the diagonals of their LCP portions, which make them poorly suited for standard Gauss-Seidel without modification. Further, the LCP definition requires that the solution $\mathbf{x}$ be non-negative.

2.5.1 Projected Gauss-Seidel, Splitting Method

Consider the "pure" LCP formulation, rewritten here

$$\begin{aligned} \mathbf{Ax} &\geq \mathbf{b} \\ \mathbf{x} &\geq \mathbf{0} \\ \mathbf{x}^T(\mathbf{Ax}+\mathbf{b}) &= \mathbf{0} \end{aligned} \tag{2.42}$$

One variation on Gauss-Seidel that attempts to solve the LCP is the so called splitting method [57, 58]. The idea is to split $\mathbf{A}$ into two different and more numerically convenient matrices. Using splitting, we break up $\mathbf{A}$ into $\mathbf{A} = \mathbf{M} - \mathbf{N}$. It follows

from equation (2.42) then

$$\begin{aligned} \mathbf{Mx} - \mathbf{Nx} + \mathbf{b} &\geq \mathbf{0} \\ \mathbf{x} &\geq \mathbf{0} \\ \mathbf{x}^T(\mathbf{Mx} - \mathbf{Nx} + \mathbf{b}) &= \mathbf{0} \end{aligned} \tag{2.43}$$

It can be shown that equation (2.43) is equivalent to finding

$$\min(\mathbf{x}^{k+1}, \mathbf{Mx}^{k+1} + \mathbf{b} - \mathbf{Nx}^k) = \mathbf{0}$$

which is easily rearanged by subtracting $\mathbf{x}^{k+1}$ and negating into the fixed-point formulation

$$\mathbf{x}^{k+1} = \max(0, -\mathbf{Mx}^{k+1} - \mathbf{b} + \mathbf{Nx}^k + \mathbf{x}^{k+1}) \tag{2.44}$$

Here, we can see (by trying both cases of $\mathbf{x}^{k+1} = \mathbf{0}$ and $\mathbf{x}^{k+1} \geq \mathbf{0}$) that $\mathbf{Mx}^{k+1} = -\mathbf{b} + \mathbf{Nx}^k$. So assuming that $\mathbf{M}$ was chosen well, it can be inverted and we can back substitute into equation (2.44) to get

$$\mathbf{x}^{k+1} = \max(\mathbf{0}, \mathbf{M}^{-1}(\mathbf{Nx}^k - \mathbf{b})) \tag{2.45}$$

From here, we obtain our fixed-point iterative method of Projected Gauss-Seidel (PGS) for solving LCPs of the form in equation (2.42):

Algorithm 2 Projected Gauss-Seidel using Splitting Method

Given matrix $\mathbf{A}$, vector $\mathbf{b}$, and initial guess $\mathbf{x}_0$
Determine matrices $\mathbf{M}$ and $\mathbf{N}$ where $\mathbf{A} = \mathbf{M} - \mathbf{N}$
Initalize $\mathbf{x}^k \leftarrow \mathbf{x}_0$
while not converged
    $\mathbf{x}^{k+1} \leftarrow -\mathbf{M}^{-1}(\mathbf{Nx}^k - \mathbf{b})$
    if $\mathbf{x}^{k+1} < \mathbf{0}$
        $\mathbf{x}^{k+1} \leftarrow \mathbf{0}$ *The projection step*
    $\mathbf{x}^k \leftarrow \mathbf{x}^{k+1}$

Of course, it remains to be determined what good choices are for $\mathbf{M}$ and $\mathbf{N}$ in algorithm 2, and how to find them.

2.5.2 Projected Succesive Over Relaxation

Projected Successive Over Relaxation (PSOR) is a more general form of PGS [59]. For the square matrix **A**, the iterative elemental form is a modified version of equation (2.40), and has the form

$$x_i^{(k+1)} \leftarrow max(0, x_i^{(k)} + \lambda \left[\tfrac{1}{A_{ii}} (b_i - \sum_{j<i} A_{ij} x_j^{(k+1)} - \sum_{j>i} A_{ij} x_j^{(k)}) - x_i^{(k)} \right]) \qquad (2.46)$$

where λ is a tuning parameter intended to improve convergence, and PGS is the special case where $\lambda = 1$.

We can also modify the Jacobi method to use a similar technique [60]. I have compared its performance against other methods and found in practice that a value of λ around 0.5 improves convergence for the problems tested. The idea is the same as when applied to GS: we use a tuning parameter λ in order to improve convergence. The iterative scheme I have used shares the same relationship with Jacobi that PSOR shares with GS and is given by

$$x_i^{(k+1)} \leftarrow max(0, x_i^{(k)} + \lambda \left[\tfrac{1}{A_{ii}} (b_i - \sum_{j \neq i} A_{ij} x_j^{(k)}) - x_i^{(k)} \right]) \qquad (2.47)$$

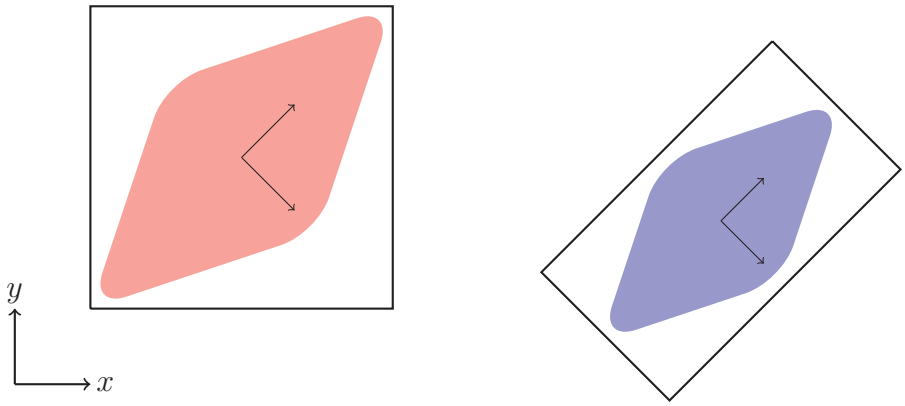
This is a projected Jacobi method with successive over-relaxation, and without projection onto zero is sometimes referred to as the modified Richardson iteration method.

2.5.3 Matrix Augmentation

Although shown to be useful for particular types of matrices, Gauss-Seidel requires non-zero diagonal elements. One simple idea for side-stepping this limitation is to add a constant value ϵ to all diagonal elements of A as a preprocessing step. Although it is not clear what a "good" value of ϵ is for a given matrix, I have experimented with problems in the RPI-MATLAB-Simulators HDF5 database and found that it takes unfortunately large values (*i.e.,* 0.5) to improve the convergence time. This is unfortunate of course because introducing ϵ also introduces an offset in the solution, impeding the effectiveness of the solver for converging to the correct solution, seemingly within the order of $\frac{\epsilon}{N}$.

2.6 Collision Detection

Collision detection is naturally an important step in simulation and can be extremely computationally expensive if not implemented well. Collision detection can be considered to occur in three phases: broad phase, mid phase, narrow phase. A major focus in collision detection research has been on broad phase collision detection [1], including spacial acceleration data structures such as uniform or hierarchical grids, trees, and sweep and prune methods [61, 62]. Additional tests using bounding volumes such as axis-aligned bounding boxes (AABB) or object-oriented bounding boxes (OBB) [63] are common. AABBs may be represented by a minimum and maximum value for each primary axis in the world frame. Alternatively, they may be represented by a center point of the bounding box along with three scalar magnitudes corresponding to the distance in each primary direction the box extends. OBBs may be represented in much the same way, but in terms of the corresponding body's primary axes.



(a) Axis aligned **(b) Object oriented**

Figure 2.7: The two common types of bounding boxes.

In the broad phase, spatial partitioning is used to determine which pairs of bodies should be considered. Being able to determine that two bodies are decidedly not is the first of many performance improvements since no more tests are necessary between those bodies. In the mid phase, there may be additional tests, such as the separating axis theorem, used for deciding if a body pair can be removed from consideration. In the final, and and most computationally expensive narrow phase,

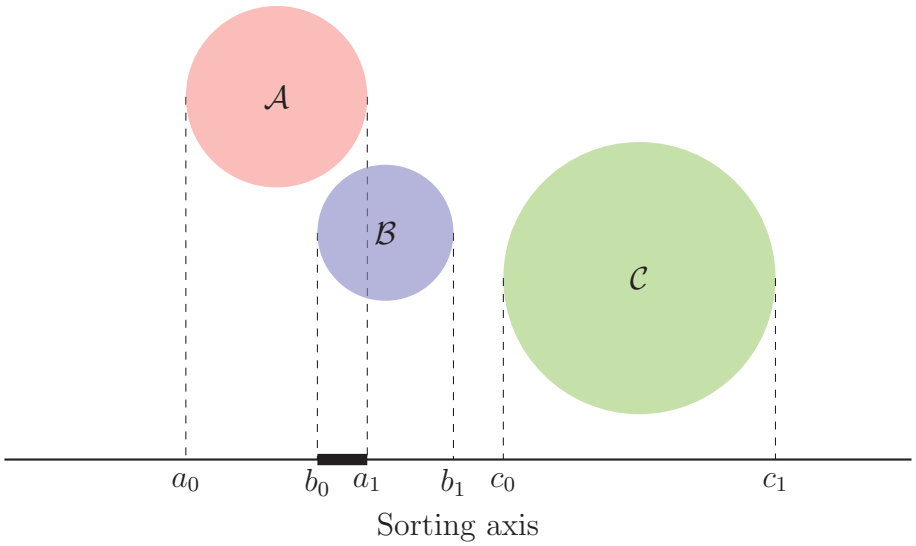


Figure 2.8: Depiction of the sort and sweep process. The body geometries, or possibly the corresponding bounding geometries, are projected onto an axis. Bodies that are found to have overlapping projections, *e.g.,* $\mathcal{A}$ and $\mathcal{B}$, will be considered at finer levels of collision detection.

contacts are determined and fully defined in terms of contact locations relative to each body and contact normal direction.

The sweep and prune method projects objects or the bounding volumes of objects onto a chosen axis, then checks which pairs of bodies have overlapping projections. In Figure 2.8, we see that the body pair $(\mathcal{A}, \mathcal{B})$ has an overlapping projection whereas $(\mathcal{A}, \mathcal{C})$ and $(\mathcal{B}, \mathcal{C})$ do not.

The sweep and prune method is an application of the separating axis theorem (SAT) or hyperplane separation theorem (HST). HST is a convexity-based method which states that given two convex sets $\mathcal{A}$ and $\mathcal{B}$, the sets are disjoint if and only if there exists a hyperplane between them that does not intersect $\mathcal{A}$ or $\mathcal{B}$.

Because simulation bottlenecks occur in the solve step, it is critical with regard to timing to minimize the size of the problem by limiting the number of contacts, ideally while avoiding non-physical results. A problem quickly arises however with choosing which contacts to include in the active set, and which to throw out. Including the "wrong" set of contacts by including too many results in over constraining (section 4.1). Not including enough contact results in interpenetrations which are

physically impossible and lead to instabilities.

A great collection of collision detection software is available online at `gamma.cs.unc.edu/software/`. The following are brief descriptions of some of the popular collision detection algorithms and libraries available.

GJK

The Gilbert-Johnson-Keerthi algorithm [64] has been very popular in video games and other real-time simulators, including the Bullet Physics Library [26]. Accelerated versions such as the so called "Enhanced GJK" from Cameron [65] greatly improve performance by exploiting mesh data structures. GJK finds the shortest distance between two complex polyhedra by iteratively walking along simplices of point subsets of two bodies until convergence on the nearest points. It does this using a support function that at each iteration determines the points that maximize the Minkowski Difference of the two simplices, then checks each body to see if there is an adjacent simplex that contains each point.

Unfortunately, GJK returns only a single nearest point by default, which is not necessarily a complete set between two bodies. Consider for example a cube sitting on a plane which requires at least three contacts to sit stably. Bullet uses an interesting approach in which they perturb one of the bodies about the contact normal and re-perform GJK to test for additional contacts. This is most useful when a face is in contact with another face. Finally, GJK returns accurate gap distances only for positive gaps.

RAPID

Robust and accurate polygon interference detection (RAPID) [63] is mainly a broad-phase algorithm that uses oriented bounding box trees (OBBTrees) to accelerate collision detection. In their original paper, Gottschalk et al. compare OOBTrees to axis aligned bounding boxes (AABBs) and claim fewer operations are needed for their test. However, they leave the definition of "operation" ambiguous! Further, their algorithm, although potentially useful for broad-phase, does not generate accurate contacts at the narrow-phase. I believe that AABBs are more easily

and commonly implemented in modern simulators. There is a memory-optimized version of RAPID called OPCODE.

V-Clip

Given two polyhedra in terms of their features, where a feature is any of vertex, edge, or face, V-Clip identifies a set of points such that each feature containing those points on each body lies within the Voronoi region of the other's feature [66]. It can be shown that when such a pair of features exist for two convex bodies, these are the nearest features between the bodies. V-Clip claims to be simpler and more robust than the Lin-Canny nearest features algorithm [67]. The claim of robustness stems from V-Clip's ability to deal well with penetrating bodies.

SWIFT and Lin-Canny

SWIFT utilizes accelerated spacial data structures or what they called "multi-level-of-detail representation" [68], as well as a Voronoi-based algorithm modified from the Lin-Canny collision detection in order to determine closest features. Unfortunately, like GJK, classic Lin-Canny performs poorly on overlapping polyhedra, which is practically guaranteed. The technique in SWIFT is called "Voronoi marching." Among its features, SWIFT claims to be faster than V-Clip, which claims to be faster than Enhanced GJK. Additionally, it claims to be more robust, which it justifies by having passed the so called "Algorithm 10" test from the original V-Clip paper.

Chapter summary

In this chapter, we were introduced to rigid body kinematics and dynamics, as well as approaches to collision detection. We saw how to represent rigid bodies in simulation, and how to formulate sets of dynamic constraints. Discrete time-stepping formulations were derived, including those for contact, friction, and joints. In particular, we utilized the Stewart-Trinkle time-stepping method. We also discussed solution methods for multibody dynamics problems, particularly the popular projected Gauss-Seidel (PGS) method. Lastly, we briefly introduced the

broad-phase, middle-phase, and narrow-phase in collision detection, and surveyed some of the influential and popular collision detection libraries available for multi-body simulation.

CHAPTER 3
DETECTING CONTACT BETWEEN MOVING GEOMETRIES

> "Practitioners, new to the field or otherwise, quickly discover that the attempt to build a fast, accurate, and robust collision detection system takes them down a long path fraught with perils and pitfalls unlike most they have ever encountered." - Gino van den Bergen

There are two major pitfalls common in collision detection which we will address in this chapter. The first is to treat discrete time simulations, evolving over each step, as though they were continuous or static systems. Such treatment leads to models that cannot prevent interpenetration at the end of the time step. Much of the robotics community approaches simulation using similar tools that they use in motion planning [69]. However, motion planning techniques do not necessarily deal with dynamic behavior, and are typically useful only when determining when interpenetration has already occurred. The second pitfall is more subtle, and regards how we choose the subset of *potential* contacts we keep for inclusion in the formulation of our dynamics. In the following sections, we will develop a set of geometric tools for precisely the purpose of choosing this contact set.

3.1 Representing convex polyhedra

We define a convex polyhedron in terms of its features. We may refer to a vertex either as an object v or as the vector $\mathbf{v}$ representing the vertex's position. Similarly, an edge object $e = (v_t, v_h)$ corresponds to a vector $\mathbf{e} = \mathbf{v}_h - \mathbf{v}_t$. A body is composed of a set vertices V, edges E, and triangle faces F defined by three vertices in counter-clockwise order. The normal vector for a given face $f = (v_i, v_j, v_k) \in F$ is given by

$$\hat{\boldsymbol{\eta}} = \frac{(\mathbf{v}_j - \mathbf{v}_i) \times (\mathbf{v}_k - \mathbf{v}_j)}{||(\mathbf{v}_j - \mathbf{v}_i) \times (\mathbf{v}_k - \mathbf{v}_j)||}. \tag{3.1}$$

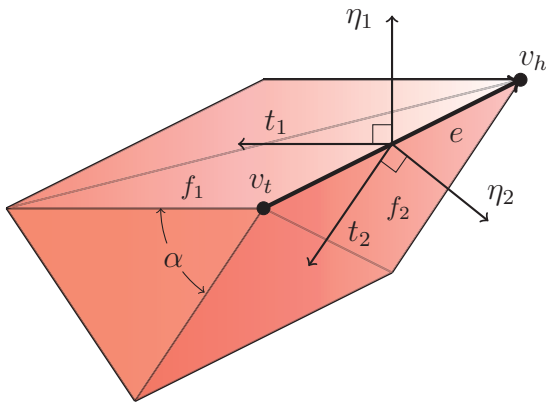


Figure 3.1: Geometric representation of a convex polyhedron. Vectors t_1 and t_2 are perpendicular to edge e and planar with their corresponding faces f_1 and f_2. The face normals η_1 and η_2 are normal with f_1 and f_2, as well as perpendicular to e.

The angle α between two faces joined by an edge is given by

$$\alpha = \pi - arccos(\hat{\boldsymbol{\eta}}_1 \cdot \hat{\boldsymbol{\eta}}_2),$$

and because we will make the assumption of convexity, it is necessarily the case that $\alpha \in (0, \pi]$.

Let us also define two vectors $\mathbf{t}_1$ and $\mathbf{t}_2$ for each edge that are planar with f_1 and f_2 respectively, and perpendicular to the vector $\mathbf{e} = \mathbf{v}_h - \mathbf{v}_t$. These vectors are defined as

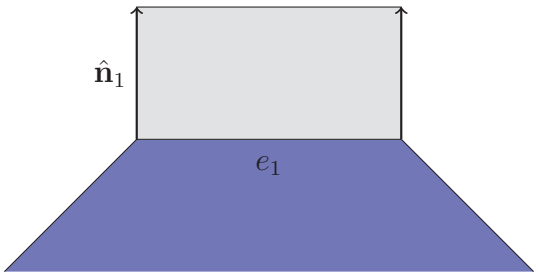
$$\begin{aligned} \mathbf{t}_1 &= \hat{\boldsymbol{\eta}}_1 \times \mathbf{e} \\ \mathbf{t}_2 &= \mathbf{e} \times \hat{\boldsymbol{\eta}}_2 \end{aligned} \tag{3.2}$$

and are useful for geometric tests when determining contacts.

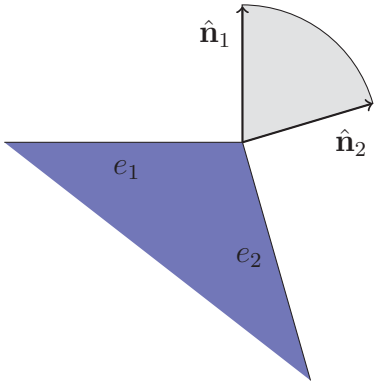
3.2 Applicability

The geometric test of applicability is well known [70–73] and commonly used in applications such as collision detection for motion planning. We geometrically relax the traditional notion of applicability in order to better determine which contacts to include. We have adopted notation similar to that of Latombe [71].

Figures 3.2 and 3.3 depict the possible positions and orientations of contact normal vectors for each feature in 2 and 3 dimensions. These normal vectors corre-



(a) 2D normal region of an edge.

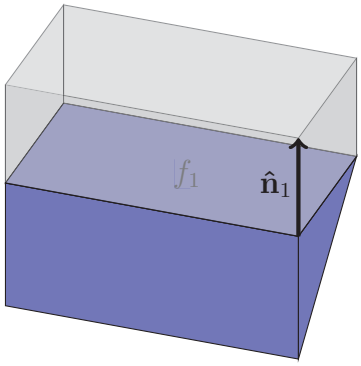


(b) 2D normal "cone" at a vertex. A contact at this vertex may result in a contact normal $\hat{\mathbf{n}}$ anywhere in this wedge.

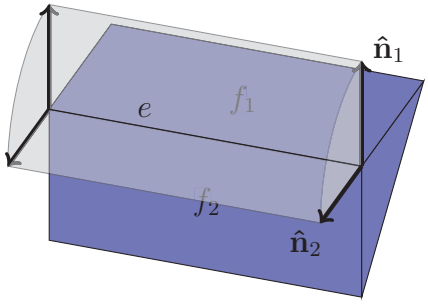
Figure 3.2: 2D normal regions for edge and vertex. These regions represent the possible directions in which a force could physically feasibly be enacted with the given feature.

spond to direction in which feasible contact forces could manifest at these surface regions. For example, realize that a flat edge such as that in 3.2a can only have a force that acts in the direction perpendicular to and outward from that edge. Although we could technically consider the vertices which define an edge as part of that edge, we distinguish between the two for clarity and convenience.

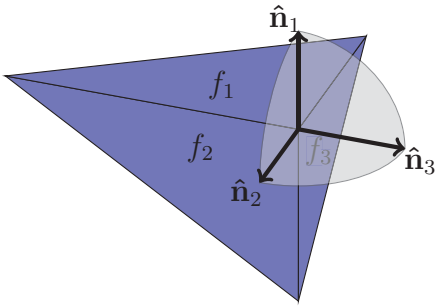
It is important to realize that when using a time-stepping method, we hope to identify contacts which will reflect these feasible contact normals at the *end* of the time step. Yet, many methods of contact identification use approaches that assume continuous or static body positions, resulting in inevitable interpenetrations due to the inability to foresee potential contacts. This is why we geometrically relax the following set of tests that will prove useful for identifying and making heuristic



(a) **3D normal region for a face. The simplest of the three 3D normal regions, it is defined by a single half space. The Voronoi region (depicted) is a subspace of that half space.**



(b) **3D normal region at an edge. Again the Voronoi region (pictured) is a subset of the normal region.**



(c) **3D normal "cone" at a vertex joining 3 faces. For the general case of m joined faces, the polyhedral region will be defined by m half spaces planar with the faces, and is equivalent to the Voronoi region for the vertex.**

Figure 3.3: 3D normal regions for face, edge, and vertex.

choices regarding potential contacts.

Consider the simple example in Figure 3.4 where two vertices, v_{a1} and v_{a2} of body $\mathcal{A}$ are near an edge e_b of body $\mathcal{B}$. In order to achieve stability in such a configuration, it is clear that both potential contacts $C_1 = \mathcal{C}(v_{a1}, e_b)$ and $C_1 = \mathcal{C}(v_{a2}, e_b)$ must be considered by the dynamics formulation. If only one were included, *e.g.,* C_1 then it is possible for the other to penetrate the half space defined by e_b given a large enough time step or velocity. We observe that the second contact normal n_2 is outside of the normal cone region of v_{a2}, does not correspond to a physically reasonable contact normal. However, this is because n_2 is determined at the beginning of the time step, yet we wish to prevent interpenetration for the end of the time step. In order to avoid interpenetrations, we must detect and consider such contacts as C_2, even when they are not physically feasible at the current time step. This necessity requires that we geometrically relax the normal cone region, or equivalently, relax the definition of applicability.

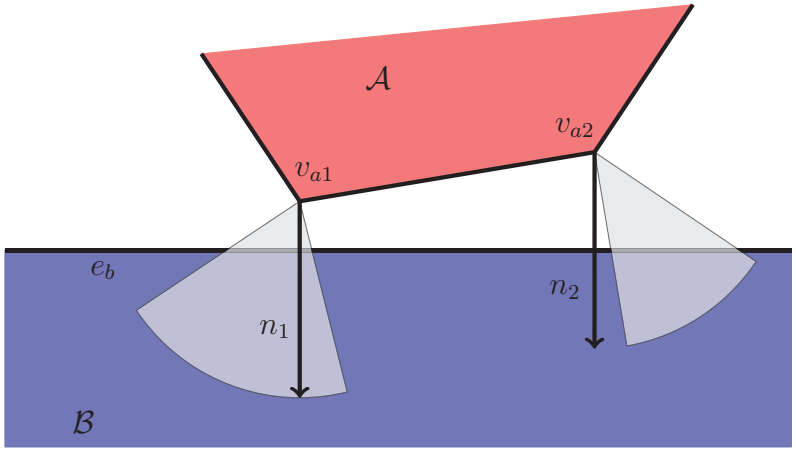


Figure 3.4: Two vertices of body $\mathcal{A}$ near an edge of $\mathcal{B}$. Vertex v_{a1} has classical applicability with e_b and the normal n_1 of $\mathcal{C}(v_{a1}, e_b)$ is within the normal cone. Vertex v_{a2} does not have classical applicability with e_b and the normal n_2 of $\mathcal{C}(v_{a2}, e_b)$ is outside its corresponding normal cone.

2D applicability

Let $\omega(v)$ be the set of all vertices connected by an edge to v. We refer to v_a having applicability with e_b when $APPL^{\{ve\}}(\mathbf{q}, v_a, e_b) \geq \epsilon_\theta$ for relaxation parameter

$$\epsilon_\theta = -\sin(\theta)$$

and where

$$APPL^{\{ve\}}(\mathbf{q}, v_a, e_b) = \min_{\mathbf{v}_k \in \omega(v_a)} \hat{\boldsymbol{\eta}}_b \cdot \frac{\mathbf{v}_k - \mathbf{v}_a}{||\mathbf{v}_k - \mathbf{v}_a||} \tag{3.3}$$

Whereas classical applicability is a boolean function, our function returns a value as we may wish to use the value of applicability as a heuristic when comparing contacts.

3D applicability

Figure 3.5 depicts a vertex v_a near a face f_b. Regardless of the proximity of v_a to f_b, it would be unreasonable to consider a potential force between these features if there existed a vertex connected to v_a that was significantly closer to f_b since we would expect this other vertex to make contact with f_b first.

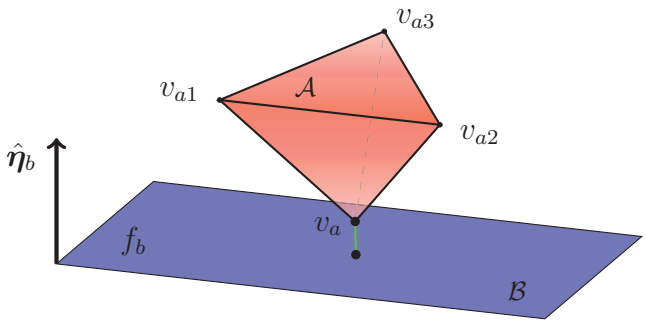


Figure 3.5: Vertex-face applicability tests all adjacent edges of a vertex v_a against a face normal $\hat{\boldsymbol{\eta}}$. In the configuration depicted here, v_a clearly has applicability with f_b since all adjacent edges of v_a are pointed "away" from f_b.

The 3D vertex-face case is analogous to the 2D vertex-edge case and is defined as

$$APPL^{\{vf\}}(\mathbf{q}, v_a, f_b) = \min_{\mathbf{v}_k \in \omega(v_a)} \hat{\boldsymbol{\eta}}_b \cdot \frac{\mathbf{v}_k - \mathbf{v}_a}{||\mathbf{v}_k - \mathbf{v}_a||} \tag{3.4}$$

In 2D, the applicability test will always be with two adjacent edges, whereas in 3D we may have an arbitrary number of edges greater or equal to three.

Figure 3.6 depicts two edges, e_a and e_b, in contact. Just as there are vertex-face configurations that have proximity but could not result in a force, so too are there edge-edge configurations that should not be associated with potential contact forces. First, we should note that given an edge-edge contact normal $\mathbf{n} = e_a \times e_b$, we cannot immediately say if we have the correct sign for $\mathbf{n}$ (this issue is addressed in 3.4). However, we can still determine edge-edge applicability in the following manner. Let

$$d_{a1} = \hat{\mathbf{n}} \cdot \hat{\mathbf{t}}_{a1}$$

$$d_{a2} = \hat{\mathbf{n}} \cdot \hat{\mathbf{t}}_{a2}$$

$$d_{b1} = -\hat{\mathbf{n}} \cdot \hat{\mathbf{t}}_{b1}$$

$$d_{b2} = -\hat{\mathbf{n}} \cdot \hat{\mathbf{t}}_{b2}$$

We then define edge-edge applicability as

$$\max\left(\min\left(d_{a1}, d_{a2}, d_{b1}, d_{b2}\right), \min\left(-d_{a1}, -d_{a2}, -d_{b1}, -d_{b2}\right)\right) \tag{3.5}$$

Conceptually similar to (3.4), (3.5) measures the maximal extent by which the adjacent faces of an edge are pointed "in" on another body.

3.3 Feasibility

Feasibility is similar to applicability, but is dependent on feature positions as opposed to orientations. The region of feasibility is essentially the region in which we anticipate a contact between a vertex v and a facet f could occur. We define vertex-edge and vertex-face feasibility similarly. Given an angle ϕ and a small value ϵ greater than machine precision, a vertex with contact $\mathcal{C}(v_a, f_b)$ has feasibility if

- $v_a \in VR(f)$ and $\psi > -\epsilon$, or
- $v_a \notin VR(f)$ and v_a within region defined by ϕ below the point ϵ below the nearest point on f,

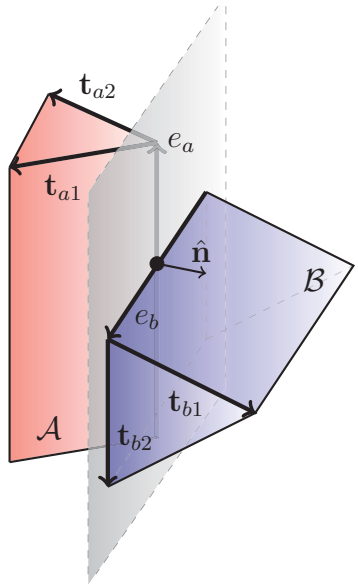


Figure 3.6: Edge-edge contact at the border of stability for classical applicability ($\epsilon_\theta^{\{ee\}} = 0$). The result in this configuration is dependent on machine precision error. Relaxation eliminates this dependency by increasing the domain of applicability.

where $VR(f)$ refers to the Voronoi region of the facet f [66]. An example of the feasibility region is depicted in Figure 3.7.

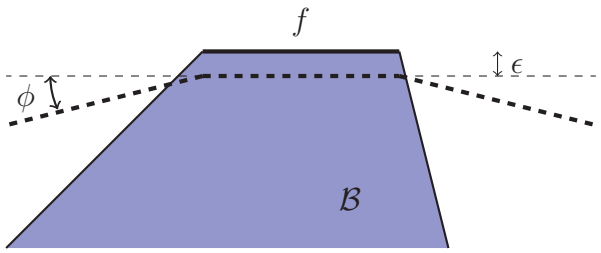
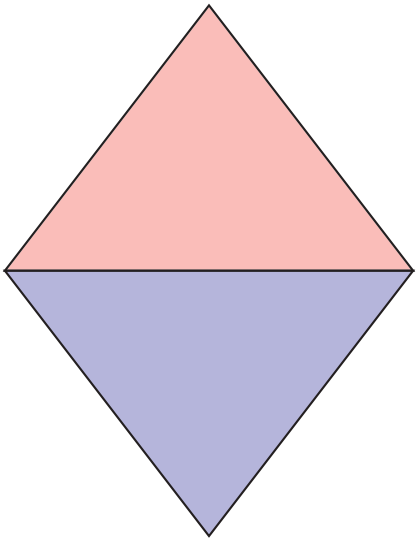


Figure 3.7: The region of feasibility for a vertex against a facet f of a body $\mathcal{B}$ is the region above the thick dashed line. A vertex in this region is considered to have a potential contact force λ with f.

The reason we allow for the distance ϵ below the face is that we must allow and even expect penetrations of at least machine precision. Additional penetration is even possible when considering rotation (see section 5.3). Consider Figure 3.8 where a dynamic red triangular body is sitting on top of, and horizontally aligned



(a) A dynamic red triangle sits atop a static blue triangle. The two triangles are similar and aligned horizontally.

(b) A close up view (exaggerated) of the nearest edges of the two bodies, where they are in contact.

Figure 3.8: Our body representation uses vertices, with edges and faces in terms of these vertices. Due to machine error, solver error, etc., the location of these vertices can only be traced and known with some ϵ. This is especially true in cases in which are are concerned about it most, when bodies are in contact.

with, a similar but static blue triangle. In 3.8b, we see an example a configuration that is invalid, but within ϵ of a valid configuration. Configurations such as this are what require the ϵ relaxation in feasibility.

To further impart the dangers of making assumptions regarding vertex positions, consider that the example of Figure 3.8 is ignoring the horizontal component of errors, and is a relatively clean example in that the left and right vertices of the red body will not necessarily have symmetrical errors. These errors, and the

subsequent lack of assumptions available regarding contact, are more evidence that there exists no heuristic approach to choosing a contact subset to enforce.

3.4 Edge-edge orientation

Vertex-edge and vertex-face contact are always determined by the corresponding edge or face normal that the vertex is colliding with. With edge-edge contact we must deal with the issue of the normal direction given by $\mathbf{e}_a \times \mathbf{e}_b$ having arbitrary sign. We do this using the notion of orientation of edges.

Consider an edge e_b of body $\mathcal{B}$. We define an orientation vector $\mathbf{O}_{e_b} = \mathbf{e}_b \times -(\hat{\boldsymbol{\eta}}_1 + \hat{\boldsymbol{\eta}}_2)$.

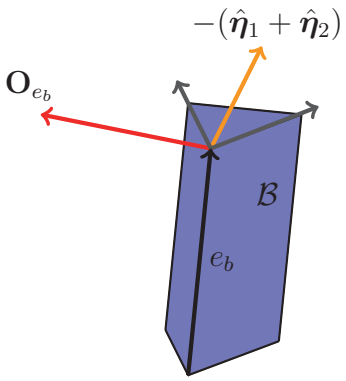
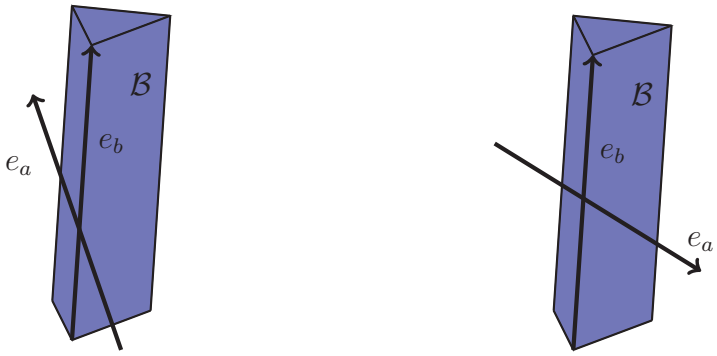


Figure 3.9: The orientation vector $\mathbf{O}_{e_b}$ is defined in such a way as to always point "left" of edge $\mathbf{e}_b$ when looking onto the edge from the outside of the body.



$\mathbf{e}_a \cdot \mathbf{O}_{e_b} > 0$
$\mathbf{e}_a$ **is rotated CCW w.r.t.** $\mathbf{e}_b$

$\mathbf{e}_a \cdot \mathbf{O}_{e_b} < 0$
$\mathbf{e}_a$ **is rotated CW w.r.t.** $\mathbf{e}_b$

Figure 3.10: Two different configurations of edge-edge contacts.

Knowing the orientation of edge-edge contact allows us to determine the correctly signed contact normal

$$\hat{\mathbf{n}} = \begin{cases} \dfrac{\mathbf{e}_a \times \mathbf{e}_b}{||\mathbf{e}_a \times \mathbf{e}_b||} & \text{if } \mathbf{e}_a \cdot \mathbf{O}_{e_b} > 0 \\ \\ \dfrac{\mathbf{e}_b \times \mathbf{e}_a}{||\mathbf{e}_b \times \mathbf{e}_a||} & \text{if } \mathbf{e}_a \cdot \mathbf{O}_{e_b} < 0 \end{cases} \tag{3.6}$$

In the special case that the edges have the same direction, that is, $\mathbf{e}_a \cdot \mathbf{O}_{e_b} = 0$, we are then in a fortunate position of not needing to include this edge-edge contact directly. Instead, we rely on the vertex-face contacts to prevent interpenetration.

3.5 2D narrow phase collision detection

Of particular interest when performing 2D collision detection are the distances and nearest points between either a point and a line, or a point and a segment. This is due to the fact that we are concerned with vertices penetrating edges. Given an edge composed of vertices a and b which define a line a line $\overleftrightarrow{\mathbf{ab}}$ where $\mathbf{a} \neq \mathbf{b}$, there is a unit normal vector $\hat{\mathbf{n}}$ defined as perpendicular to $\overleftrightarrow{\mathbf{ab}}$ and pointing "out" of the body. Given the convention in 2D that the vertices are listed in counter-clockwise order for each polygon, we may determine the normal vector $\hat{\mathbf{n}}$ by rotating the edge vector by $-\pi/2$ with

$$\hat{\mathbf{n}} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \frac{\mathbf{b} - \mathbf{a}}{||\mathbf{b} - \mathbf{a}||}$$

The signed gap distance ψ of a point $\mathbf{c}$ to the line is given by

$$\psi = (\mathbf{c} - \mathbf{a}) \cdot \hat{\mathbf{n}} \tag{3.7}$$

where we could replace $\mathbf{a}$ with $\mathbf{b}$ and obtain the same result. The closest point $\mathbf{d}$ on the line (the closest point on the segment is trickier) is given by

$$\mathbf{d} = \mathbf{c} - \psi\hat{\mathbf{n}} \tag{3.8}$$

Equation (3.7) is useful for determining the gap distance, but may be unnecessary in cases where $\mathbf{c}$ is a distance greater than some ϵ away from the segment $\overline{\mathbf{ab}}$. The nearest point on a segment $\overline{\mathbf{ab}}$ to a point $\mathbf{c}$, as well as the distance from $\mathbf{c}$ to the segment, are given by Algorithm 3 [1].

Algorithm 3 Point-segment nearest point, for both 2D and 3D.

Given $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$, determines nearest point $\mathbf{d}$ on $\overline{\mathbf{ab}}$ to $\mathbf{c}$.
function POINT-SEGMENT($\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$)
  $\mathbf{e} = \frac{\mathbf{b}-\mathbf{a}}{||\mathbf{b}-\mathbf{a}||}$
  $t = \frac{\mathbf{c}-\mathbf{a}}{\mathbf{e}\cdot\mathbf{e}}$
  if $t < 0$ **then**
    $t = 0$
  else if $t > 1$ **then**
    $t = 1$
  end if
  $\mathbf{d} = \mathbf{a} + t\mathbf{e}$
end function

We observe that Algorithm 3 contains a "clamping" function in that it projects t onto the segment $(0, 1)$. As this function is useful in following algorithms as well, let us define it as a function as in Algorithm 4.

Algorithm 4 Clamp value to segment.

function CLAMP(t,t_{min},t_{max})
  if $t < t_{min}$ **then**
    $t = t_{min}$
  else if $t > t_{max}$ **then**
    $t = t_{max}$
  end if
end function

Figure 3.11 depicts a vertex v_a within ϵ of the edges e_{b1} and e_{b2}. The vertex identified as v_b is the nearest point on both edges to v_a. We note that v_a has applicability with e_{b2} but not e_{b1}. Moreover, v_a has feasibility with both edges. It is clear that the contact $\mathcal{C}(v_a, e_{b1})$ could not result in a contact force (given reasonable time step size and velocities), while the contact $\mathcal{C}(v_a, e_{b2})$ could result in a contact force.

We will see in chapter 4 how such contacts are utilized in the formulation of a time-stepping subproblem. Further, we will analyze what heuristics we can use

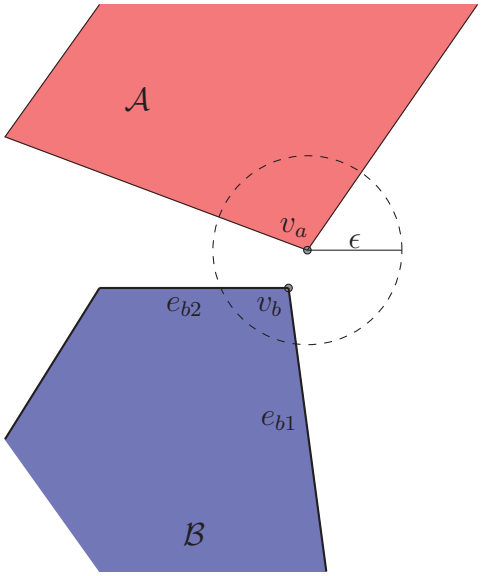


Figure 3.11: 2D vertex-edge contact identification. All edges within ϵ of v_a will be considered as potential vertex-edge contacts.

to improve performance by reducing the number of these contacts or the number of contacts which may potentially result in contact forces.

Algorithm 5 represents a brute force approach to vertex-edge collision detection, however it should be noted that there are optimizations which exploit spacial and temporal coherence, as well as geometric convexity, to drastically reduce computational complexity to be effectively constant. We denote a *primary* contact as

Algorithm 5 2D contact identification.

Given polygons $\mathcal{A}$ and $\mathcal{B}$, a distance ϵ, and relaxation ϵ_θ
for $v_a \in V_\mathcal{A}$ **do**
  for $e_b = \{v_{b1}, v_{b2}\} \in E_\mathcal{B}$ **do**
    $\psi_n = (\mathbf{v}_a - \mathbf{v}_{b1}) \cdot \hat{\boldsymbol{\eta}}_b$
    $\mathbf{p}_b = POINT\text{-}SEGMENT(v_{b1}, v_{b2}, v_a)$
    if $||\mathbf{v}_a - \mathbf{p}_b|| \leq \epsilon$ **then**
      if $APPL^{\{ve\}}(v_a, e_b) \geq \epsilon_\theta \bigwedge FEAS^{\{ve\}}(v_a, e_b)$ **then**
        Add primary contact $\mathcal{C}(v_a, e_b)$
      else
        Add secondary contact $\mathcal{C}(v_a, e_b)$
      end if
    end if
  end for
end for

one which could feasibly result in a force. A *secondary* contact is one that, although it could not reasonably be associated with a contact force, is necessary to include in order to accurately represent the local geometry near the point of contact. The distinction of a contact's applicability and feasibility is important because a contact can only be considered as primary if it has both applicability and feasibility. However, the contact must still be considered if it is within ϵ and regardless of its applicability and feasibility since satisfaction of a constraint which includes this contact may be necessary in order to achieve geometrically accurate interactions of bodies.

3.6 3D narrow phase collision detection

3D collision detection is quite different than 2D. Even reducing our subsets of possible considered bodies to be closed complete convex polyhedra, adding the single spacial dimension drastically increases the potential complexities. For example, in 2D all vertices connect exactly two edges, while in 3D all vertices connect at least three edges, but may connect arbitrarily many more.

3.6.1 Vertex-face

The vertex-face process in 3D is similar to the vertex-edge algorithm in 2D. A vertex is considered for potential collision with a face when it is within a distance ϵ of that face, which in 3D manifests as an ϵ sphere. Consider Figure 3.12, and in particular the vertex v_a near multiple faces of body $\mathcal{B}$. Vertex v_a has three potential contacts with the three faces of $\mathcal{B}$: $C_1 = \mathcal{C}(v_a, f_{b1})$, $C_2 = \mathcal{C}(v_a, f_{b2})$, and $C_3 = \mathcal{C}(v_a, f_{b3})$. In the configuration shown, it happens that v_a has applicability with all of these faces. Moreover, it has feasibility with all three faces. Certainly all three contacts would need to be considered in the formulation of a dynamics problem in order to guarantee geometrically accurate interaction, but moreover because all three contacts have applicability and feasibility, they are all candidates to have a contact force associated with them.

This set of vertex-face contacts begins to hint at a question of which contacts should be included at the stage of formulating dynamics. If we were to include all

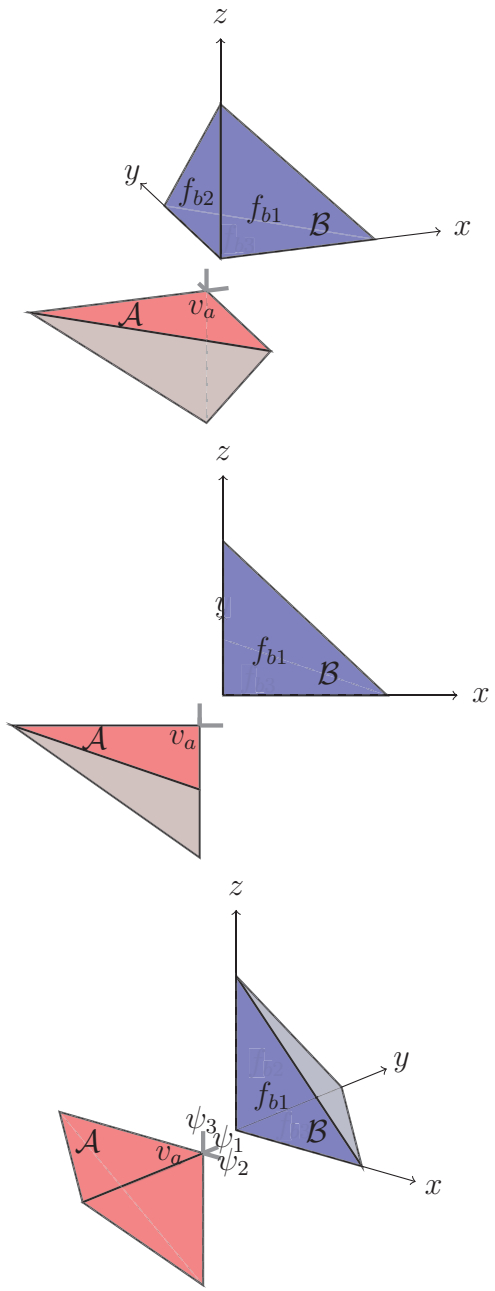


Figure 3.12: Multiple views of a vertex v_a near multiple faces. The gap distances ψ_1, ψ_2, and ψ_3 correspond to the three possible vertex-face contacts between v_a with body $\mathcal{B}$. Physically, only one of these contacts should be active at one time.

of these contacts and enforce unilateral constraints on each of them, we would be trapping v_a against the corner of $\mathcal{B}$. This question is precisely the topic of Chapter 4, and we will see there how we can include all primary and secondary contacts but

still allow the accurate passing of bodies through free space.

A high level approach to vertex-face contact identification is given in Algorithm 6. I have not defined the function $POINT\text{-}TRIANGLE\text{-}3D$, but it identifies a point on a triangle in $\mathbb{R}^3$ that is nearest to a given point.

Algorithm 6 3D vertex-face contact identification.

Given polyhedra $\mathcal{A}$ and $\mathcal{B}$, a distance ϵ
for $v_a \in V_{\mathcal{A}}$ **do**
  for $f_b = \{v_{b1}, v_{b2}, v_{b3}\} \in F_{\mathcal{B}}$ **do**
    $\psi_n = (\mathbf{v}_a - \mathbf{v}_{b1}) \cdot \hat{\boldsymbol{\eta}}_b$
    $\mathbf{p}_b = POINT\text{-}TRIANGLE\text{-}3D(v_{b1}, v_{b2}, v_{b3}, v_a)$
    if $||\mathbf{v}_a - \mathbf{p}_b|| \leq \epsilon$ **then**
      if $APPL^{\{vf\}}(v_a, f_b) \geq \epsilon_\theta \bigwedge FEAS^{\{vf\}}(v_a, f_b)$ **then**
        Add primary contact $\mathcal{C}(v_a, f_b)$
      else
        Add secondary contact $\mathcal{C}(v_a, f_b)$
      end if
    end if
  end for
end for

3.6.2 Edge-edge

An important part of edge-edge contact identification is determining the nearest points between two segments in 3D. This procedure is detailed in Algorithm 7. Although this algorithm is known and available from other sources such as [1], it is fundamental to collision detection in 3D and I have therefore chosen to include it for completeness. Noting that an edge is defined by two vertices, we observe that the case of a vertex near an edge (vertex-edge) is a subset of edge-edge detection. This observation may be useful during a mid-phase collision detection step since in 4.4.3 we will treat the vertex-edge configuration as a special case which requires a special set of constraints. Further, edges which are found to be within ϵ of another body, but not within ϵ of an edge of that other body, will be found to have vertex-face contact. This observation can be used to decrease computational expense in determining the complete set of contacts between two bodies.

Algorithm 7 Segment-segment nearest points and distance in 3D.

Given $\mathbf{p}_1$, $\mathbf{q}_1$, $\mathbf{p_2}$, $\mathbf{q}_2$, determines nearest points $\mathbf{c}_1$ on $\overline{\mathbf{p}_1\mathbf{q}_1}$ and $\mathbf{c}_2$ on $\overline{\mathbf{p}_2\mathbf{q}_2}$.

function SEGMENT-SEGMENT-3D($\mathbf{p}_1$, $\mathbf{q}_1$, $\mathbf{p}_2$, $\mathbf{q}_2$)

$\mathbf{d}_1 = \mathbf{q}_1 - \mathbf{p}_1$
$\mathbf{d}_2 = \mathbf{q}_2 - \mathbf{p}_2$
$\mathbf{r} = \mathbf{p}_1 - \mathbf{p}_2$
$a = \mathbf{d}_1 \cdot \mathbf{d}_1$
$e = \mathbf{d}_2 \cdot \mathbf{d}_2$
$f = \mathbf{d}_2 \cdot \mathbf{r}$
if $a \leq 0 \wedge e \leq 0$ **then**
$s = t = 0$
$\mathbf{c}_1 = \mathbf{p}_1$
$\mathbf{c}_2 = \mathbf{p}_2$
$dist = ||\mathbf{c}_1 - \mathbf{c}_2||$
return
end if
if $a \leq 0$ **then**
$s = 0$
$t = \frac{f}{e}$
$t = CLAMP(t, 0, 1)$
else
$b = \mathbf{d}_1 \cdot \mathbf{d}_2$
if $ae - bb \neq 0$ **then**
$s = CLAMP(\frac{bf-ce}{ae-bb}, 0, 1)$
else
$s = 0$
end if
$t = \frac{bs+f}{e}$
if $t < 0$ **then**
$t = 0$
$s = CLAMP(\frac{-c}{a}, 0, 1)$
else if $t > 1$ **then**
$t = 1$
$s = CLAMP(\frac{b-c}{a}, 0, 1)$
end if
end if
$\mathbf{c}_1 = \mathbf{p}_1 + \mathbf{d}_1 s$
$\mathbf{c}_2 = \mathbf{p}_2 + \mathbf{d}_2 t$
$dist = ||\mathbf{c}_1 - \mathbf{c}_2||$

end function

Chapter summary

In this chapter, we defined our representation of polytopes and developed two important geometric tests, namely applicability and feasibility, for determining the set of potential contacts between pairs of bodies. In the case of applicability, we applied a geometric relaxation to a classical definition in order to better identify contacts which are not currently active, but could become active within the time step. Details were given on the nature of some errors which can manifest in computer simulation, providing motivation for and insight into the computational geometry tools we were developing. We were then presented with a reliable way of determining the sign of the contact normal for edge-edge contact, resolving the ambiguity that can arise when taking the cross product of two edge vectors. Finally, we looked at algorithms, which utilize the geometric tools we developed, for choosing sets of contacts between polytopes.

CHAPTER 4
POLYTOPE EXACT GEOMETRY

The standard approach to modeling contact between two bodies is to choose a set of contacts between them (possibly limited to one) and to model all of these contacts as unilateral constraints. In this chapter, we will begin to elucidate the problems imposed by this classical representation of contact, and derive an alternative contact model. This new model is based on a mathematical framework which allows us to incorporate the complete set of possible contacts in order to accurately capture the body geometries at those points of potential contact, fully representing the physically possible outcomes. We will see how to apply this new model to convex bodies in 2D as well as 3D, and how incorporate this model into the Stewart-Trinkle time-stepping method.

4.1 Motivation and a new approach

One approach to multibody simulation is to allow interpenetrations (and other constraint violations) to occur, and then to apply *ad hoc* corrections as necessary to catch the poorly defined geometric configurations that inevitably result. This approach, which includes penalty methods, is notoriously prone to instabilities and oscillatory behavior. The primary approach to dealing with these instabilities is to use a very small time step size. However, this often merely lessens the apparentness of instabilities, and does not address their cause. Our approach attempts to prevent non-physical configurations from occurring. In order to achieve this we must detect potential contacts before interpenetration occurs.

Contact constraints have been traditionally modeled as fundamentally unilateral, however, representing features as infinite half-spaces at points of potential contact generates non-physical behavior at corners where multiple finite features meet and terminate. Consider a vertex v near a corner as depicted in Figure 4.1. When the collision detection routine identifies both $C_1 = \mathcal{C}(v, e_1)$ and $C_2 = \mathcal{C}(v, e_2)$, the vertex becomes erroneously trapped in the convex subspace of the free space

surrounding the corner if both contacts are enforced. This is a known consequence in the standard model, and we therefore refer to it as the *standard-model trap* or SM trap. Most often, heuristics based on gap distances or body properties are em-

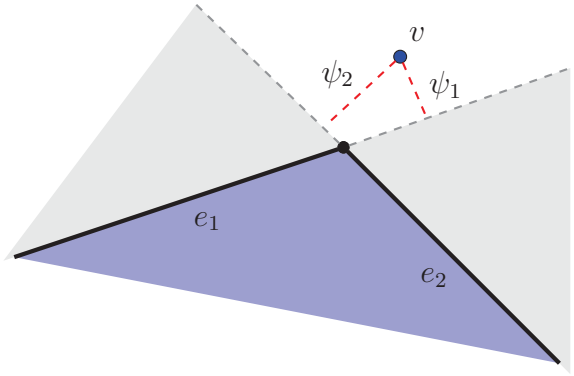


Figure 4.1: The standard-model trap. A vertex v approaching two edges e_1 and e_2 at a corner of a body. If unilateral constraints are enforced against both edges, the vertex becomes trapped by the sum of the edges' half-spaces (gray regions plus body). The same trap can occur in 3D between a vertex and set of faces or an edge against multiple edges.

ployed in order to make a guess about a subset of these potential contacts to include. However, such heuristics are easily broken, and in some cases incorporate additional instabilities.

Deriving a new model

Let us begin by solving the SM trap in 2D for a single vertex v approaching a corner composed of two edges e_1 and e_2, similarly to as was done by Nguyen [74]. We identify two potential contacts $C_1 = \mathcal{C}(v_a, e_1)$ and $C_1 = \mathcal{C}(v_a, e_2)$ with corresponding gap distances ψ_1 and ψ_2 for example as in Figure 4.1. We wish to enforce that

$$\max(\psi_1, \psi_2) \geq 0, \tag{4.1}$$

allowing v to penetrate either half space corresponding to e_1 or e_2, but not both.

Lemma 4.1.1

Given $a, b \in \mathbb{R}, \max(a, b) = a + \max(b - a, 0) \geq 0$

Proof:

Case $a \geq b \implies a + \max(b - a, 0) = a + 0 = a$

Case $a \leq b \implies a + \max(b - a, 0) = a + (b - a) = b$ □

Lemma 4.1.2

Given $a, b \in \mathbb{R}, b = \max(a, 0) \iff 0 \leq b - a \perp b \geq 0$

Proof:

Case $a \leq 0$*:* $\max(a, 0) = 0 \implies b = 0 \implies 0 \leq b - a \perp b \geq 0$

Case $a > 0$*:* $\max(a, 0) = a \implies b = a \implies 0 \leq b - a \perp b \geq 0$

Case $b - a = 0, b \geq 0$*:* $b = \max(a, 0) = a$

Case $b = 0, b - a \geq 0$*:* $a \leq 0 \implies \max(a, 0) = 0 = b$ □

Using Lemma 4.1.1, we rewrite and constrain equation (4.1) as

$$\max(\psi_1, \psi_2) = \psi_1 + \max(\psi_2 - \psi_1, 0) \geq 0$$

Then using Lemma 4.1.2 and letting $c = \max(\psi_2 - \psi_1, 0)$, we can constrain $\max(\psi_1, \psi_2) \geq 0$ with

$$\begin{array}{rcrcl} 0 & \leq & c + \psi_1 - \psi_2 \perp c & \geq & 0 \\ & & c + \psi_1 & \geq & 0 \end{array} \tag{4.2}$$

To give some intuition regarding c and its relationship with ψ_1 and ψ_2, consider the figure below and the value of c on each side of the bisector where $\psi_1 = \psi_2$. A vertex

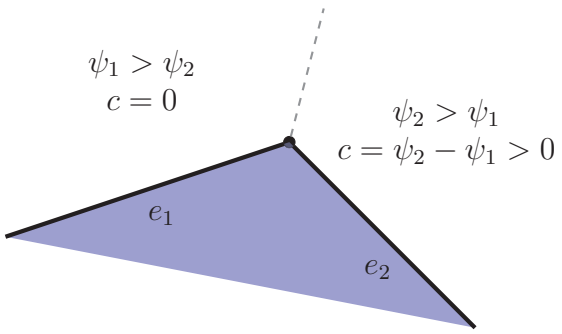


Figure 4.2: The possible values for c in the space surrounding edges e_1 and e_2. The regions where c is zero and where $c = \psi_2 - \psi_1$ correspond to the Voronoi regions of the edges of the body.

in the free space left of the bisector has $\psi_1 > \psi_2 \implies \psi_1 - \psi_2 > 0$ which requires

$c = 0$ to satisfy the complementarity condition of (4.2). A vertex to the right of the bisector has $\psi_1 < \psi_2 \implies \psi_1 - \psi_2 < 0$ which requires $c = \psi_2 - \psi_1$ in order to satisfy the complementarity condition. In both cases, satisfaction of the constraints in (4.2) requires that at least one gap distance be non-negative.

We now wish to include potential forces λ_1 and λ_2 while ensuring that only one is non-zero since allowing both to be simultaneously non-zero would be non-physical.

Lemma 4.1.3

Given $a, b \in \mathbb{R}$, $b = |min(a, 0)| \iff 0 \leq b + a \perp b \geq 0$

Proof:

Case $a \leq 0$: $|min(a, 0)| = -a = b \implies b + a = 0 \implies 0 \leq b + a \perp b \geq 0$

Case $a > 0$: $|min(a, 0)| = 0 = b \implies (a + b)b = 0 \implies 0 \leq b + a \perp b \geq 0$

Case $b + a = 0, b \geq 0$: $a \leq 0 \implies |min(a, 0)| = -a = b$

Case $b = 0, b + a \geq 0$: $a > 0 \implies |min(a, 0)| = 0 = b$ □

Lemma 4.1.4

Given $a, b \in \mathbb{R}$, $a = 0, b \leq 0 \iff (max(a, b) \geq 0) \wedge (max(a, b) + |min(a, 0)| = 0)$

Proof:

The only way to satisfy both $max(a, b) \geq 0$ *and* $|min(a, 0)| \geq 0$ *is for both to reach equality. When* $|min(a, 0)| = 0$, *it is possible that* $a \geq 0$, *so in order to have* $max(a, b) = 0$ *we must have* $a = 0, b \leq 0$. *The argument for the reverse of* a *and* b *is similar.* □

Using Lemma 4.1.3, we write

$$\begin{array}{rcrcrcl} 0 & \leq & d_1 + \psi_1 & \perp & d_1 & \geq & 0 \\ 0 & \leq & d_2 + \psi_2 & \perp & d_2 & \geq & 0 \end{array} \tag{4.3}$$

where d_1 and d_2 are slack variables that will equal the magnitude of the gap distance when in penetration and zero otherwise. Using Lemma 4.1.4, we have

$$\begin{array}{rcrcrcl} 0 & \leq & c + \psi_1 + d_1 & \perp & \lambda_1 & \geq & 0 \\ 0 & \leq & c + \psi_1 + d_2 & \perp & \lambda_2 & \geq & 0 \end{array} \tag{4.4}$$

which permits at most one of λ_1 or λ_2 to be non-zero. At this point, equations (4.2), (4.3), and (4.4) correspond to the locally non-convex (LNC) model proposed by Nguyen [74] which accurately models the case for a single vertex against multiple edges. Although this model has been shown to contribute some improvement over other methods [75], stopping here would correct the SM trap at the expense of permitting inter-penetrations [76]. To see this, let us continue with a slightly more complex case.

Consider two bodies approaching as depicted in Figure 4.3. In this configu-

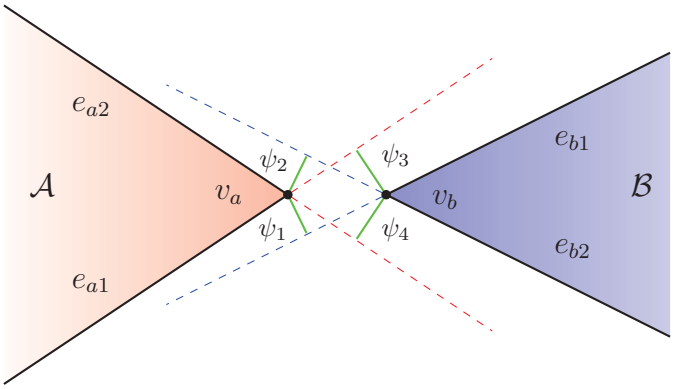


Figure 4.3: A dual case of a vertex near two edges in 2D. The physically feasible trajectories of v_a are interdependent with the trajectories of v_b. Consider that if v_a goes "on top" of $\mathcal{B}$, then v_b cannot simultaneously go "on top" of $\mathcal{A}$.

ration, our collision detection routine identifies four potential contacts. These are $C_1 = \mathcal{C}(v_a, e_{b1})$, $C_2 = \mathcal{C}(v_a, e_{b2})$, $C_3 = \mathcal{C}(v_b, e_{a1})$, and $C_4 = \mathcal{C}(v_b, e_{a2})$. We cannot simply constrain all of these contacts, for this would result in a dual SM trap. It is possible to make a guess as to which contacts to enforce, but this inserts the likely possibility of generating contact forces where there are none. Of course we could also ignore all four contacts and allow interpenetration, but this too is clearly a non-physical approach.

Instead, let us write constraints including all four potential contacts in such a way as to allow all physically feasible interactions while preventing interpenetration. We start by observing that C_1 and C_2 should both be able to be violated, but not simultaneously. Similarly, C_3 and C_4 should not be simultaneously violated. We

have seen how these constraints can be represented with

$$
\begin{array}{rcccccl}
0 & \leq & c_1 + \psi_1 - \psi_2 & \perp & c_1 & \geq & 0 \\
0 & \leq & c_2 + \psi_4 - \psi_3 & \perp & c_2 & \geq & 0 \\
 & & c_1 + \psi_1 & & & \geq & 0 \\
 & & c_2 + \psi_4 & & & \geq & 0 \\
0 & \leq & d_1 + \psi_1 & \perp & d_1 & \geq & 0 \\
0 & \leq & d_2 + \psi_2 & \perp & d_2 & \geq & 0 \\
0 & \leq & d_3 + \psi_3 & \perp & d_3 & \geq & 0 \\
0 & \leq & d_4 + \psi_4 & \perp & d_4 & \geq & 0 \\
0 & \leq & c_1 + \psi_1 + d_1 & \perp & \lambda_1 & \geq & 0 \\
0 & \leq & c_1 + \psi_2 + d_2 & \perp & \lambda_2 & \geq & 0 \\
0 & \leq & c_2 + \psi_3 + d_3 & \perp & \lambda_3 & \geq & 0 \\
0 & \leq & c_2 + \psi_4 + d_4 & \perp & \lambda_4 & \geq & 0
\end{array}
\tag{4.5}
$$

but we make the important observation that this model erroneously allows for ψ_1 and ψ_4 or ψ_2 and ψ_3 to be simultaneously violated. An example of such a consequence is depicted in Figure 4.4. The model of equation (4.5) can be completed by including

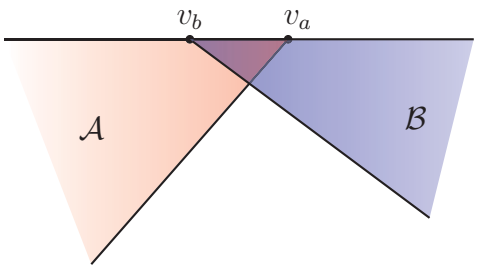


Figure 4.4: A valid solution to the dual vertex-edge case using the LNC model. Note that neither vertex is penetrating the other body, satisfying the constraints of equation (4.5).

constraints to prevent such inter-penetrations. That is, we additionally constrain

$\max(\psi_1, \psi_4) \geq 0$ and $\max(\psi_2, \psi_3) \geq 0$ with

$$\begin{array}{rcccccl} 0 & \leq & c_3 + \psi_1 - \psi_4 & \perp & c_3 & \geq & 0 \\ 0 & \leq & c_4 + \psi_2 - \psi_3 & \perp & c_4 & \geq & 0 \\ & & c_3 + \psi_1 & & & \geq & 0 \\ & & c_4 + \psi_2 & & & \geq & 0 \end{array} \tag{4.6}$$

Equations (4.5) and (4.6) together represent our first example of a geometrically accurate contact model. We will refer to the constraint type in equation (4.5) as *inter-contact* constraint, and the type in equation (4.6) as *cross-contact* constraint. These, along with unilateral constraint, form the set of three contact constraints necessary for our model.

4.2 Three fundamental contact constraints

In order to resolve the issues of interdependency among contacts discussed in the previous section, we had to change the way we formulate contact constraints. In this section, we define three fundamental contact constraints in their general form and begin to describe how they can be applied.

4.2.1 Unilateral contact constraint

The simplest of the three contact constraints is the common unilateral constraint ($\mathcal{U}$-constraint) that was introduced by equation (2.14). We define a $\mathcal{U}$-constraint as a function of a single contact C_i and write it as

$$\mathcal{U}(C_i) \tag{4.7}$$

which corresponds to the complementarity problem

$$0 \leq \psi_i \perp \lambda_i \geq 0 \tag{4.8}$$

where λ_i is the force corresponding to contact C_i.

4.2.2 Inter-contact constraint

An inter-contact constraint ($\mathcal{I}$-constraint) is that which was demonstrated in equation (4.5) for preventing the SM trap. An $\mathcal{I}$-constraint deals with contacts of a single feature of one body against multiple features of another body, as for example in Figure 4.5. Here we will define the general formulation of $\mathcal{I}$-constraint for a vertex

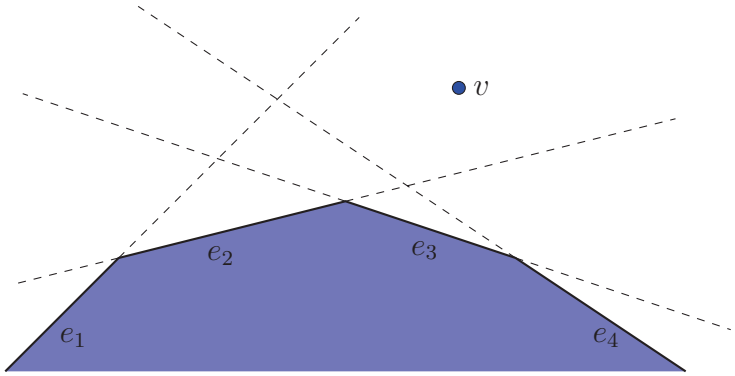


Figure 4.5: A vertex near multiple edges. Depending on the complexity of body geometries, as well as body velocities and the time step size, there may be an arbitrary number of relevant contacts between a vertex of one body and facets of another. Further, some of these contacts may not have a contact force associated with them, but are still necessary when writing contact constraints in order to represent contact geometry.

of $\mathcal{A}$ against multiple features of $\mathcal{B}$ where a subset of these potential contacts may feasibly result in a contact force at the end of the current time step.

Consider two sets of contacts: $\mathbf{C}_\iota$ which is composed of all contacts which could feasibly result in a contact force, and $\mathbf{C}_\kappa$ which contains all contacts that could not feasibly result in a contact force but are necessary to accurately represent the contact geometry. An $\mathcal{I}$-constraint seeks to

- Constrain at least one gap distance $\psi_{\{\iota,\kappa\}}$ of $\{\mathbf{C}_\iota, \mathbf{C}_\kappa\}$ to be non-negative,
- Permit a non-zero contact force $\lambda_{\iota i} > 0$ for at most one contact in $\mathbf{C}_\iota$, and
- Only permit $\lambda_{\iota i} > 0$ when $\psi_i = 0$ and all other $\psi_{\iota,\kappa}$ are non-positive.

We represent an $\mathcal{I}$-constraint as

$$\mathcal{I}(\mathbf{C}_\iota, \mathbf{C}_\kappa) \tag{4.9}$$

which corresponds for $m = |\mathbf{C}_\iota|$ and $n = |\mathbf{C}_\kappa|$ to the set of constraints

$$\begin{array}{rcrcrcl}
0 & \leq & c_{\iota 2} + \psi_{\iota 1} - \psi_{\iota 2} & \perp & c_{\iota 2} & \geq & 0 \\
0 & \leq & c_{\iota 3} + c_{\iota 2} + \psi_{\iota 1} - \psi_{\iota 3} & \perp & c_{\iota 3} & \geq & 0 \\
 & & \vdots & & & & \\
0 & \leq & c_{\iota m} + c_{\iota m-1} + ... + c_{\iota 2} + \psi_{\iota 1} - \psi_{\iota m} & \perp & c_{\iota m} & \geq & 0 \\
0 & \leq & c_{\kappa 1} + c_{\iota m} + c_{\iota m-1} + ... + c_{\iota 2} + \psi_{\iota 1} - \psi_{\kappa 1} & \perp & c_{\kappa 1} & \geq & 0 \\
 & & \vdots & & & & \\
0 & \leq & c_{\kappa n} + ... + c_{\kappa 1} + c_{\iota m} + ... + c_{\iota 2} + \psi_{\iota 1} - \psi_{\kappa n} & \perp & c_{\kappa n} & \geq & 0
\end{array} \tag{4.10}$$

$$c_{\kappa n} + ... + c_{\kappa 1} + c_{\iota m} + ... + c_{\iota 2} + \psi_{\iota 1} \geq 0 \tag{4.11}$$

$$\begin{array}{rcrcrcl}
0 & \leq & d_{\iota 1} + \psi_{\iota 1} & \perp & d_{\iota 1} & \geq & 0 \\
 & & \vdots & & & & \\
0 & \leq & d_{\iota m} + \psi_{\iota m} & \perp & d_{\iota m} & \geq & 0
\end{array} \tag{4.12}$$

$$\begin{array}{rcrcrcl}
0 & \leq & c_{\kappa n} + ... + c_{\kappa 1} + c_{\iota m} + ... + c_{\iota 2} + \psi_{\iota 1} + d_{\iota 1} & \perp & \lambda_{\iota 1} & \geq & 0 \\
 & & \vdots & & & & \\
0 & \leq & c_{\kappa n} + ... + c_{\kappa 1} + c_{\iota m} + ... + c_{\iota 2} + \psi_{\iota 1} + d_{\iota m} & \perp & \lambda_{\iota m} & \geq & 0
\end{array} \tag{4.13}$$

Equations (4.10) and (4.11) enforce that at least one contact in $\{\mathbf{C}_\iota, \mathbf{C}_\kappa\}$ is non-negative. Equations (4.12) and (4.13) allow at most one contact force to be non-zero, and only permit this force to be non-zero when all other gap distances are non-positive. Solving such a set of constraints is possible as a linear program with complementarity constraints (LPCC) [77], or by reducing or rewriting to eliminate the inequality of (4.11).

An $\mathcal{I}$-constraint requires there be at least one contact in $\mathbf{C}_\iota$. If there were not, then there would exist no non-trivial solution since these potential contact forces are necessary for preventing interpenetration. In the case that $|\mathbf{C}_\iota| = 1$, we are able to dramatically reduce the set of constraints by eliminating equations (4.11) and (4.12) and replacing equation (4.13) with the single constraint

$$0 \leq c_{\kappa n} + ... + c_{\kappa 1} + c_{\iota m} + ... + c_{\iota 2} + \psi_{\iota 1} \perp \lambda_\iota \geq 0 \tag{4.14}$$

This greatly reduces the number of constraints, and removes the inequality of (4.11) to leave a pure LCP. We see now the incentive for choosing our contact set with a single primary contact C_ι, possibly using heuristics like those described in Chapter 3.

4.2.3 Cross-contact constraint

In contrast to $\mathcal{I}$-constraint which deals with a single feature of $\mathcal{A}$ against multiple features of $\mathcal{B}$, a cross-contact constraint ($\mathcal{X}$-constraint) deals with sets of contacts that may not depend on the same features, but could result in interpenetration if simultaneously violated, *e.g.,* equation (4.6).

Given sets of contacts $\mathbf{C}_a$ and $\mathbf{C}_b$, a $\mathcal{X}$-constraint attempts to constrain either

- At least one non-negative ψ_a in $\mathbf{C}_a$ OR
- At least one non-negative ψ_b in $\mathbf{C}_b$

We represent a $\mathcal{X}$-constraint as

$$\mathcal{X}(\mathbf{C}_a, \mathbf{C}_b) \tag{4.15}$$

and can write the LCP in multiple forms. Conveniently, $\mathcal{X}$-constraints can be written in the form of equations (4.10) and (4.11). That is, $\mathcal{X}(\mathbf{C}_a, \mathbf{C}_b)$ is similar to $\mathcal{I}(\mathbf{C}_a, \mathbf{C}_b)$, but excludes equations (4.12) and (4.13) which involve contact forces.

Although a $\mathcal{X}$-constraint necessarily deals with contacts between the same pair of bodies, the contacts need not share features. Subsequently, there is no argument for incorporating contact forces directly into a $\mathcal{X}$-constraint since the purpose is to effectively "mask" the set of existing $\mathcal{U}$- and $\mathcal{I}$- constraints to eliminate configurations with interpenetration. It should be understood however that care must be taken, when constructing a time-stepping subproblem containing $\mathcal{X}$-constraints, not to construct a subproblem that is inherently unsatisfiable. This is accomplished by ensuring that there exist sufficient $\mathcal{U}$- or $\mathcal{I}$-constraints so as to allow generation of forces which may contribute to achieving configurations that will satisfy all constraints.

Despite there being no formal need to associate any potential forces with an $\mathcal{X}$-constraint, I will risk sounding contradictory by observing that one may represent an $\mathcal{X}$-constraint as an $\mathcal{I}$-constraint. Doing so will create redundant contact forces at

the solver level, but the total resultant forces from these solutions will be the same as if only a single force component were used, giving the same outcome at the end of the time step. This is convenient in terms of notation as well as implementation, and is used in section 4.7 when we write a time stepping formulation using PEG.

4.3 PEG 2D

We consider two contact configurations in 2D: vertex-edge and vertex-vertex. We do not consider edge-edge because to do so would be redundantly covered by the other two configurations [71]. To be clear, edge-edge penetration in 2D is impossible if vertex-edge and vertex-vertex constraints are properly enforced, simply because edges defined and composed of vertices, and for two edges to come near each other implies that the vertices of at least one of those edges is nearing the other edge. We can formulaically generate the set of constraints for the two 2D configurations as given below.

4.3.1 Vertex-edge

Given a vertex v_a near a single edge e_b, this case is modeled by a unilateral constraint on a single contact as

$$\mathcal{U}(\mathcal{C}(v_a, e_b)) \tag{4.16}$$

In this case, the vertex is within a reasonably small distance ϵ of the edge, but farther than ϵ from either of the edge's vertices.

4.3.2 Vertex-vertex

The 2D vertex-vertex case was depicted in Figure 4.3. Using the contacts defined for this case requires the constraints

$$\begin{gathered} \mathcal{I}(\mathbf{C}_{\iota 1}, \mathbf{C}_{\kappa 1}) \\ \mathcal{I}(\mathbf{C}_{\iota 2}, \mathbf{C}_{\kappa 2}) \\ \mathcal{X}(\{C_1\}, \{C_4\}) \\ \mathcal{X}(\{C_2\}, \{C_3\}) \end{gathered} \tag{4.17}$$

where $\mathbf{C}_{\iota 1}$ contains at least one of C_1 or C_2, $\mathbf{C}_{\kappa 1}$ may contain one of C_1 or C_2, and similarly $\mathbf{C}_{\iota 2}$ contains at least one of C_3 or C_4, and $\mathbf{C}_{\kappa 2}$ may contain one of C_3 or C_4. As is the case for all $\mathcal{I}$-constraints, which contacts are included in $\mathbf{C}_{\iota}$ or $\mathbf{C}_{\kappa}$ are determined using the geometric heuristics defined in chapter 3.

4.3.3 PEG abstraction layers

Given the fundamental constraints for vertex-edge and vertex-vertex cases, we have now been presented with the full approach represented by the abstraction layers of PEG as depicted in Figure 4.6. At the lowest level is the complementarity

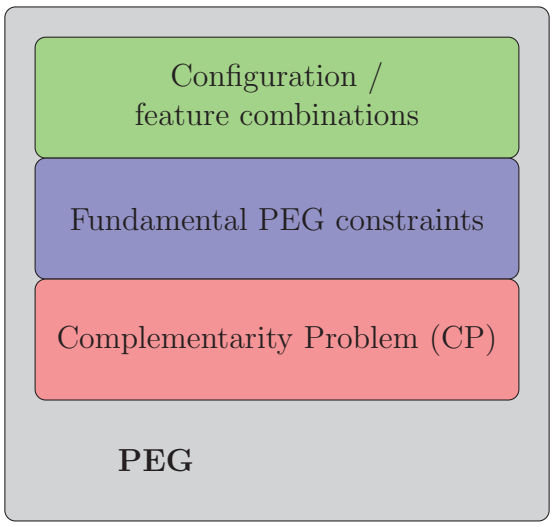


Figure 4.6: Abstraction layers of PEG.

problem which is utilized first as a model of unilateral contact with force, but further as a tool for generating logical XOR conditions on subsets of potential contacts. Sets of these low level constraints can be entirely represented by abstracting to the three fundamental constraints of unilateral, inter-contact, and cross-contact. In turn, the fundamental constraints are formulaically generated by the highest abstraction layer which looks only at what pairs of features between two bodies are near each other. In the next section, we will see how precisely the same abstraction can be developed in 3D.

The following algorithms represent PEG 2D, and roughly correspond to the structure of the abstraction layers. The translation to the lowest level of complementarity problem from fundamental constraints was given previously.

The function PEG-2D-VERTEX-EDGE of Algorithm 8 corresponds to equation (4.16) and represents vertex-edge interaction in 2D.

Algorithm 8 PEG 2D, vertex-edge.

function PEG-2D-VERTEX-EDGE(v_a, e_b)
  if $FEAS^{\{ve\}}(v_a, e_b) \wedge APPL^{\{ve\}}(v_a, e_b) \geq \epsilon_\theta$ **then**
    return $\{\mathcal{U}(\mathcal{C}(v_a, e_b))\}$
  else
    return $\{\emptyset\}$
  end if
end function

The C-SPLIT function of Algorithm 9 is useful as it separates a set of contacts into those that could reasonably result in a force over a time step $\mathbf{C}_\iota$, and those that could not $\mathbf{C}_\kappa$.

Algorithm 9 *C-split*.

function C-SPLIT($\mathbf{C}$)
  $\mathbf{C}_\iota = \{\}$
  $\mathbf{C}_\kappa = \{\}$
  for $C_i \in \mathbf{C}$ **do**
    $f_1 = C_i.feature_1$
    $f_2 = C_i.feature_2$
    if $APPL(f_1, f_2) \wedge FEAS(f_1, f_2)$ **then**
      $\mathbf{C}_\iota = \mathbf{C}_\iota \cup \{C_i\}$
    else
      $\mathbf{C}_\kappa = \mathbf{C}_\kappa \cup \{C_i\}$
    end if
  end for
  return $(\mathbf{C}_\iota, \mathbf{C}_\kappa)$
end function

The function PEG-2D-VERTEX-VERTEX of Algorithm 10 corresponds to equation (4.17) and returns the set of contacts for geometrically accurate interaction based on a vertex-vertex configuration for convex polygons in 2D. One can use Figure 4.3 as a guide and follow Algorithm 10 to see how this algorithm chooses the correct contact set.

The PEG-2D function takes as parameters two bodies $\mathcal{A}$ and $\mathcal{B}$, and generates the lowest level set of PEG constraints representing accurate interactions of those bodies.

Algorithm 10 PEG 2D, vertex-vertex.

Returns the set of PEG constraints between v_a and v_b.
function PEG-2D-VERTEX-VERTEX(v_a, v_b)
$e_{a1} = (v_{a-1}, v_a)$, $e_{a2} = (v_a, v_{a+1})$, $e_{b1} = (v_{b-1}, v_b)$, $e_{b2} = (v_b, v_{b+1})$
$C_1 = \mathcal{C}(v_a, e_{b1})$, $C_2 = \mathcal{C}(v_a, e_{b2})$, $C_3 = \mathcal{C}(v_b, e_{a1})$, $C_4 = \mathcal{C}(v_b, e_{a2})$
$\mathbf{C}_{1\iota} = \mathbf{C}_{1\kappa} = \mathbf{C}_{2\iota} = \mathbf{C}_{2\kappa} = \emptyset$
$(\mathbf{C}_{\iota 1}, \mathbf{C}_{\kappa 1}) = C\text{-}split(\{C_1, C_2\})$
$(\mathbf{C}_{\iota 2}, \mathbf{C}_{\kappa 2}) = C\text{-}split(\{C_3, C_4\})$
return $\{\mathcal{I}(\mathbf{C}_{\iota 1}, \mathbf{C}_{\kappa 1})\} \cup \{\mathcal{I}(\mathbf{C}_{\iota 2}, \mathbf{C}_{\kappa 2})\} \cup \{\mathcal{X}(\{C_1\}, \{C_4\})\} \cup \{\mathcal{X}(\{C_2\}, \{C_3\})\}$
end function

Algorithm 11 PEG 2D.

Given polygons $\mathcal{A}$ and $\mathcal{B}$, generates the set of contact constraints between them.
function PEG-2D($\mathcal{A}, \mathcal{B}$)
$Constraints = \{\}$
for $v_a \in V_\mathcal{A}$, $v_b \in V_\mathcal{B} : ||\mathbf{v}_a - \mathbf{v}_b|| \leq \epsilon$ **do**
$Constraints = Constraints \cup PEG\text{-}2D\text{-}vertex\text{-}vertex(v_a, v_b)$
$v_a.collision = true$
$v_b.collision = true$
end for
for $v_a \in V_\mathcal{A}, e_b = (v_{b1}, v_{b2}) \in E_\mathcal{B} : point\text{-}segment\text{-}2D(\mathbf{v}_a, \mathbf{v}_{b1}, \mathbf{v}_{b2}) \leq \epsilon$ **do**
if $\neg v_a.collision$ **then**
$Constraints = Constraints \cup PEG\text{-}2D\text{-}vertex\text{-}edge(v_a, e_b)$
end if
end for
for $v_b \in V_\mathcal{B}, e_a = (v_{a1}, v_{a2}) \in E_\mathcal{A} : point\text{-}segment\text{-}2D(\mathbf{v}_b, \mathbf{v}_{a1}, \mathbf{v}_{a2}) \leq \epsilon$ **do**
if $\neg v_b.collision$ **then**
$Constraints = Constraints \cup PEG\text{-}2D\text{-}vertex\text{-}edge(v_b, e_a)$
end if
end for
return $Constraints$
end function

4.4 PEG 3D

There are four contact configurations in 3D: vertex-face, edge-edge, vertex-edge, and vertex-vertex. Again, we exclude certain feature pairs just as we did in 2D, namely edge-face and face-face, because they become redundant when the other constraints are properly enforced. Edge-face is redundant since edges are composed of vertices, and for an edge to approach a face requires that either the vertices of that edge approach that face, or the edge approaches the edges at the perimeter of that face (possibly adjacent edges), reducing to either vertex-face or edge-edge respectively, and edge-vertex in the case that the edges of the second body are adjacent. By a similar argument, face-face becomes redundant; a face, being composed of edges in turn composed of vertices, nearing another face implies that edges and vertices are approaching the other face. In all possible configurations, this results in one of the four feature combinations considered.

4.4.1 Vertex-face

The vertex-face case is analogous to the 2D vertex-edge case. Given a vertex v_a near a face f_b as depicted in Figure 4.7, we include

$$\mathcal{U}(\mathcal{C}(v_a, f_b)) \tag{4.18}$$

in the set of constraints.

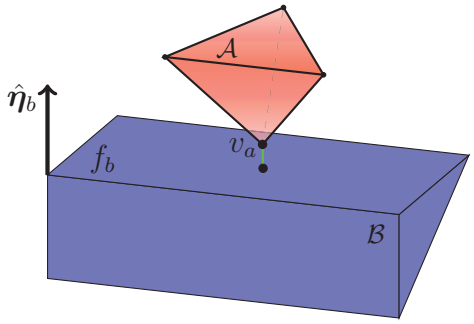


Figure 4.7: A well defined vertex-face contact between vertex v_a of body $\mathcal{A}$ and face f_b of body $\mathcal{B}$. $\mathcal{C}(v_a, f_b)$ is clearly the only contact which could be active.

4.4.2 Edge-edge

Edge-edge contact, such as depicted in Figure 3.6, is another example of a single contact with unilateral constraint, and is written for an edge e_a against an edge e_b by

$$\mathcal{U}(\mathcal{C}(e_a, e_b)) \tag{4.19}$$

4.4.3 Vertex-edge

Consider Figure 4.8 where a vertex v_a of body $\mathcal{A}$ approaches the edge e_b of body $\mathcal{B}$. There are an arbitrary number of edges (greater or equal to three) that could connect to v_a, however we can limit the edge-edge contacts considered to a subset of those with relatively large magnitudes of edge orientation $|\mathbf{e}_a \cdot \mathbf{O}_{e_b}|$. Given

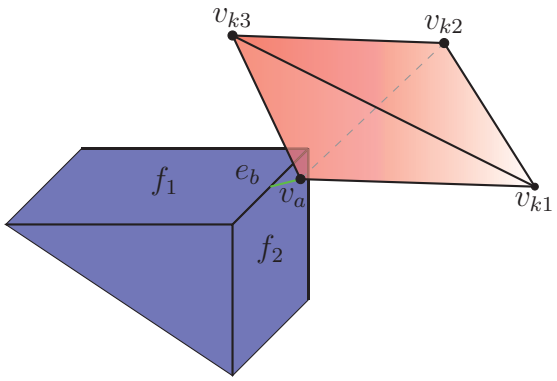


Figure 4.8: An example of a vertex-edge configuration. Consider the relationship between the contacts $\mathcal{C}(v_a, f_{b1})$ and $\mathcal{C}((v_{k3}, v_a), e_b)$. If the first is not enforced, then the second must be, and vice versa.

$C_1 = \mathcal{C}(v_a, f_1)$ and $C_2 = \mathcal{C}(v_a, f_2)$,The set of constraints for vertex-edge is given by

$$\begin{matrix} \mathcal{I}(\mathbf{C}_\iota, \mathbf{C}_\kappa) \\ \mathcal{X}(\{C_{ei}\}, \{C_1\}) \\ \mathcal{X}(\{C_{ej}\}, \{C_2\}) \end{matrix} \tag{4.20}$$

where C_1 and C_2 are distributed appropriately into $\mathbf{C}_\iota$ and $\mathbf{C}_\kappa$. Each C_{ei} is an edge with positive edge orientation, and each C_{ej} is an edge with negative edge orientation, and both sets of edges have applicability and feasibility with e_b.

4.4.4 Vertex-vertex

In the case of a vertex v_a near another body's vertex v_b as depicted in Figure 4.9, we wish to avoid trapping v_a against the faces of $\mathcal{B}$, and similarly for v_b against $\mathcal{A}$. This first requires the set of $\mathcal{I}$-constraints

$$\begin{aligned} &\mathcal{I}(\mathbf{C}_{\iota a}, \mathbf{C}_{\kappa a}) \\ &\mathcal{I}(\mathbf{C}_{\iota b}, \mathbf{C}_{\kappa b}) \end{aligned} \tag{4.21}$$

where $\mathbf{C}_{\iota a}$ and $\mathbf{C}_{\kappa a}$ are composed of contacts between v_a and faces of $\mathcal{B}$, and $\mathbf{C}_{\iota b}$ and $\mathbf{C}_{\kappa b}$ are composed of contacts between v_b and faces of $\mathcal{A}$. We additionally require $\mathcal{X}$-constraints for each face pair (f_a, f_b) in the form

$$\mathcal{X}(\{\mathcal{C}(v_a, f_b), \mathcal{C}(v_b, f_a)\}, \{\mathcal{C}(e_{a1}, e_{b1}), \mathcal{C}(e_{a2}, e_{b2})\}) \tag{4.22}$$

where f_a has edges e_{a1}, e_{a2} connected to v_a, and f_b has edges e_{b1}, and e_{b2} connected to v_b. These $\mathcal{X}$-constraints are enforced only for edges e_{ar} and e_{bs} which have edge-edge applicability and feasibility. The constraint in equation (4.22) is similar to the $\mathcal{X}$-constraints in equation (4.20) for the vertex-edge case, but permits a vertex to pass on either side of a face instead of either side of an edge.

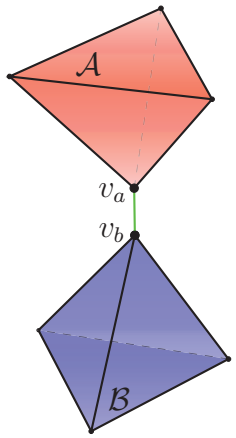


Figure 4.9: A vertex v_a of body $\mathcal{A}$ approaching a vertex v_b of body $\mathcal{B}$ from above. Vertex v_a should be permitted to break the half spaces represented by the falsely extended faces of $\mathcal{B}$, but not all of them and only when doing so does not violate edge-edge contacts.

Given the previous definitions, we can write the algorithms for 3D just as we did for 2D, also corresponding roughly to the same abstraction layers.

Algorithms 12, 13, 15, and 16 respectively correspond to the four configurations, or feature combinations, for 3D in 4.4.1, 4.4.2, 4.4.3 and 4.4.4. Algorithm 14

Algorithm 12 PEG 3D, vertex-face.

function PEG-3D-VERTEX-FACE(v_a, f_b)
  if $FEAS(v_a, f_b) \wedge APPL^{\{vf\}}(v_a, f_b) \geq \epsilon_\theta$ **then**
    return $\{\mathcal{U}(\mathcal{C}(v_a, f_b))\}$
  end if
  return $\{\emptyset\}$
end function

Algorithm 13 PEG 3D, edge-edge.

function PEG-3D-EDGE-EDGE(e_a, e_b)
  if $APPL^{\{ee\}}(e_a, e_b) \geq \epsilon_\theta$ **then**
    return $\{\mathcal{U}(\mathcal{C}(e_a, e_b))\}$
  end if
  return $\{\emptyset\}$
end function

represents a useful auxiliary function that is used for both the vertex-edge and vertex-vertex configurations.

Algorithm 17 keeps track of which edges are involved in contact subsets in order to avoid over-constraining the potential contacts between $\mathcal{A}$ and $\mathcal{B}$. To do so could result in a trap analogous to the standard-model trap.

Algorithm 14 Auxiliary function for 3D PEG functions.

function VERTEX-EDGE-3D-X-CONSTRAINTS(v_a, e_b)
  $C_1 = \mathcal{C}(v_a, e_b.f_{b1})$
  $C_2 = \mathcal{C}(v_a, e_b.f_{b2})$
  $X_{constr} = \{\}$
  for $e_a = (v_a, v_{ai}) \in E_{\mathcal{A}}$ **do**
    if $APPL^{ee}(e_a, e_b) \geq \epsilon_\theta$ **then**
      if $\frac{\mathbf{e}_a}{||\mathbf{e}_a||} \cdot \mathcal{O}_{e_b} \geq |\epsilon_\theta|$ **then** ▷ Only $e_b.f_1$
        $X_{constr} = X_{constr} \cup \{\mathcal{I}(C\text{-}split(C_1, \mathcal{C}(e_a, e_b)))\}$
      else if $\frac{\mathbf{e}_a}{\mathbf{e}_a} \cdot \mathcal{O}_{e_b} \leq |\epsilon_\theta|$ **then** ▷ Only $e_b.f_2$
        $X_{constr} = X_{constr} \cup \{\mathcal{I}(C\text{-}split(C_2, \mathcal{C}(e_a, e_b)))\}$
      else ▷ Both $e_b.f_1$ and $e_b.f_2$
        $C_3 = \mathcal{C}(e_a, e_b)$
        $C_4 = C_3$
        $C_4.\hat{\mathbf{n}} = -C_4.\hat{\mathbf{n}}$ ▷ Flip the contact normal
        if $e_a \cdot \mathcal{O}_{e_b} \geq 0$ **then**
          $X_{constr} = X_{constr} \cup \{\mathcal{I}(C\text{-}split(C_1, C_3)\} \cup \{\mathcal{I}(C\text{-}split(C_2, C_4)\}$
        else
          $X_{constr} = X_{constr} \cup \{\mathcal{I}(C\text{-}split(C_2, C_3)\} \cup \{\mathcal{I}(C\text{-}split(C_1, C_4)\}$
        end if
      end if
    end if
  end for
  return X_{constr}
end function

Algorithm 15 PEG 3D, vertex-edge.

function PEG-3D-VERTEX-EDGE(v_a, e_b)
  $C_1 = \mathcal{C}(v_a, e_b.f_{b1})$
  $C_2 = \mathcal{C}(v_a, e_b.f_{b2})$
  $I_{constr} = \mathcal{I}(C\text{-}split(\{C_1, C_2\}))$
  return $I_{constr} \cup Vertex\text{-}edge\text{-}3D\text{-}X\text{-}constraints(v_a, e_b)$
end function

Algorithm 16 PEG 3D, vertex-vertex.

function PEG-3D-VERTEX-VERTEX(v_a, v_b)
$I_{constr} = \{\mathcal{I}(C\text{-}split(\{\mathcal{C}(v_a, f_b)\})), \mathcal{I}(C\text{-}split(\{\mathcal{C}(v_b, f_a)\}))\} : f_b \in F_{\mathcal{B}}, v_b \in f_b, f_a \in F_{\mathcal{A}}, v_a \in f_a$
$X_{constr} = \{\}$
for $e_b \in E_{\mathcal{B}} : v_b \in e_b$ **do**
$X_{constr} = Vertex\text{-}edge\text{-}3D\text{-}X\text{-}constraints(v_a, e_b)$
end for
for $e_a \in E_{\mathcal{A}} : v_a \in e_a$ **do**
$X_{constr} = Vertex\text{-}edge\text{-}3D\text{-}X\text{-}constraints(v_b, e_a)$
end for
return $I_{constr} \cup X_{constr}$
end function

Algorithm 17 PEG 3D.

Given polyhedra $\mathcal{A}$ and $\mathcal{B}$, generates the set of contact constraints between them.

function PEG-3D($\mathcal{A}, \mathcal{B}$)
  $Constraints = \{\}$
  for $v_a \in V_\mathcal{A}, v_b \in V_\mathcal{B} : ||\mathbf{v}_a - \mathbf{v}_b|| \leq \epsilon$ **do** ▷ Vertex-vertex
    $Constraints = Constraints \cup PEG\text{-}3D\text{-}vertex\text{-}vertex(v_a, v_b)$
    $e_a.collision = true : v_a \in e_a$
    $e_b.collision = true : v_b \in e_b$
  end for
  ▷ Vertex-edge
  for $v_a \in V_\mathcal{A}, e_b = (v_{b1}, v_{b2}) \in E_\mathcal{B} : point\text{-}segment\text{-}distance(\mathbf{v}_{b1}, \mathbf{v}_{b2}, \mathbf{v}_a) \leq \epsilon$ **do**
    if $\neg e_a.collision : v_a \in e_a$ **then**
      $Constraints = Constraints \cup PEG\text{-}3D\text{-}vertex\text{-}edge(v_a, e_b)$
      $e_a.collision = true : v_a \in e_a$
      $e_b.collision = true : v_b \in e_b$
    end if
  end for
  ▷ Vertex-face
  for $v_a \in V_\mathcal{A}, f_b = (v_{b1}, v_{b2}, v_{b3}) \in F_\mathcal{B} : point\text{-}triangle\text{-}distance(\mathbf{v}_a, \mathbf{v}_{b1}, \mathbf{v}_{b2}, \mathbf{v}_{b3}) \leq \epsilon$ **do**
    if $\neg e_a.collision : v_a \in e_a$ **then**
      $Constraints = Constraints \cup PEG\text{-}3D\text{-}vertex\text{-}face(v_a, f_b)$
      $e_a.collision = true : v_a \in e_a$
      $e_b.collision = true : v_b \in e_b$
    end if
  end for
  ▷ Edge-edge
  for $e_a = (v_{a1}, v_{a2}) \in E_\mathcal{A}, e_b = (v_{b1}, v_{b2}) \in E_\mathcal{B} : segment\text{-}segment\text{-}distance\text{-}3D(\mathbf{v}_{a1}, \mathbf{v}_{a2}, \mathbf{v}_{b1}, \mathbf{v}_{b2}) \leq \epsilon$ **do**
    if $\neg e_a.collision \vee \neg e_b.collision$ **then**
      $Constraints = Constraints \cup PEG\text{-}3D\text{-}vertex\text{-}face(v_a, f_b)$
    end if
  end for
  return $Constraints$
end function

4.5 Non-convexity with PEG

We could approach non-convex cases by augmenting the methods described already in this chapter. For example, the 2D vertex-vertex case must be augmented, as it is clear in the configuration of Figure 4.10 that unilateral constraint on both contacts $\mathcal{C}(v_a, e_{b1})$ and $\mathcal{C}(v_a, e_{b2})$ must be enforced. However, we can see for this

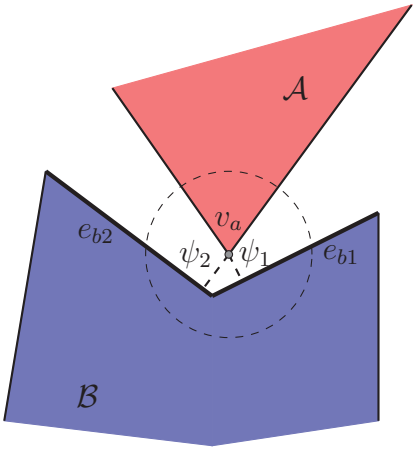


Figure 4.10: Contact determination in a non-convex region of a polygon. Here, two unilateral constraints are required to prevent interpenetration of $\mathcal{A}$ and $\mathcal{B}$. Equivalently, body $\mathcal{B}$ could be decomposed into a union of convex bodies, requiring no modifications to PEG.

particular case that this is equivalent to decomposing body $\mathcal{B}$ into two bodies which overlap near where e_{b1} and e_{b2} meet, or share a vertex there. In fact, convex decomposition is a good general approach to using PEG with non-convex bodies.

Consider the body of Figure 4.11 of which we see four edges and note particularly the two convex vertices v_1 and v_3 and the non-convex vertex v_2. Decomposing this body into a union of two overlapping convex bodies allows us to use PEG on each of these sub-bodies independently.

The decomposition approach depicted if Figure 4.11 avoids two pitfalls in particular: allowing interpenetration for non-area (or non-volume in 3D) bodies such as a single vertex, and regenerating standard-model traps. The first pitfall is avoided by having the convex decompositions overlap as opposed to having zero distance which is large enough to permit a vertex through. The second pitfall is avoided

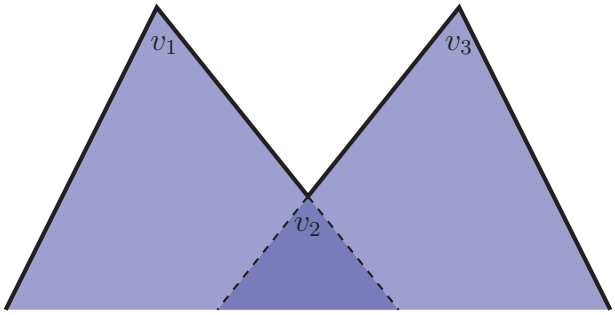


Figure 4.11: A portion of a non-convex body and a possible decomposition into two overlapping convex parts.

since the decomposition will add edges on the interior of the body, as opposed to extending the original edges as infinite half spaces which could potentially recreate the standard-model traps we were originally trying to avoid.

Figure 4.12 is an example of a saddle point vertex. This vertex has degree of four where two of the edges adjoin faces which are convex relative to the interior of the body, and two of the edges adjoin faces which are non-convex. This particular example is one which, when identified, can be handled with configuration approaches we have already defined for PEG. In particular, if a vertex of another body is approaching this saddle point vertex, we can utilize PEG's vertex-edge constraints for the approaching vertex against the two edges of the saddle that are convex. Letting the convex edges be e_1 and e_2 and the non-convex edges be e_3 and e_4, we can write the constraints for a vertex v_a of a convex body against this vertex as

$$\begin{gathered}
\mathcal{I}(\mathbf{C}_{\iota 1}, \mathbf{C}_{\kappa 1}) \\
\mathcal{X}(\{C_{e1i}\}, \{C_{1a}\}) \\
\mathcal{X}(\{C_{e1j}\}, \{C_{1b}\}) \\
\mathcal{I}(\mathbf{C}_{\iota 2}, \mathbf{C}_{\kappa 2}) \\
\mathcal{X}(\{C_{e2i}\}, \{C_{2a}\}) \\
\mathcal{X}(\{C_{e2j}\}, \{C_{2b}\})
\end{gathered} \tag{4.23}$$

where the first three constraints are for v_a against e_1 and the last three are for v_a against e_2. Being able to write (4.23) for this configuration is in great part due to the fact that the plane defined by the two convex edges is inside the body, and

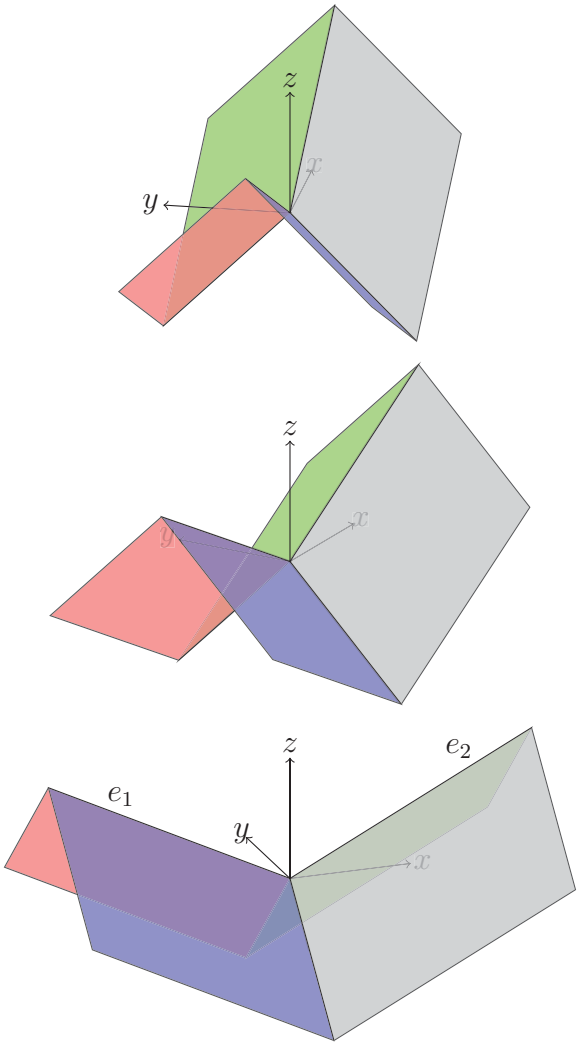


Figure 4.12: Three views of a saddle point vertex, a vertex with two convex edges (top edges) and two non-convex edges (middle side edges). The interior of the body is below the faces.

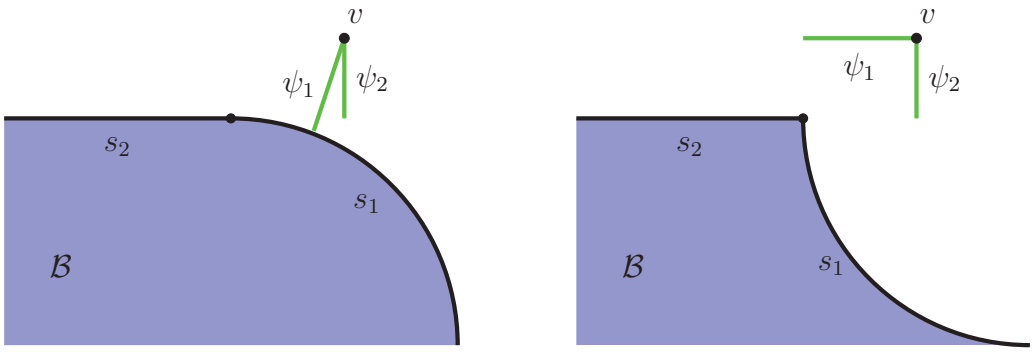
moreover that a vertex that is on one side of this plane for one of those edges will necessarily be on the same side of this plane for the other edge. Further, note that we did not have to include e_3 and e_4 in the constraints. This formulation exploits the fact that having independent constraints, in this case those on e_1 and those on e_2, is precisely the AND condition that is appropriate for non-convex regions. Having simultaneous constraints on the two faces of e_3 is enough to prevent penetration of these faces, and similarly for e_4. This turns out to be exactly the same approach as would be taken if we had decomposed the body into a union of convex bodies.

4.6 PEG with non-polytopes.

It is possible to utilize the underlying mathematical framework of PEG for non-polytope bodies, for example, those defined by semi-algebraic sets [78], or non-uniform rational basis splines (NURBs) which include Bézier curves and B-splines. The framework to which we are referring is that which allows us to effectively translate sets of AND conditions on unilateral constraints into subsets of OR conditions. The only requirement for utilizing PEG for alternative body geometries is that at the very lowest level, the concept of a single contact is fundamentally unilateral, as depicted in Figure 2.2.

For alternative body geometries, the approach would still follow that represented by the abstraction layers depicted in Figure 4.6. The underlying mathematical tools at the lowest level remain the same. The difference for each type of body geometry would be the set of fundamental constraints, as well as the possible configurations which generate the sets of these fundamental constraints.

For example, consider the two different cases of a particle v approaching a body $\mathcal{B}$ with surfaces as depicted in Figure 4.13. Instead of line segment edges,



(a) **(b)**

Figure 4.13: Two examples of non-polytopal body geometries whose contact interactions can be modeled using PEG. In (b), ψ_1 is determined using the tangent at the point on s_1 nearest to particle v.

body $\mathcal{B}$ is composed of surfaces s_1 and s_2. In 4.13a, s_1 is convex, and in 4.13b, s_1 is non-convex. In both cases, the first contact $C_1 = \mathcal{C}(v, s_1)$ is determined by the

closest distance not to that feature, but to the curve that represents that feature. In the case of C_1, we have used to the tangent of s_1 at the point nearest to v. This is precisely the same as what is done for non-curved edges such as s_2 where $C_2 = \mathcal{C}(v, s_2)$. We may observe that for both cases of curved and straight facets, as well as both cases of convex and non-convex, it is possible for the contact point p_b on body $\mathcal{B}$ to be located off the surface of $\mathcal{B}$. From here, for both examples, we make the PEG-like observations that the particle v can only be in contact with one of s_1 or s_2, not both. Further, C_1 should be allowed to be violated as long as C_2 is not, and vice versa. This set of contact constraints is easily written for either of our examples as

$$\begin{array}{rcccccl} 0 & \leq & c_1 + \psi_1 - \psi_2 & \perp & c_1 & \geq & 0 \\ & & c_1 + \psi_1 & & & \geq & 0 \\ 0 & \leq & d_1 + \psi_1 & \perp & d_1 & \geq & 0 \\ 0 & \leq & d_2 + \psi_2 & \perp & d_2 & \geq & 0 \\ 0 & \leq & c_1 + \psi_1 + d_1 & \perp & \lambda_1 & \geq & 0 \\ 0 & \leq & c_1 + \psi_2 + d_2 & \perp & \lambda_2 & \geq & 0 \end{array} \tag{4.24}$$

where just as was the case in our derivation of PEG, c_1, d_1, and d_2 are slack variables, and λ_1 and λ_2 are the potential contact forces associated with C_1 and C_2, respectively. Equation (4.24) can even be simplified to the equivalent

$$\mathcal{I}(\{C_1, C_2\}, \{\}) \tag{4.25}$$

where both C_1 and C_2 have applicability and feasibility.

It should be noted that time-stepping with curved surfaces does introduce error similar to that due to rotation of polytopes over the time step (see 5.3 for details). This is due to the nature of the unilateral constraint used to model contact, as described in 2.3, which treats the region of contact as an infinite half space divided by a line in 2D or plane in 3D. Figure 4.14 depicts an example of this contact model applied to a curved surface. This is a well known issue with curved surfaces in time-stepping simulation and can be ignored for small time steps. Alternatively, depending on the application and required behavior, multiple contacts can be used at points on s within some small distance $\pm\delta$ from the point nearest to v, or constraints

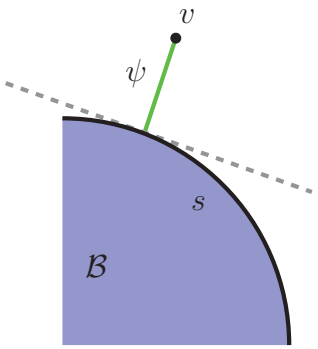


Figure 4.14: Contact between a particle v and a curved surface. The dashed line represents the half space corresponding to the non-penetration constraint. At each time step, v is prevented from crossing the tangent of s at the point nearest to v.

can be written to implicitly account for the curved body geometries [78].

We begin to approach non-convex non-polytopal bodies with curved surfaces in the same manner as we did for non-convex polytopal bodies. That is, we decompose them into unions of sub-bodies. However, it requires theoretically infinitely many sub-bodies to decompose a region with a non-convex curve. One approach to dealing with this issue is depicted in Figure 4.15, where a larger body is decomposed into sub-bodies which are either convex, or have their exterior defined by a single curve. Contact between this body and any other is then done per sub-body where contact

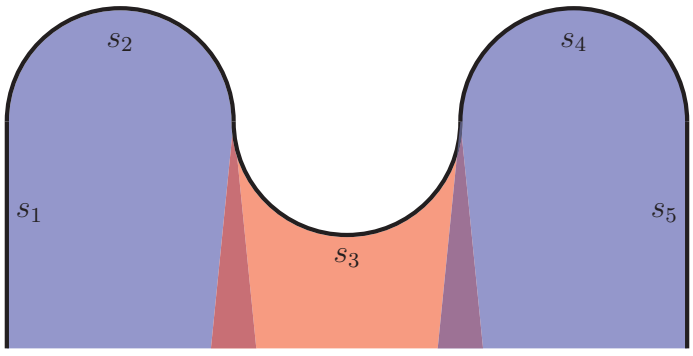


Figure 4.15: A single body composed of convex and non-convex curved surfaces. The thick black line represents the body perimeter while the alternating colors correspond to sub-bodies in the decomposition. Sub-bodies overlap to prevent possible interpenetration from zero-area bodies such as particles.

with a non-convex region of this body includes $\mathcal{I}$-constraints between the exterior curve and the adjacent interior surfaces created during decomposition.

4.7 Time-stepping formulation with PEG

As an independent contact model, there are any number of time-stepping methods into which we may incorporate PEG. However, I have regularly used the Stewart-Trinkle method of equations (2.34) and (2.35). To do this, we start by discretizing and rewriting the constraints of PEG, and here we'll focus on the case where we make a choice of which contact is the primary contact which may have a potential force associated with it. Then we let that primary contact be the first in the set. We have already seen the unilateral constraint in section 2.4. An $\mathcal{I}$-constraint may be discretized also by substituting equation (2.25) into (4.10) and (4.13) to get

$$\mathbf{0} \leq \mathbf{E}_{\mathcal{I}1}\mathbf{c}_{\mathcal{I}}^{\ell+1} + \frac{\boldsymbol{\psi}_{\mathcal{I}}{}^{\ell}}{h} + \mathbf{G}_{\mathcal{I}}^{T}\boldsymbol{\nu}^{\ell+1} + \frac{\boldsymbol{\psi}_{\mathcal{I}}^{\ell}}{\partial h} \perp \mathbf{c}_{\mathcal{I}}^{\ell+1} \geq \mathbf{0} \tag{4.26}$$

$$\mathbf{0} \leq \mathbf{E}_{\mathcal{I}2}\mathbf{c}_{\mathcal{I}}^{\ell+1} + \frac{\boldsymbol{\psi}_{\mathcal{I}\iota}^{\ell}}{h} + \mathbf{G}_{\mathcal{I}\iota}^{T}\boldsymbol{\nu}^{\ell+1} + \frac{\partial\boldsymbol{\psi}_{\mathcal{I}\iota}^{\ell}}{\partial t} \perp \mathbf{p}_{\mathcal{I}}^{\ell+1} \geq \mathbf{0} \tag{4.27}$$

for which we also extend the Newton-Euler equations to include

$$\mathbf{M}\boldsymbol{\nu}^{\ell+1} = \mathbf{M}\boldsymbol{\nu}^{\ell} + \cdots + \mathbf{G}_{n\mathcal{I}}\mathbf{p}_{n\mathcal{I}}^{\ell+1} \tag{4.28}$$

where for m $\mathcal{I}$-constraints, $\mathbf{c}_{\mathcal{I}}$ is the solution vector composed of unknown variables

$$\mathbf{c}_{\mathcal{I}}^{\ell+1} = \begin{bmatrix} \mathbf{c}_{\mathcal{I}1}^{\ell+1} \\ \vdots \\ \mathbf{c}_{\mathcal{I}j}^{\ell+1} \\ \vdots \\ \mathbf{c}_{\mathcal{I}m}^{\ell+1} \end{bmatrix}$$

where the cardinality of each $\mathbf{c}_{\mathcal{I}j}^{\ell+1}$ is k where the j^{th} $\mathcal{I}$-constraint has $k+1$ sub-contacts, $\mathbf{E}_{\mathcal{I}1}$ is diagonally composed of sub-matrices where

$$\mathbf{E}_{\mathcal{I}1j} = \begin{bmatrix} 1 & \mathbf{0} & 0 \\ \mathbf{1} & \ddots & \mathbf{0} \\ 1 & \mathbf{1} & 1 \end{bmatrix}$$

which is the lower triangular matrix of size k, $\mathbf{E}_{\mathcal{I}2}$ is diagonally composed of sub-matrices of row vectors of ones where

$$\mathbf{E}_{\mathcal{I}2j} = \begin{bmatrix} 1 & 1 & \dots & 1 \end{bmatrix}$$

has length k, $\boldsymbol{\psi}_{\mathcal{I}}^{\ell}$ is the vector

$$\boldsymbol{\psi}_{\mathcal{I}}^{\ell} = \begin{bmatrix} \boldsymbol{\psi}_{\mathcal{I}1}^{\ell} \\ \vdots \\ \boldsymbol{\psi}_{\mathcal{I}j}^{\ell} \\ \vdots \\ \boldsymbol{\psi}_{\mathcal{I}m}^{\ell} \end{bmatrix}$$

where each sub-vector $\boldsymbol{\psi}_{\mathcal{I}j}^{\ell}$ is given by

$$\boldsymbol{\psi}_{\mathcal{I}j} = \begin{bmatrix} (\frac{\psi_{\mathcal{I}j_1}}{h} + \frac{\psi_{\mathcal{I}j_1}}{\partial t}) - (\frac{\psi_{\mathcal{I}j_2}}{h} + \frac{\psi_{\mathcal{I}j_2}}{\partial t}) \\ \vdots \\ (\frac{\psi_{\mathcal{I}j_1}}{h} + \frac{\psi_{\mathcal{I}j_1}}{\partial t}) - (\frac{\psi_{\mathcal{I}j_k}}{h} + \frac{\psi_{\mathcal{I}j_k}}{\partial t}) \end{bmatrix},$$

$\boldsymbol{\psi}_{\mathcal{I}\iota}$ is the vector of primary sub-contacts

$$\boldsymbol{\psi}_{\mathcal{I}\iota} = \begin{bmatrix} \frac{\psi_{\mathcal{I}\iota_1}}{h} + \frac{\psi_{\mathcal{I}\iota_1}}{\partial t} \\ \vdots \\ \frac{\psi_{\mathcal{I}\iota_m}}{h} + \frac{\psi_{\mathcal{I}\iota_m}}{\partial t} \end{bmatrix},$$

$\mathbf{p}_{\mathcal{I}}^{\ell+1}$ is a vector of unknown impulses

$$\mathbf{p}_{\mathcal{I}}^{\ell+1} = \begin{bmatrix} \mathbf{p}_{\mathcal{I}1}^{\ell+1} \\ \vdots \\ \mathbf{p}_{\mathcal{I}j}^{\ell+1} \\ \vdots \\ \mathbf{p}_{\mathcal{I}m}^{\ell+1} \end{bmatrix},$$

$\mathbf{G}_{\mathcal{I}}$ is the Jacobian difference matrix diagonally composed of sub-matrices $\mathbf{G}_{\mathcal{I}j}$ each composed of sub-matrices $\mathbf{G}_{\mathcal{I}j_{is}}$ where

$$\mathbf{G}_{\mathcal{I}j_{is}} = \begin{bmatrix} \hat{\mathbf{n}}_{j1} & - & \hat{\mathbf{n}}_{js} \\ \mathbf{r}_{j1} \times \hat{\mathbf{n}}_{j1} & & \mathbf{r}_{js} \times \hat{\mathbf{n}}_{js} \end{bmatrix}, \tag{4.29}$$

and $\mathbf{G}_{n\mathcal{I}}$ is the Jacobian matrix of only primary contacts

$$\mathbf{G}_{n\mathcal{I}j} = \begin{bmatrix} \hat{\mathbf{n}}_{j1} \\ \mathbf{r}_{j1} \times \hat{\mathbf{n}}_{j1} \end{bmatrix}.$$

An $\mathcal{X}$-constraint can be discretized and incorporated in precisely the same way as the $\mathcal{I}$-constraint. The result is that the PEG model may be written with quadratic friction as

$$\mathbf{M}\boldsymbol{\nu}^{\ell+1} = \mathbf{M}\boldsymbol{\nu}^{\ell} + \mathbf{G}_b\mathbf{p}_b^{\ell+1} + \mathbf{G}_n\mathbf{p}_n^{\ell+1} + \mathbf{G}_{n\mathcal{I}}\mathbf{p}_{\mathcal{I}}^{\ell+1} + \mathbf{G}_{n\mathcal{X}}\mathbf{p}_{\mathcal{X}}^{\ell+1} + \mathbf{G}_f\mathbf{p}_f^{\ell+1} + \mathbf{p}_{ext}^{\ell} \tag{4.30}$$

$$\mathbf{0} = \boldsymbol{\psi}_b^{\ell} + \mathbf{G}_b^T\boldsymbol{\nu}^{\ell+1}h + \frac{\partial\boldsymbol{\psi}_b^{\ell}}{\partial t}h \tag{4.31}$$

$$\mathbf{0} \leq \boldsymbol{\psi}_n^{\ell} + \mathbf{G}_n^T\boldsymbol{\nu}^{\ell+1}h + \frac{\partial\boldsymbol{\psi}_n^{\ell}}{\partial t}h \perp \mathbf{p}_n^{\ell+1} \geq \mathbf{0} \tag{4.32}$$

$$\mathbf{0} \leq \mathbf{E}_{\mathcal{I}1}\mathbf{c}_{\mathcal{I}}^{\ell+1} + \frac{\boldsymbol{\psi}_{\mathcal{I}}{}^{\ell}}{h} + \mathbf{G}_{\mathcal{I}}^T\boldsymbol{\nu}^{\ell+1} + \frac{\boldsymbol{\psi}_{\mathcal{I}}^{\ell}}{\partial h} \perp \mathbf{c}_{\mathcal{I}}^{\ell+1} \geq \mathbf{0} \tag{4.33}$$

$$\mathbf{0} \leq \mathbf{E}_{\mathcal{I}2}\mathbf{c}_{\mathcal{I}}^{\ell+1} + \boldsymbol{\psi}_{\mathcal{I}}^{\ell} \perp \mathbf{p}_{\mathcal{I}}^{\ell+1} \geq \mathbf{0} \tag{4.34}$$

$$\mathbf{0} \leq \mathbf{E}_{\mathcal{X}1}\mathbf{c}_{\mathcal{X}}^{\ell+1} + \frac{\boldsymbol{\psi}_{\mathcal{X}}{}^{\ell}}{h} + \mathbf{G}_{\mathcal{X}}^T\boldsymbol{\nu}^{\ell+1} + \frac{\boldsymbol{\psi}_{\mathcal{X}}^{\ell}}{\partial h} \perp \mathbf{c}_{\mathcal{X}}^{\ell+1} \geq \mathbf{0} \tag{4.35}$$

$$\mathbf{0} \leq \mathbf{E}_{\mathcal{X}2}\mathbf{c}_{\mathcal{X}}^{\ell+1} + \boldsymbol{\psi}_{\mathcal{X}}^{\ell} \perp \mathbf{p}_{\mathcal{X}}^{\ell+1} \geq \mathbf{0} \tag{4.36}$$

$$\mathbf{0} = \left(\mathbf{U}\mathbf{p}_n^{\ell+1}\right) \circ \mathbf{u}_t^{\ell+1} + \mathbf{p}_t^{\ell+1} \circ \mathbf{s}^{\ell+1} \tag{4.37}$$

$$\mathbf{0} = \left(\mathbf{U}\mathbf{p}_n^{\ell+1}\right) \circ \mathbf{u}_o^{\ell+1} + \mathbf{p}_o^{\ell+1} \circ \mathbf{s}^{\ell+1} \tag{4.38}$$

$$\boldsymbol{\sigma} = \left(\mathbf{U}\mathbf{p}_n^{\ell+1}\right) \circ \left(\mathbf{U}\mathbf{p}_n^{\ell+1}\right) - \mathbf{p}_t^{\ell+1} \circ \mathbf{p}_t^{\ell+1} - \mathbf{p}_o^{\ell+1} \circ \mathbf{p}_o^{\ell+1} \tag{4.39}$$

$$\mathbf{0} \leq \boldsymbol{\sigma}^{\ell+1} \perp \mathbf{s}^{\ell+1} \geq \mathbf{0} \tag{4.40}$$

where equations (4.37) through (4.40) represent the quadratic friction constraints of Coulomb friction, and we have augmented the appropriate values, *e.g.,* $\mathbf{U}$, to include the primary sub-contacts of the $\mathcal{I}$ and $\mathcal{X}$-constraints.

We can also write PEG with the linearized friction model by replacing the quadratic friction constraints.

$$\mathbf{M}\boldsymbol{\nu}^{\ell+1} = \mathbf{M}\boldsymbol{\nu}^{\ell} + \mathbf{G}_b\mathbf{p}_b^{\ell+1} + \mathbf{G}_n\mathbf{p}_n^{\ell+1} + \mathbf{G}_{n\mathcal{I}}\mathbf{p}_{\mathcal{I}}^{\ell+1} + \mathbf{G}_{n\mathcal{X}}\mathbf{p}_{\mathcal{X}}^{\ell+1} + \mathbf{G}_f\mathbf{p}_f^{\ell+1} + \mathbf{p}_{ext}^{\ell} \tag{4.41}$$

$$\mathbf{0} = \boldsymbol{\psi}_b^{\ell} + \mathbf{G}_b^T\boldsymbol{\nu}^{\ell+1}h + \frac{\partial\boldsymbol{\psi}_b^{\ell}}{\partial t}h \tag{4.42}$$

$$\mathbf{0} \leq \boldsymbol{\psi}_n^{\ell} + \mathbf{G}_n^T\boldsymbol{\nu}^{\ell+1}h + \frac{\partial\boldsymbol{\psi}_n^{\ell}}{\partial t}h \perp \mathbf{p}_n^{\ell+1} \geq \mathbf{0} \tag{4.43}$$

$$\mathbf{0} \leq \mathbf{E}_{\mathcal{I}1}\mathbf{c}_{\mathcal{I}}^{\ell+1} + \frac{\boldsymbol{\psi}_{\mathcal{I}}}{h}^{\ell} + \mathbf{G}_{\mathcal{I}}^T\boldsymbol{\nu}^{\ell+1} + \frac{\boldsymbol{\psi}_{\mathcal{I}}^{\ell}}{\partial h} \perp \mathbf{c}_{\mathcal{I}}^{\ell+1} \geq \mathbf{0} \tag{4.44}$$

$$\mathbf{0} \leq \mathbf{E}_{\mathcal{I}2}\mathbf{c}_{\mathcal{I}}^{\ell+1} + \boldsymbol{\psi}_{\mathcal{I}}^{\ell} \perp \mathbf{p}_{\mathcal{I}}^{\ell+1} \geq \mathbf{0} \tag{4.45}$$

$$\mathbf{0} \leq \mathbf{E}_{\mathcal{X}1}\mathbf{c}_{\mathcal{X}}^{\ell+1} + \frac{\boldsymbol{\psi}_{\mathcal{X}}}{h}^{\ell} + \mathbf{G}_{\mathcal{X}}^T\boldsymbol{\nu}^{\ell+1} + \frac{\boldsymbol{\psi}_{\mathcal{X}}^{\ell}}{\partial h} \perp \mathbf{c}_{\mathcal{X}}^{\ell+1} \geq \mathbf{0} \tag{4.46}$$

$$\mathbf{0} \leq \mathbf{E}_{\mathcal{X}2}\mathbf{c}_{\mathcal{X}}^{\ell+1} + \boldsymbol{\psi}_{\mathcal{X}}^{\ell} \perp \mathbf{p}_{\mathcal{X}}^{\ell+1} \geq \mathbf{0} \tag{4.47}$$

$$\mathbf{0} \leq \mathbf{G}_f\boldsymbol{\nu}^{\ell+1} + \mathbf{E}\mathbf{s}^{\ell+1} \perp \mathbf{p}_f^{\ell+1} \geq \mathbf{0} \tag{4.48}$$

$$\mathbf{0} \leq \mathbf{U}\mathbf{p}_{n\mathcal{I}\mathcal{X}}^{\ell+1} - \mathbf{E}^T\mathbf{p}_f^{\ell+1} \perp \mathbf{s}^{\ell+1} \geq \mathbf{0} \tag{4.49}$$

where we have augmented $\mathbf{G}_f$, $\mathbf{U}$, and $\mathbf{E}$ of the friction constraints of equation (4.48) to include friction information on the primary contacts of the $\mathcal{I}$ and $\mathcal{X}$-constraints.

Considering that in this form, $\mathcal{X}$-constraints may be represented as $\mathcal{I}$-constraints,

let's simplify by using the linearized friction cone and letting

$$\begin{aligned}
\boldsymbol{\rho}_n^{\ell+1} &= \mathbf{G}_n^T\boldsymbol{\nu}^{\ell+1} + \tfrac{\boldsymbol{\psi}_n}{h} + \tfrac{\boldsymbol{\psi}_n}{\partial t} \\
\boldsymbol{\rho}_{n\mathcal{I}}^{\ell+1} &= \mathbf{G}_{n\mathcal{I}}^T\boldsymbol{\nu}^{\ell+1} + \mathbf{E}_{\mathcal{I}2}\mathbf{c}_{\mathcal{I}}^{\ell+1} + \boldsymbol{\psi}_{\mathcal{I}\iota} \\
\boldsymbol{\gamma}_{\mathcal{I}}^{\ell+1} &= \mathbf{G}_{\mathcal{I}}^T\boldsymbol{\nu}^{\ell+1} + \mathbf{E}_{\mathcal{I}1}\mathbf{c}_{\mathcal{I}}^{\ell+1} + \boldsymbol{\psi}_{\mathcal{I}}^{\ell} \\
\boldsymbol{\rho}_f^{\ell+1} &= \mathbf{G}_f^T\boldsymbol{\nu}^{\ell+1} + \mathbf{E}\mathbf{s}^{\ell+1} \\
\boldsymbol{\sigma}^{\ell+1} &= \mathbf{U}\mathbf{p}_n^{\ell+1} - \mathbf{E}^T\mathbf{p}_f^{\ell+1}
\end{aligned}$$

then writing this PEG model in the form of a MCP as

$$\begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \boldsymbol{\rho}_n^{\ell+1} \\ \boldsymbol{\rho}_{n\mathcal{I}}^{\ell+1} \\ \boldsymbol{\gamma}_{\mathcal{I}}^{\ell+1} \\ \boldsymbol{\rho}_f^{\ell+1} \\ \boldsymbol{\rho}_{f\mathcal{I}}^{\ell+1} \\ \boldsymbol{\sigma}^{\ell+1} \\ \boldsymbol{\sigma}_{\mathcal{I}}^{\ell+1} \end{bmatrix} = \begin{bmatrix} -\mathbf{M} & \mathbf{G}_b & \mathbf{G}_n & \mathbf{G}_{n\mathcal{I}} & \mathbf{0} & \mathbf{G}_f & \mathbf{G}_{f\mathcal{I}} & \mathbf{0} & \mathbf{0} \\ \mathbf{G}_b^T & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{G}_n^T & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{G}_{n\mathcal{I}}^T & \mathbf{0} & \mathbf{0} & \mathbf{E}_{\mathcal{I}2} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{G}_{\mathcal{I}}^T & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{E}_{\mathcal{I}1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{G}_f^T & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{E} & \mathbf{0} \\ \mathbf{G}_{f\mathcal{I}}^T & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{E}_{\mathcal{I}} \\ \mathbf{0} & \mathbf{0} & \mathbf{U} & \mathbf{0} & \mathbf{0} & -\mathbf{E}^T & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{U}_{\mathcal{I}} & \mathbf{0} & \mathbf{0} & -\mathbf{E}_{\mathcal{I}}^T & \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\nu}^{\ell+1} \\ \mathbf{p}_b^{\ell+1} \\ \mathbf{p}_n^{\ell+1} \\ \mathbf{p}_{n\mathcal{I}}^{\ell+1} \\ \mathbf{c}_{\mathcal{I}}^{\ell+1} \\ \mathbf{p}_f^{\ell+1} \\ \mathbf{p}_{f\mathcal{I}}^{\ell+1} \\ \mathbf{s}^{\ell+1} \\ \mathbf{s}_{\mathcal{I}}^{\ell+1} \end{bmatrix} + \begin{bmatrix} \mathbf{M}\boldsymbol{\nu}^{\ell} + \mathbf{p}_{ext} \\ \frac{\boldsymbol{\psi}_b}{h} + \frac{\partial \boldsymbol{\psi}_b}{\partial t} \\ \frac{\boldsymbol{\psi}_n}{h} + \frac{\partial \boldsymbol{\psi}_n}{\partial t} \\ \boldsymbol{\psi}_{\mathcal{I}\iota}^{\ell} \\ \boldsymbol{\psi}_{\mathcal{I}}^{\ell} \\ \frac{\partial \boldsymbol{\psi}_f}{\partial t} \\ \frac{\partial \boldsymbol{\psi}_{f\mathcal{I}}}{\partial t} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} \tag{4.50}$$

$$\mathbf{0} \leq \begin{bmatrix} \boldsymbol{\rho}_n^{\ell+1} \\ \boldsymbol{\rho}_{n\mathcal{I}}^{\ell+1} \\ \boldsymbol{\gamma}_{\mathcal{I}}^{\ell+1} \\ \boldsymbol{\rho}_f^{\ell+1} \\ \boldsymbol{\rho}_{f\mathcal{I}}^{\ell+1} \\ \boldsymbol{\sigma}^{\ell+1} \\ \boldsymbol{\sigma}_{\mathcal{I}}^{\ell+1} \end{bmatrix} \perp \begin{bmatrix} \mathbf{p}_n^{\ell+1} \\ \mathbf{p}_{n\mathcal{I}}^{\ell+1} \\ \mathbf{c}_{\mathcal{I}}^{\ell+1} \\ \mathbf{p}_f^{\ell+1} \\ \mathbf{p}_{f\mathcal{I}}^{\ell+1} \\ \mathbf{s}^{\ell+1} \\ \mathbf{s}_{\mathcal{I}}^{\ell+1} \end{bmatrix} \geq \mathbf{0} \tag{4.51}$$

where we notice that our friction for the $\mathcal{I}$-constraints very nearly matches that for the $\mathcal{U}$-constraints. We could reduce this notation by redefining $\mathbf{E}_{\mathcal{I}2}$ to include zero entries for $\mathcal{U}$-constraints.

It is a good question at this point to ask what the size of the MCP will be for PEG as well as what it will have been for the standard model in Stewart-Trinkle for

comparison. For both models, the number of bodies $n_B > 0$ dynamically involved would generate (in 3D) a mass matrix of size $6n_B \times 6n_B$, and $n_b \geq 0$ bilateral constraints will generate a $\mathbf{G}_b$ of size $6n_B \times n_b$. The problem size diverges for unilateral and additional contact constraints since a subset of unilateral constraints from the standard model will be separated into $\mathcal{I}$ and $\mathcal{X}$-constraints with PEG. Given $n_d \geq 0$ friction directions in the linearized friction cone, n_u unilateral constraints with the standard model generate a $6n_B \times n_u$ matrix $\mathbf{G}_n$ and a $6n_B \times n_d$ matrix $\mathbf{G}_f$ with additional $6n_B$, so that the total size in MCP rows of the standard model is given by

$$6n_B + n_b + n_u + n_u n_d + n_u \tag{4.52}$$

With PEG, some subset of contacts of size n_s, where n_s can be any of 0, 2, 3, ..., n_u, will be involved in $\mathcal{I}$ or $\mathcal{X}$-constraints. However, there could be one single $\mathcal{I}$-constraint made up of n_s contacts, or as many as $\left\lfloor \frac{n_s}{2} \right\rfloor$ $\mathcal{I}$-constraints since each $\mathcal{I}$-constraint involves at least two contacts. Further, the higher the percentage of $\mathcal{I}$ constraints, the smaller the problem size will be, due solely to there being a smaller number of contacts which could result in frictional forces. The general problem size for PEG is given by

$$6n_B + n_b + (n_u - n_s) + (n_u - n_s)n_d + (n_u - n_s) + n_s + \left\lfloor \frac{n_s}{2} \right\rfloor n_d + \left\lfloor \frac{n_s}{2} \right\rfloor \tag{4.53}$$

which has an upper bound where $n_s = 0$, which is precisely equal to (4.52), the problem size for the standard model. The next highest bound for PEG is when $n_c \geq 2$ and $n_s = 2$, and is given by

$$6n_B + n_b + (n_u - 2) + (n_u - 2)n_d + (n_u - 2) + 2 + n_d + 1 \tag{4.54}$$

where $2 + n_d + 1$ is the number of row/columns for the $\mathcal{I}$-constraint with $n_d + 1$ columns/rows required for friction on a single $\mathcal{I}$-constraint on two contacts. The lower bound for PEG is given for $n_s = n_c$ by

$$6n_B + n_b + n_s + \left\lfloor \frac{n_s}{2} \right\rfloor n_d + \left\lfloor \frac{n_s}{2} \right\rfloor \tag{4.55}$$

The bounds on problem size are depicted in Figure 4.16 in comparison with the standard model. It can be shown that the PEG formulation is generally smaller

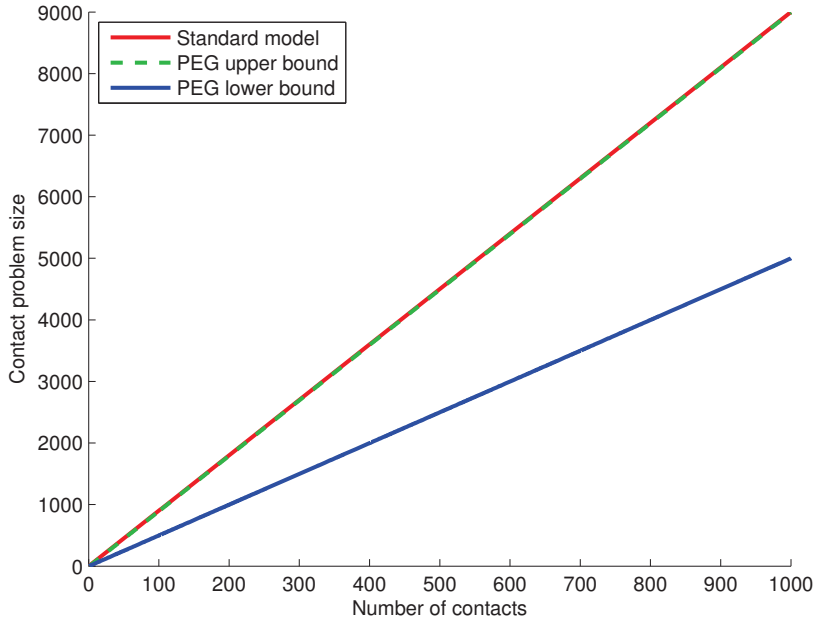


Figure 4.16: Comparison of problem sizes for contact constraints only. For both PEG and the standard model, the problem size increases with the number of contacts. The standard model

than or equal to the standard model formulation: We wish to show that (4.53) is always smaller or equal to (4.52). It follows that

$$6n_B + n_b + 2n_u - n_s + \left\lfloor \frac{n_s}{2} \right\rfloor + (n_u - n_s + \left\lfloor \frac{n_s}{2} \right\rfloor)n_d \leq 6n_B + n_b + 2n_u + n_u n_d$$

$$-n_s + \left\lfloor \frac{n_s}{2} \right\rfloor + (n_u - n_s + \left\lfloor \frac{n_s}{2} \right\rfloor)n_d \leq n_u n_d$$

and since $-n_s + \lfloor \frac{n_s}{2} \rfloor \leq 0$, we can remove this expression from the left hand side without loss of generality to obtain

$$n_u - n_s + \lfloor \frac{n_s}{2} \rfloor \leq n_u$$

which is always true since, again, $-n_s + \lfloor \frac{n_s}{2} \rfloor \leq 0$ $\square$.

The PEG formulation is equivalent to the standard model when there are only

$\mathcal{U}$-constraints without $\mathcal{I}$ or $\mathcal{X}$-constraints. PEG and the standard model are not equivalent but do have the same problem size when there are $\mathcal{I}$-constraints but no friction involved.

4.8 Solvability of the new contact model

I can currently offer no formal proof of the solvability of the PEG formulation, however I can offer the observation that in practice I have successfully implemented and run PEG using the PATH solver with robustness and reliability. Any proof of solution existence for the general PEG formulation would be a significant mathematical undertaking that is beyond the scope of my current research, and certainly is to be considered future work. To give just a small amount of perspective on the magnitude of such an undertaking, David Stewart's work on convergence for formulations of rigid body dynamics with friction [79, 80] cites no fewer than eighteen works on the topic of solution existence spanning several decades of "controversy" on the topic, notably [16, 81, 82].

Chapter summary

In this chapter, we described a contact model called PEG which replaces the classical notion of sets of unilateral constraints with three fundamental constraints which can be written in the form of complementarity problems. These constraints can be used not only in conjunction to generate logical AND statements on unilateral constraints, but offer a mathematical framework for logical XOR statements between subsets of unilateral contacts. This framework allows us to incorporate all possible contacts into a time-stepping formulation in order to accurately capture the body geometries at those points of potential contact, fully representing the physically possible outcomes. We discussed how this contact model can be used in 2D and 3D, and with non-convex bodies by decomposing them into unions of overlapping convex bodies. We were also introduced to non-polytopal bodies that can also benefit from a PEG-like contact model. Lastly, we saw how PEG can be incorporated into the Stewart-Trinkle time-stepping method, and discussed solving this formulation.

CHAPTER 5
SIMULATION BEHAVIOR AND PERFORMANCE

In this chapter, we will see the improvements offered by the Polytope Exact Geometry (PEG) model over other methods in a variety of benchmark simulations designed to test specific contact cases. We will start by observing certain instantaneous cases, or cases over a single time step. We will then proceed by analyzing simulations, using various methods, over many time steps.

The PEG contact model can be incorporated into the Stewart-Trinkle method by discretizing $\mathcal{U}$, $\mathcal{I}$, and $\mathcal{X}$-constraints using equation (2.25). At each time step ℓ, a time-stepping subproblem is constructed and solved to determine the kinematic state of dynamic bodies at the end of the time step $\ell + 1$. We will see comparisons between the new model and the standard model (SM), the locally non-convex model (LNC), and corrective methods.

5.1 Instantaneous cases

We have seen previously the pitfall of the standard-model trap as depicted in Figure 4.1. Because it is typical to only consider contacts with gap distances within a chosen epsilon value, it is consequently also possible for a vertex to approach a corner condition such as the standard-model trap from behind the trap. Consider the case depicted in Figure 5.1 where the vertex v_a is approaching a corner from inside one of the edge's half spaces. Let us consider the two gap distances ψ_1 of v_a

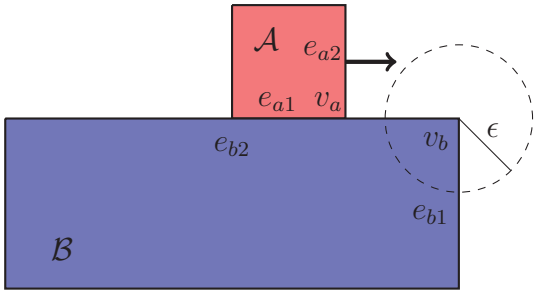


Figure 5.1: Two vertices approach one another. Note that v_a is approaching from a region of deep penetration of e_{b1}.

against e_{b1} and ψ_2 of v_a against e_{b2}. It is clear that ψ_2 is approximately zero as $\mathcal{A}$ slides left to right. It also should be clear that ψ_1 has a deep penetration. Using a standard contact approach, if we were to include both contacts $C_1 = \mathcal{C}(v_a, e_1)$ and $C_2 = \mathcal{C}(v_a, e_2)$, our dynamics formulation would attempt to correct C_1 by generating an immense force to the right at v_a and an equal and opposite force at the corner where e_{b1} meets e_{b2}. One could attempt to minimize this occurrence by using small values of epsilon, however this requires subsequently smaller time step sizes and only lessens the likelihood of this case, but does not eliminate it. Such an approach is not robust.

We should also consider that v_b is approaching a standard-model trap against the two nearest edges of $\mathcal{A}$. This dual trap will result in $\mathcal{A}$ "sticking" to the corner of $\mathcal{B}$ in the time step after v_a enters within ϵ of v_b. Notice that in Figure 5.2, v_a has

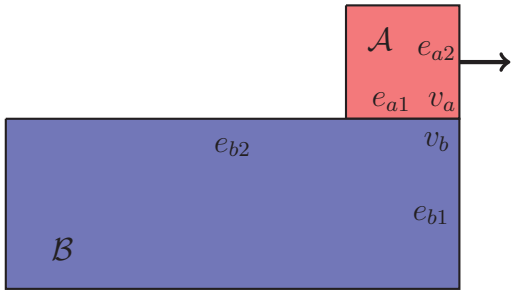
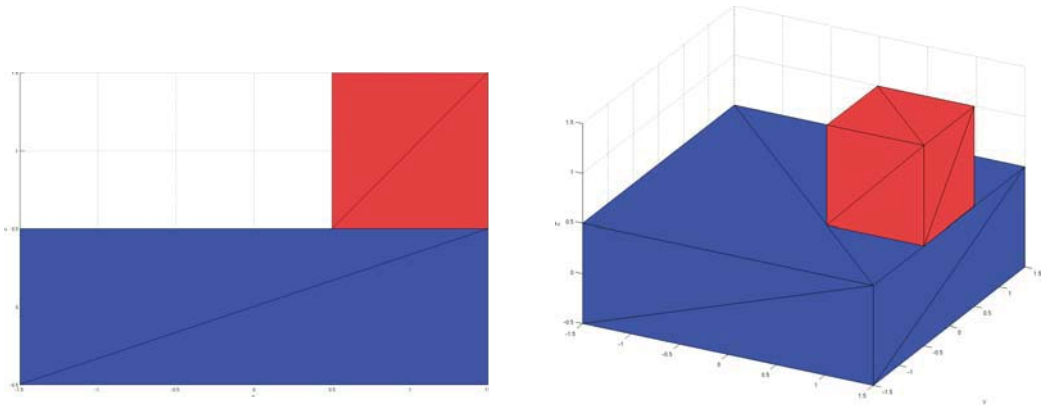


Figure 5.2: Dual standard-model trap where constraints between v_a with e_{b1} and e_{b2}, as well as constraints between v_b with e_{a1} and e_{a2}.

zero gap distance with the two nearest edges of $\mathcal{B}$, and v_b has zero gap distance with the two nearest edges of $\mathcal{A}$. Body $\mathcal{A}$'s, progress to the right will be halted by the constraint of v_b against e_{a2}, since this constraint would become negative if $\mathcal{A}$ moved any farther right.

We can generate similar standard-model traps for vertices against multiple faces in 3D. In Figure 5.3, a red cube is on top of and near the side of a larger blue rectangular body. Two vertices on the bottom right of the red cube in 5.3a are near two faces of the blue body, a top face and a side face. The analysis of this configuration is nearly the same as for the 2D case. If only the contact with the top face is enforced, then the vertex is stopped from potentially traversing the free space below the top face and to the right of the side face. Conversely, if only the



(a) Side view. **(b) Angled view.**

Figure 5.3: Example of 3D standard-model trap for vertices against multiple faces (bottom right vertices of red body in (a)). Given this configuration, what contacts should be generated to produce accurate interaction of these bodies?

contact with the side face is enforced, then the red body is erroneously prevented from sliding left, but worse there is nothing to stop the right side of the red body from falling into the blue body. Enforcing both contacts as unilateral constraints does not improve the situation, as although it prevents interpenetration, it traps the vertices in the convex free space defined by the union of the faces' half spaces. Using PEG to model this configuration, we would include all possible contacts since PEG finds a solution from among the physically feasible combinations of these contacts.

Figure 5.4 depicts a configuration between two tetrahedra in which the stability is sensitively dependent on any heuristic choices made regarding vertex-face contact. The bottom blue body is static, pinned to the world frame. The top red body is

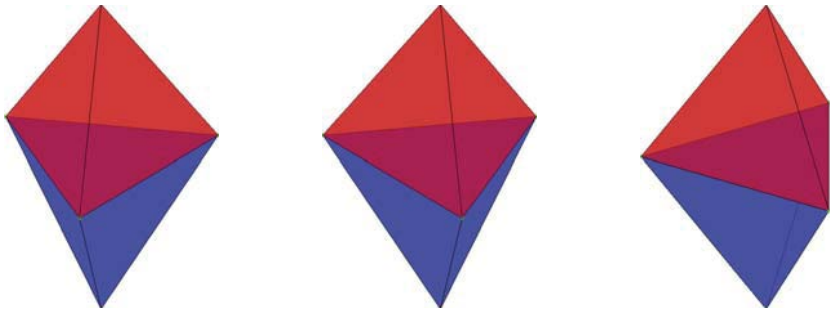


Figure 5.4: Multiple views of a tetrahedron stacked and aligned atop a second similar but inverted tetrahedron. Which contacts should be enforced in this configuration?

dynamic and affected by gravity. Consider any of the corners where there is contact, *e.g.,* the two nearest vertices in the center image of Figure 5.4. We will refer to the vertex of the top body as v_a and the three faces it is near as v_{b1}, v_{b2}, and v_{b3}. We then consider the three contacts

$$C_1 = \mathcal{C}(v_a, v_{b1})$$
$$C_2 = \mathcal{C}(v_a, v_{b2})$$
$$C_3 = \mathcal{C}(v_a, v_{b3})$$

that could be enforced. Including all of these contacts, as well as the similar contacts with other vertices between these bodies, as unilateral constraints with Stewart-Trinkle would effectively fuse these two bodies in this configuration. So what heuristic could we employ to choose a subset of the possible contacts? Is there any subset that would result only in physically realistic interaction of these bodies?

A heuristic which only considers the value of gap distances is not robust. Consider that the gap distances for C_1, C_2, and C_3 are all near zero, approximately equal, and vary by numerical errors due to both floating point error and solver convergence error. So, choosing for example to enforce the contact with the smallest non-negative gap distance would result in unpredictable behavior.

Using applicability as a heuristic in this configuration would at least be consistent regarding which contacts where included, since only the vertex-face contacts between vertices and the faces which are touching have applicability. However, this configuration is at the boarder of classical applicability and would degenerate unless we use the geometric relaxation presented in 3.2. Even with this heuristic, we are not achieving physical fidelity by including a subset of the potential contacts. In fact, this heuristic creates a partial standard-model trap by preventing any vertices of the the top body from penetrating the top face of the bottom body, and similarly preventing any vertices of the bottom body from penetrating the bottom face of the top body. Further, it is not clear that this heuristic would be any more successful for other configurations.

PEG, with its ability to accurately model the geometry of contact, is necessary if one hopes to accurately simulate the physically feasible interactions of these bodies

for configurations such as this, and ones like it.

5.2 Simulation benchmarks

In this section, we look at several examples of two or more bodies in simulation over hundreds or thousands of time steps and make observations on accuracy and stability for different models. We will particularly observe how the behavior of these models changes (or doesn't change) as we vary the size of the time step h. I try to focus here are cases that approach the border of what is poorly or well defined in traditional methods, *e.g.,* vertices near other vertices. Simulations were frictionless and the popular PATH solver [40] was used for all simulations unless stated otherwise.

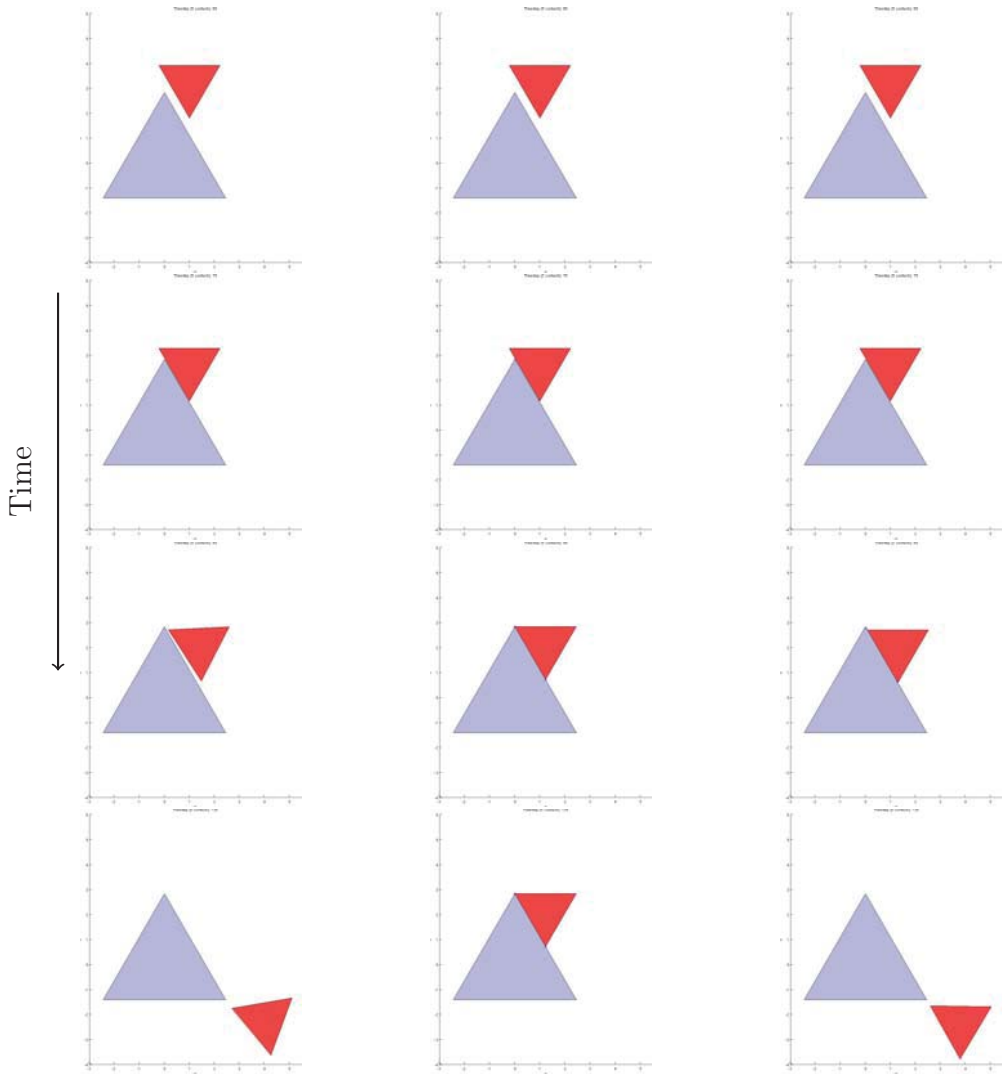
Three methods that we will compare throughout the section are a corrective method (essentially a modified penalty method), a standard approach to contact with the standard model, and PEG. The corrective method considers only contacts that are violated. Corrective forces for this method are not determined in the traditional way of a penalty method by directly treating each contact as an independent linear spring system, but are modeled as unilateral constraints and solved for simultaneously using PATH. This approach makes this corrective method more competitive in terms of the errors that we will measure in the various simulation experiments. There is no correlation between solver convergence and these errors, since the solver has no sense of what it is solving when attempting to satisfy sets of constraints. Particularly with PATH, if the constraints cannot be solved then an error is returned and the simulation crashes. In other words, for all simulations that completed, constraints were satisfied within solver convergence criteria. The contact identification used with the standard model simply includes all contacts within the ϵ value for that experiment. The contacts for PEG are identified in precisely the same way as for the standard model, the difference of course is how these contacts are incorporated into a dynamics problem.

The MATLAB code for all of these experiments is available as a branch to the RPI-MATLAB-Simulator at `http://code.google.com/p/rpi-matlab-simulator/source/browse/#svn%2FPEG_branch`.

5.2.1 Triangle-drop experiment

In this benchmark, a triangle with side length 2.45 m, mass of 1 kg, and rotational inertia of 0.6495 kg/m^2 is dropped from above and to the right of a larger static triangle of side length 4.9 m. This experiment was run for time steps ranging from 0.001 to 0.016 s over a period of 1.5 s of simulation without friction. For each time step h, an epsilon of $\epsilon = 10h$ m was used for collision detection.

Figure 5.5 depicts trials for three different models applied to the triangle drop simulation for $h = 0.01$ s. In the first two frames (top two rows), at time steps 60 and



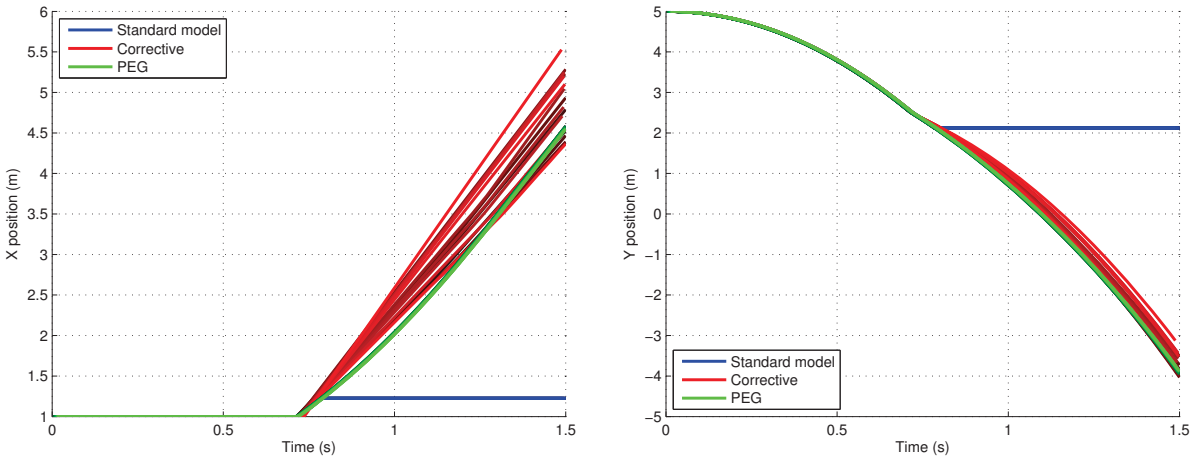
(a) Corrective method. **(b) Standard model.** **(c) PEG.**

Figure 5.5: Sample comparison of methods on triangle drop benchmark for $h = 0.01$ s.

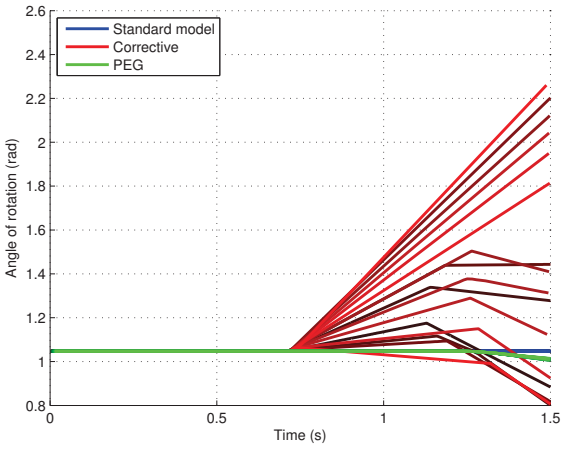
80 before contact occurs, all three methods are identical as they fall under gravity. In the third frame, all three methods diverge. The corrective method encountered an interpenetration and generated a force to correct it, resulting in a moderate bounce off of the bottom triangle. With the standard model, the left most vertex of the red body has entered a standard-model trap against the top two edges of the bottom body. With PEG, although the same set of contacts were identified as with the standard model, we see the red body slide through the trap and along the right face of the bottom body. In the final frame, the standard model is still trapped, while the smaller triangles in the other two methods have fallen off of the larger one.

Consider Figure 5.6 which shows the x and y trajectories, as well as the rotation of the red triangle as it falls for the various values of time step size (brighter colors correspond to larger time steps). A ground truth simulation was run using PEG at a time step of $h = 0.0001$ s. The standard model consistently diverges in x and y as soon as the vertex becomes trapped. All 16 trials of the PEG model overlap and cover the the ground truth on all three plots. The corrective method varied dramatically for different time steps.

It is tempting, when comparing the behavior of PEG with the corrective method in Figure 5.5, to give the corrective method more credit than it deserves, as it appears to qualitatively resemble the trajectory of the PEG simulation. However, PEG is able to recreate the same trajectory for even very large time steps, where the results from a corrective method vary considerably with changing time steps and require time steps which are multiple orders of magnitude smaller to achieve quantitatively similar behavior to PEG.



(a) x position **(b) y position**



(c) rotation

Figure 5.6: Component trajectories of the red triangle in the triangle drop experiment for three different models over time steps from $h = 0.001$ s to $h = 0.016$ s.

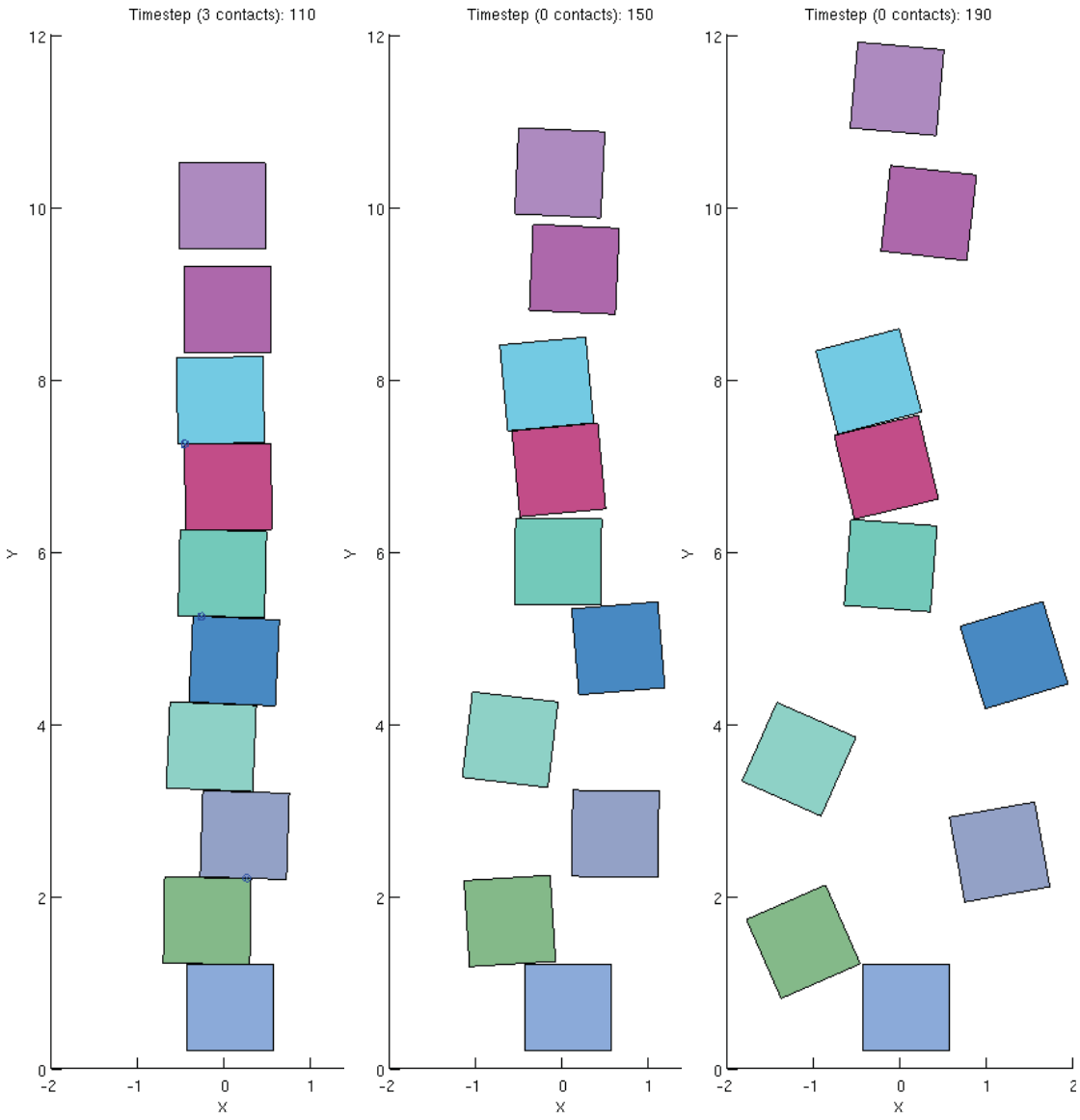
5.2.2 Stack of boxes

In this benchmark simulation, a stack of ten staggered squares of 1 m^2 is simulated with a time step of $h = 0.005$ s and $\epsilon = 0.07$ m. The initial positions of the boxes were generated randomly in the horizontal direction between values of ± 0.1 m, and separated by 0.25 m in the vertical direction. The bottom box is a static body. The physical result would be for the boxes to fall and stay stacked, as they are not so staggered to place their centers of mass off of the bottom box. We will observe three cases, the first of which is a corrective method.

Figure 5.7 shows three frames from the simulation using a corrective method. At step 110, all boxes but the top have fallen and made contact with the boxes below. We can already see some unexpected rotations in the boxes toward the bottom. At step 150, the momentum of the boxes has generated interpenetrations as the boxes fell from gravity, and we see the boxes separating as they rebound from this violation of contact constraint. It is already apparent that this stack will not recover from this rebounding. In fact, even if it did, the stack could not be considered stable, an unfortunate consequence of corrective methods. At step 190, we see the boxes as they start to fall away after their rebound. Eventually, the static bottom box is the only box that remains.

Figure 5.8 depicts the results of using the standard model with standard contact identification. The simulation appears to be behaving well at step 110. At precisely step 120, the top box has come close enough to the box below it, and the vertices of these two boxes are close enough to one another, that we see a quadruple standard-model trap occur. The horizontal forces generated by these false contacts shifts the box second from the top to the left, which in results in a sort of domino effect in which the top six boxes all enter quadruple standard-model traps with at least one of their neighbors, which we see at step 130. Although this stack does not topple in this particular case, the behavior is clearly non-physical and undesirable.

Figure 5.9 demonstrates the stability and physical behavior of PEG. At step 110, eight of the ten boxes are in contact while the last two boxes are about to make contact. At step 130, all boxes are in contact with gap distances of zero (within machine precision) at all active contacts. Thousands of steps later, the box stack is

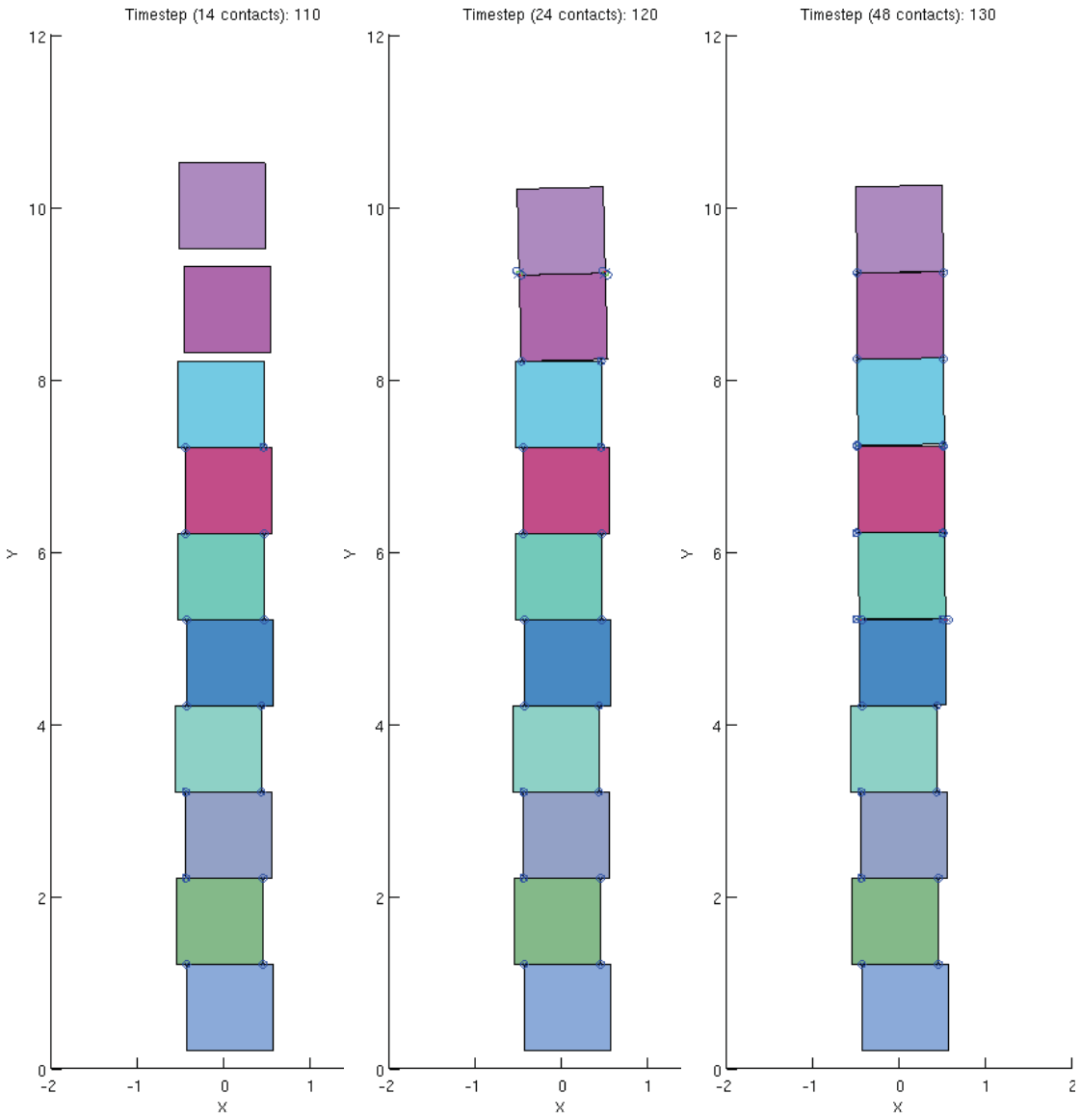


(a) Step = 110 **(b) Step = 150** **(c) Step = 190**

Figure 5.7: Results of 2D box stack with corrective method.

steady and stable.

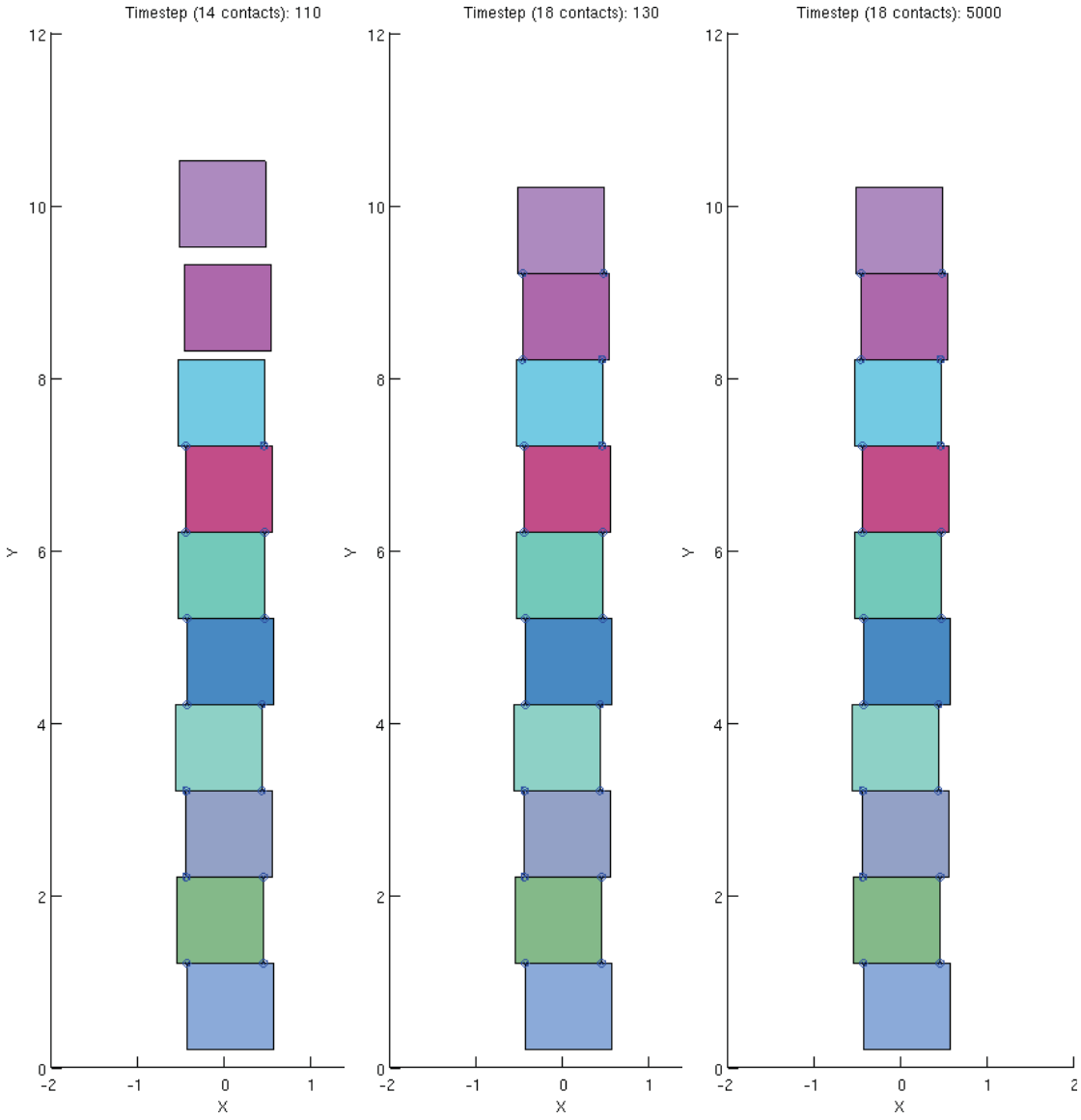
A question arises as to what modifications could be made to either the corrective or standard models in order to improve their performance. Although the corrective method waits for contacts to be violated before adding them to the active set, once in this set, contacts could be cached so as to not allow violation on successive steps. Although this technique, which is employed by Bullet Physics, does curtail instabilities for certain cases, it would not improve performance for simula-



(a) Step = 110 **(b) Step = 120** **(c) Step = 130**

Figure 5.8: Results of 2D box stack with standard model.

tions such as the box stack experiment where the initial interpenetrations generate forces which undermine the stability of the interdependent body configurations. As we follow a line of thought that has generated many of the *ad hoc* corrections common to simulators today, the natural next idea is to dampen the forces generated by deep penetrations [21, 83] in the hope of mitigating the resultant instabilities. Such corrections may be well suited for certain applications but I'll reiterate that they only treating symptoms, and can always be broken.



(a) Step = 110 (b) Step = 130 (c) Step = 5000

Figure 5.9: Results of 2D box stack with PEG. Although the same set of contacts is identified as with the standard model, especially the top right most contacts, PEG does not enforce erroneous contact and is subsequently stable where other models fail.

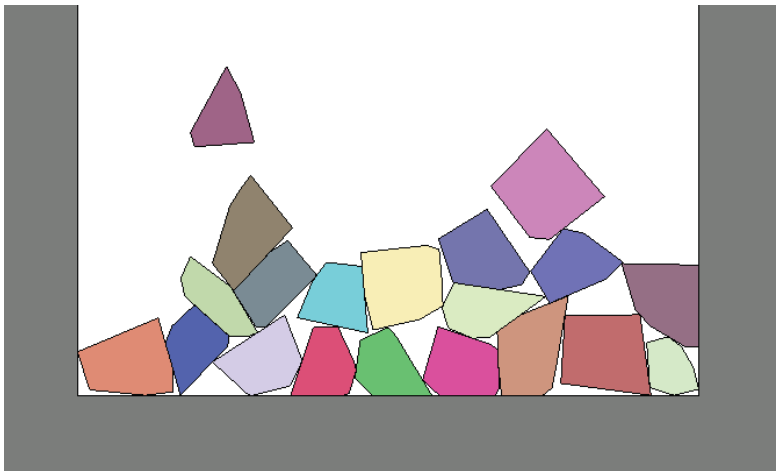
5.2.3 Box of polygons

Figure 5.10a depicts an experiment in which we simulate five seconds of pouring polygons into a container. Each polygon was generated by taking the convex hull of ten randomly generated points between values of ± 0.25. SM and LNC were unable to simulate the full five seconds as they were prone to enter non-physical configurations or generate contact sets which were unsolvable using PATH or resulted in extreme instability and "exploded" using other solution approaches such as Lemke's algorithm [41].

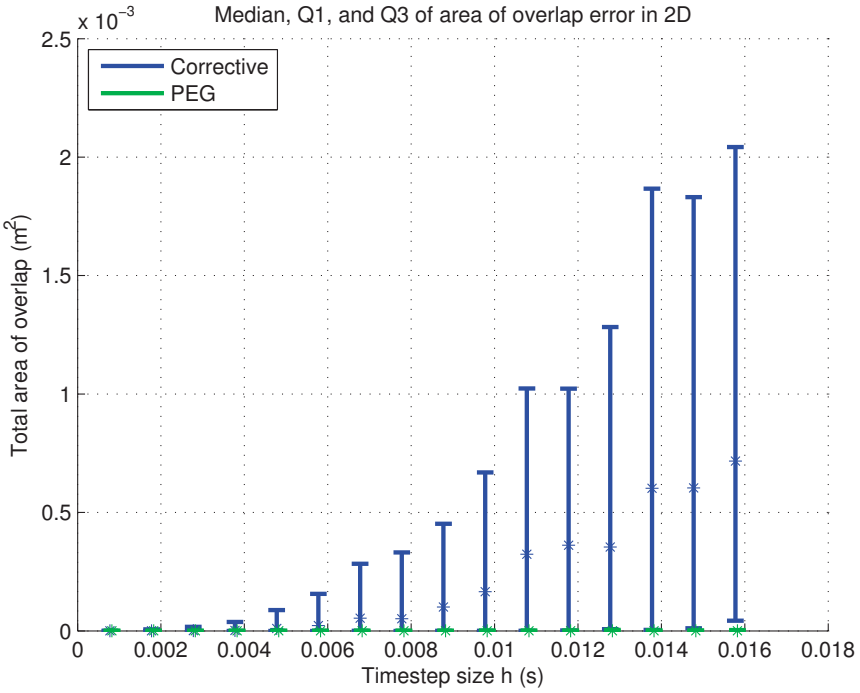
Figure 5.10b shows the median and first and third quartile of area of overlap error for ten different sets of random bodies, for increasing time step, of our model compared to a corrective method where the error is measured by the total area of overlap between all body pairs. The corrective method used here identifies areas of overlap per body pair, determines a single contact per overlap, and applies impulses to reduced the interpenetration to zero at the end of the time-step. Although this method introduces instability in the form of "bounce" between objects, it is competitive in terms of the area of overlap error metric. Even for small time steps, PEG produced errors orders of magnitudes smaller than the corrective method for this experiment.

Figure 5.11 demonstrates a body configuration and contact set that resulted in a failure to find solution with the standard model. The failure occurred due to there being three contacts that could not be simultaneously satisfied. The first contact involves the right most vertex of the green body against the right edge of the red body (incidentally the green body's vertex is trapped against the top most vertex of the red body). The second and third contact both involve vertices at the lower left of the green body against the left most edge of the red body. Essentially, the green body must move left relative to the red body but cannot.

To reiterate this idea with a contrived but simpler example than that which occurred in the box of polygons experiment, consider the basic configuration of Figure 5.12 in which a smaller cube is stacked upon a larger one. There are two contacts which have been identified between bodies $\mathcal{A}$ and $\mathcal{B}$: one between the bottom left vertex of $\mathcal{A}$ with the left edge of $\mathcal{B}$, and one with the bottom right vertex



(a) An example 2D experiment. A polygon is dropped into a container every 0.25 seconds for 5 seconds of simulation.



(b) Comparison of area overlap errors for 2D polygon experiment. Predictably, the corrective method generates larger error for larger time steps. The error of PEG is relatively imperceptible, set along the horizontal axis for all time steps tested.

Figure 5.10: Simulation experiment with several polygons in 2D.

of $\mathcal{A}$ with the right edge of $\mathcal{B}$. Both of these contacts, once identified during some mid-phase collision detection, would be considered by the narrow-phase and could

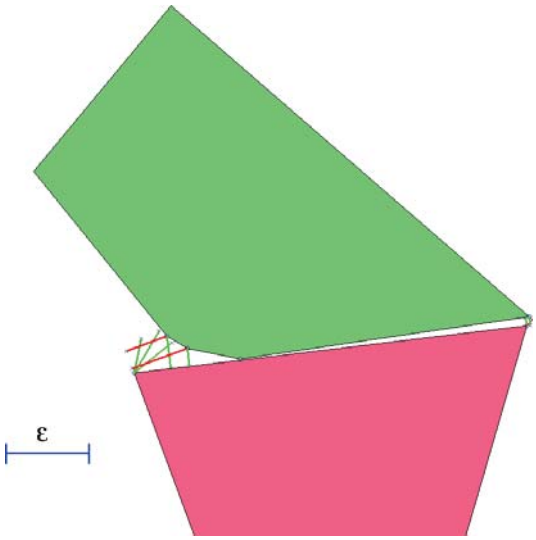


Figure 5.11: Failure to find solution with standard model. Red lines correspond to negative gap distance of contacts, and green lines to positive gap distances. Contacts on the left and right sides of the bodies cannot be simultaneously satisfied. Smaller time steps allow for smaller epsilons, reducing this occurrence, but not eliminating it.

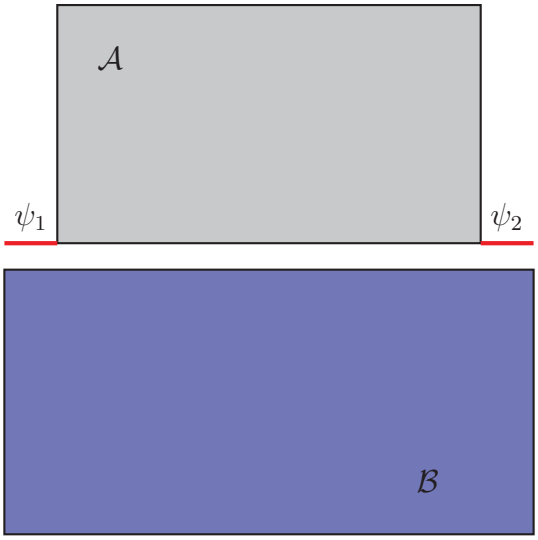


Figure 5.12: Two contacts have been identified: one on the left of $\mathcal{A}$ with negative gap distance ψ_1, and one on the right side of $\mathcal{A}$ with negative gap distance ψ_2. There is no solution that can move $\mathcal{A}$ both relatively left and right of $\mathcal{B}$ simultaneously.

both be included during the formulation of the time-stepping problem. However, the negative gap distance ψ_1 on the left requires $\mathcal{A}$ to move left relative to $\mathcal{B}$ in

order to be "corrected," and the negative gap distance ψ_2 on the right requires $\mathcal{A}$ to move right relative to $\mathcal{B}$. Clearly, this contact set in this configuration cannot be satisfied, and attempting to solve the dynamics formulation will fail and possibly crash the simulator.

An interesting statistic to observe is the rate of occurrence of the SM-trap. In other words, how frequently is PEG necessary to achieve a geometrically accurate interaction between bodies. Figure 5.13 depicts the number of SM-traps encountered at each time step for the 2D experiment above.

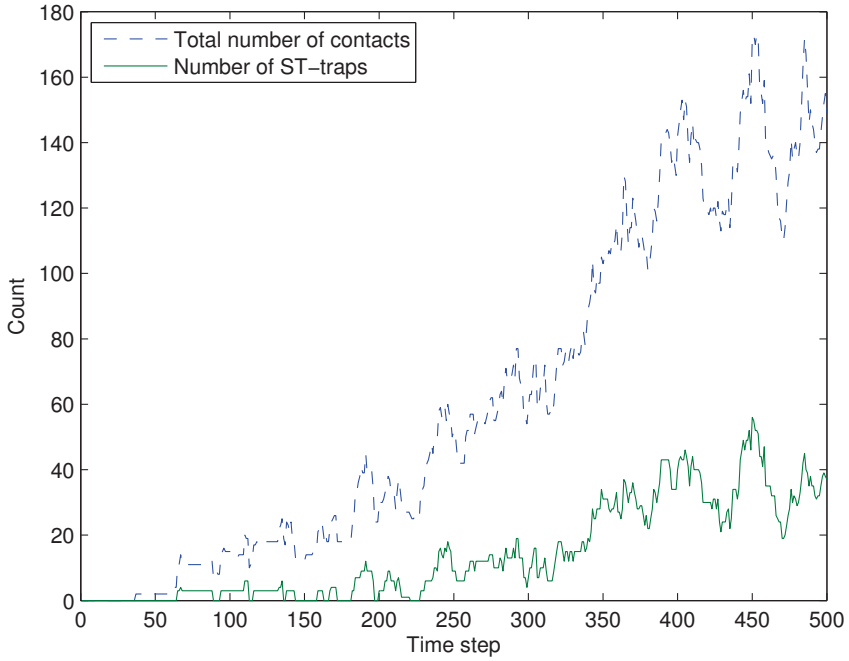
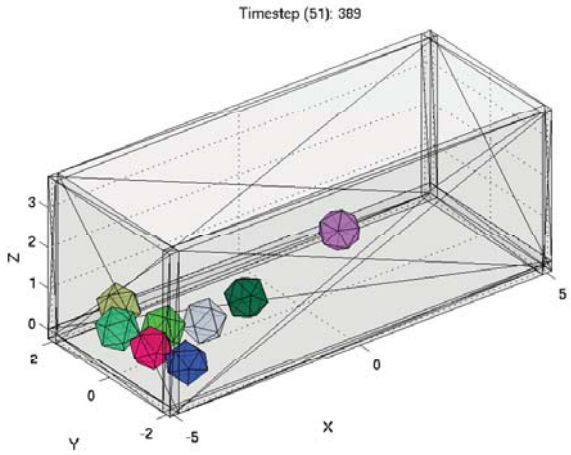


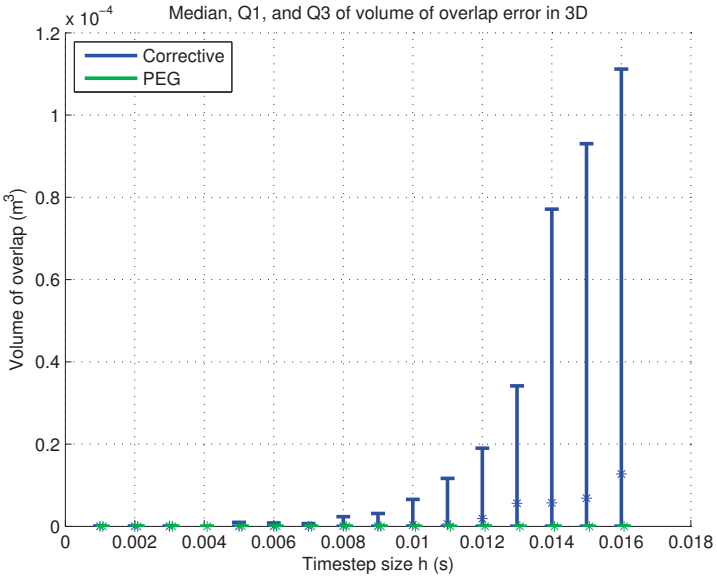
Figure 5.13: Number of contacts and number of SM-traps encountered at each time step for an example 2D simulation experiment with time step of $h = 0.01$ s. Smaller time steps generate fewer SM-traps since the ϵ used are smaller.

5.2.4 Box of polyhedra

A similar set of simulation experiments was run for PEG and the corrective method in 3D, an example of which is shown in Figure 5.14a. Every 0.5 seconds, a polyhedron was dropped into the box with a horizontal velocity in the negative x direction. The results, depicted in Figure 5.14b, are extremely similar to the 2D results. As the step size increases, the interpenetrations grow steadily larger



(a) An example 3D experiment. A polyhedron is dropped into a box container every 0.5 seconds for 5 seconds of simulation.



(b) Results for 3D polyhedra experiment.

Figure 5.14: Simulation experiment with several polyhedra in 3D.

with a corrective method, whereas PEG virtually eliminates this error at all time steps. In addition to this similarity between 2D and 3D, we also see a similar rate of occurrence of the SM-trap for this particular experiment at the step size of $h = 0.01$ s (Figure 5.15). Although this similarity is intriguing, it is simply coincidence, as these results are only representative of a single experiment at a single time step and 2D, and a single experiment at a single time step in 3D.

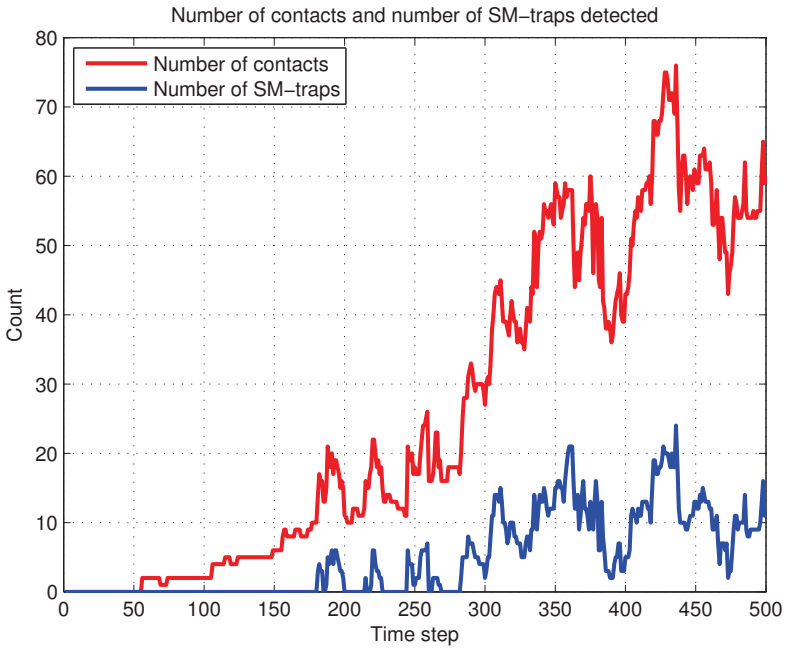
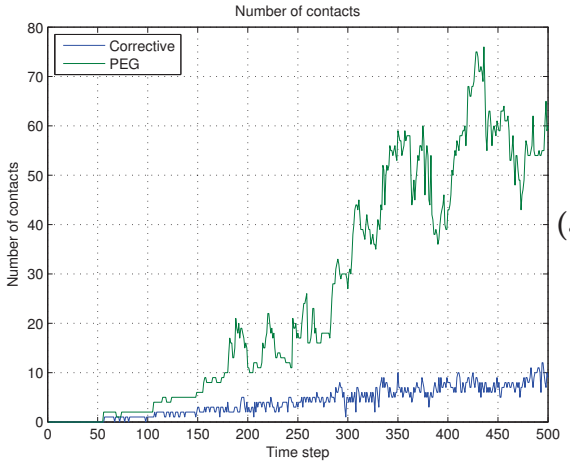
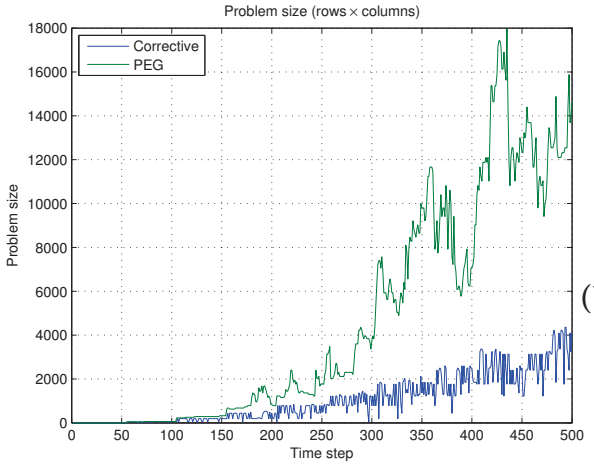


Figure 5.15: Number of contacts and number of SM-traps encountered at each time step for an example 3D simulation experiment with time step of $h = 0.01$ s.

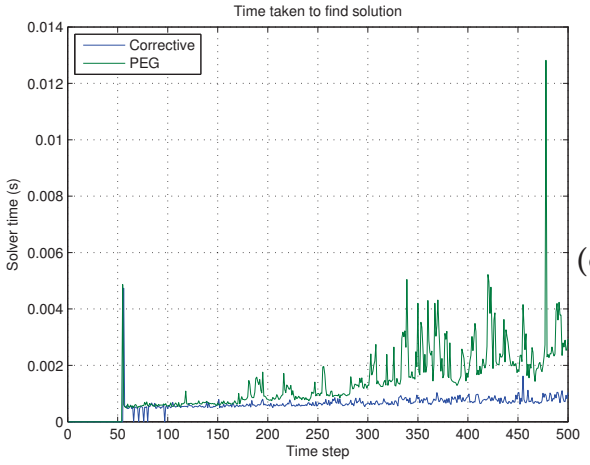
Although accuracy is paramount, practicality dictates the importance of speed as well. Figure 5.16 compares several statistics between PEG and the corrective method for the 3D experiment with regard to problem size and solver time. For this experiment, the mean solution time for the corrective method was 0.52342 ms, and the mean solution time for PEG was 0.96736 ms.



(a) The number of contacts in the corrective method corresponds roughly to the number of bodies since each body has relatively consistent contact with the bottom of the box and inter-body contacts do not last long.



(b) The size of the matrix representing the time-stepping problem was roughly 4 times larger on average for PEG than for the corrective method.



(c) The amount of time that PATH took to solve PEG averaged 1.85 times that to solve the implicit formulation for the corrective method. This is an excellent performance for PEG.

Figure 5.16: Comparisons of PEG versus corrective method for the 3D box experiment over 5 seconds with $h = 0.01$ s.

5.2.5 2D grasp experiment

Figure 5.17 shows a sequence from a 2D grasping experiment. In this experiment, a five-link two-fingered gripper grasps a randomly generated convex polygon sitting on a surface.

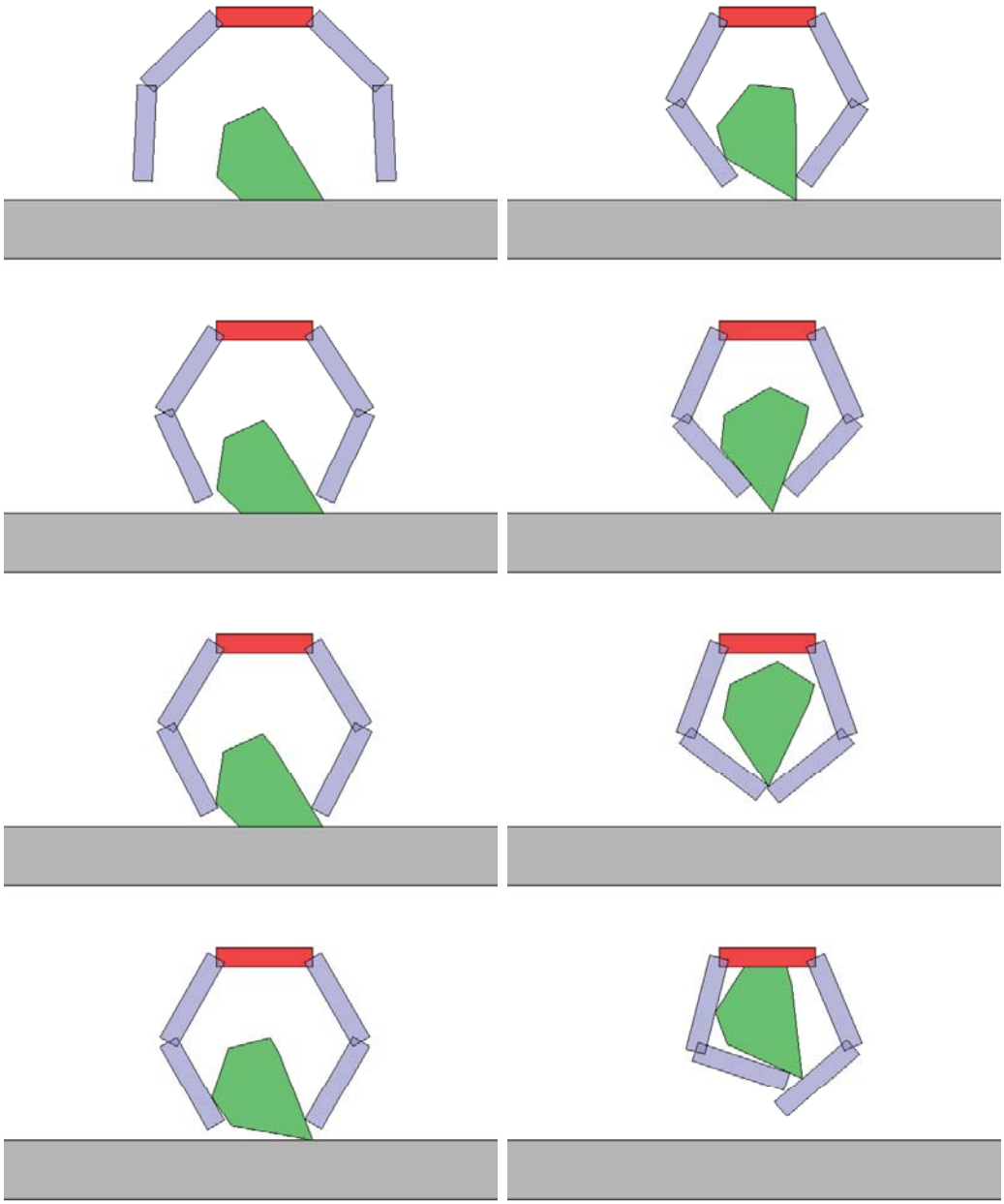


Figure 5.17: Sequence (top to bottom, left to right) of a 2D grasping experiment with a PD controlled two-fingered gripper.

The same three models were compared as in the other experiments. For each model, ten trials were run over multiple time step sizes from $h = 0.001$ s to $h = 0.01$ s, where each of the ten trials corresponded to a randomly generated convex polygon. Each method was tested on the same set of polygons. For each of these trials, the position at each step was compared to the position from a "ground truth" experiment which was run with a very small time step of $h = 0.0001$ s. For now, the rotation is ignored, and the positional error is determined by the Euclidean distance between the ground truth and the position at the corresponding step of each trial. Figure 5.18 shows results from this experiment using the first polygon, where lighter shades of a color correspond to trials with larger time steps.

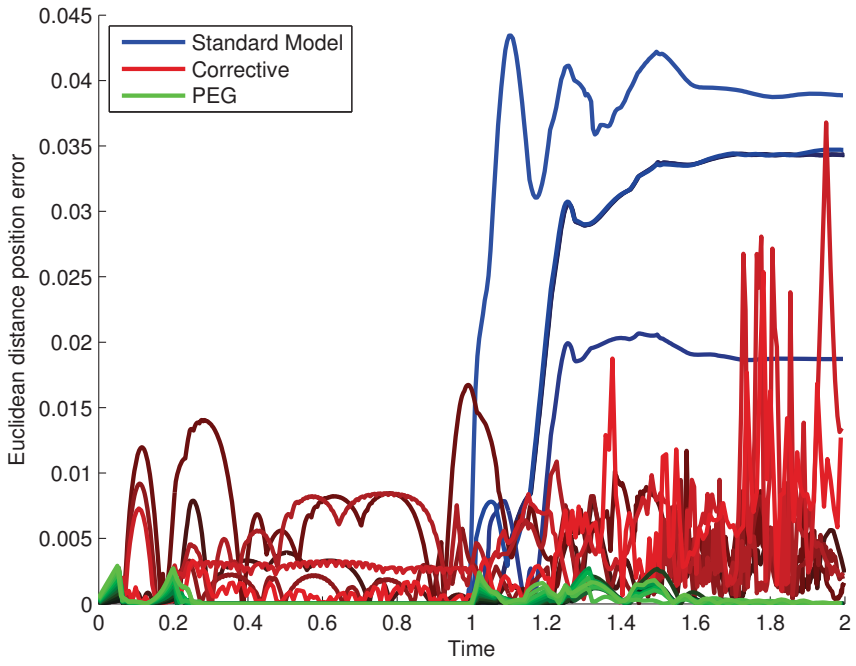


Figure 5.18: Sample results from the first 2D grasp experiment.

The standard model tends to generate large errors starting near the time at which the fingers make contact with the object. This is due to the vertices at the finger tips entering into the standard-model trap against vertices on the surface of the object. For example, consider Figure 5.19 where both finger tips are trapped against the object. Although this "grasp" will remain in this configuration stably, it is clearly non-physical as the finger on the right is actually above the object but

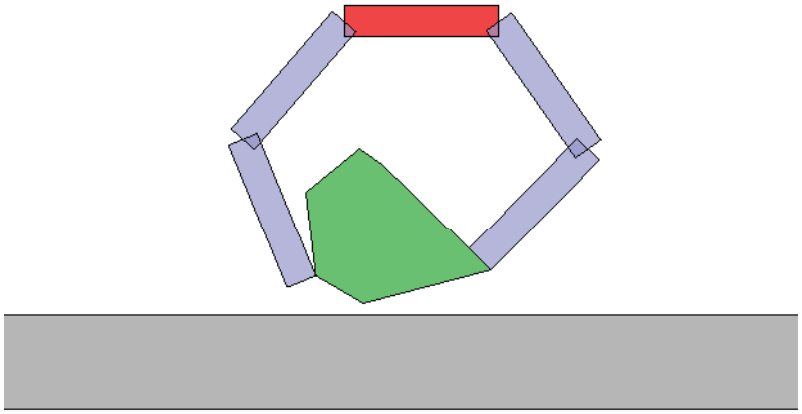


Figure 5.19: A non-physical grasp due to fingers entering into standard-model traps near vertices of the object.

still applying a force upward on the nearest bottom face of the object. The finger on the left is also trapped where we can clearly see vertex-vertex contact.

Predictably, the corrective method performs poorly in the 2D grasp experiment, as it has persistent bounce and never settles into a stable configuration. Some trials for h greater than 0.003 s are not shown for the corrective method in Figure 5.18, as it was mostly unstable above this time step and the object being grasped tended to "explode" out of the gripper due to deep contact.

PEG performs best of the three models tested, even for relatively large time steps. Additionally, even though PEG has small deviations during times when dynamics are changing, *e.g.,* initial contact, these deviations are temporary and all PEG trials converge to the ground truth.

Trials for all three models are prone to increasing error as the time step increases since the proportional-derivative controller used to control the four joint torques of the gripper had constant gains over all trials, and the controller becomes less stable with increasing time step size. Although this could be improved by implementing gravity compensation and employing gain tuning, it does not affect the relative results of the grasping experiment.

The 2D grasp experiment of Figure 5.17 was repeated for ten different randomly generated polygons over time steps from $h = 0.001$ s to $h = 0.01$ s. The

median and first and third quartiles of the positional error for three models at each time step is presented in Figure 5.20. Trials for the standard model entered con-

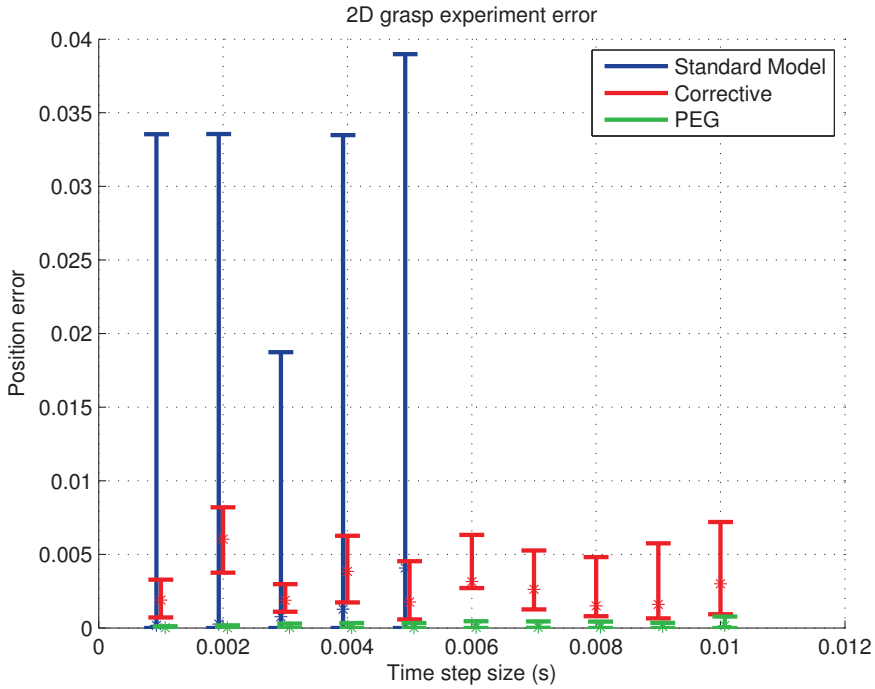


Figure 5.20: Median, Q1, and Q3 of position error over ten trials for different time steps in the 2D grasping experiment.

figurations without solution for time steps above $h = 0.005$ s, so these trials do not appear in in the data. Such configurations were conceptually similar to that of Figure 5.11.

5.2.6 2D grasp experiment with friction

Precisely the same experiment was performed as in 5.2.5, but this time a friction coefficient of 0.25 was used between all bodies. Looking at the results in Figure 5.21 for the first trial object grasped, we see that all three models performed similarly as in the experiment without friction. One notable difference is that although PEG converges to the ground truth for the smallest time steps, it consistently converges to a slightly different position for larger time steps. This has to do with the timing of the fingers closing and whether or not certain contacts were sticking or slipping with friction.

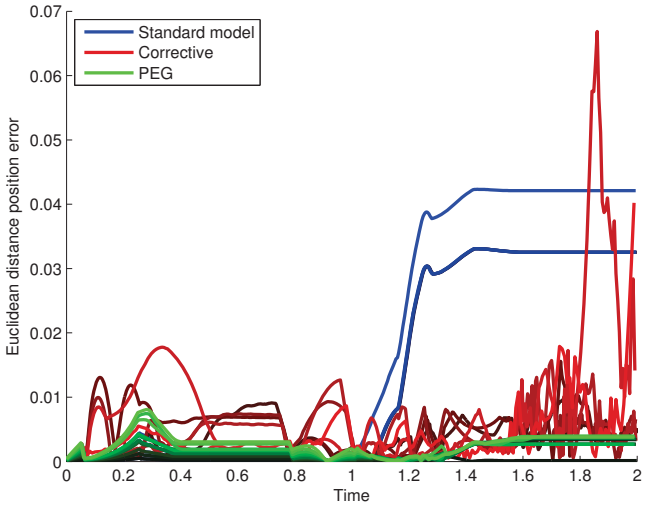


Figure 5.21: Sample results of 2D grasp experiment with friction for first polygon.

Figure 5.22 shows results for all ten trials for each of the three models for time steps for ten different time steps. The results are quite similar to the grasping experiment without friction. This time, however, the standard model was able to make it to $h = 0.008$ s without crashing, and the corrective method was unable to complete trials for $h = 0.01$ s.

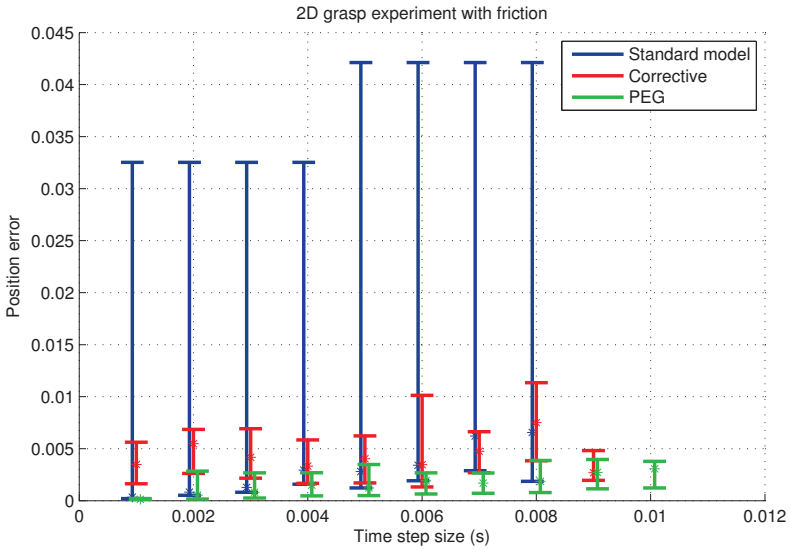


Figure 5.22: Median, Q1, and Q3 of position error for all trials of the 2D grasping experiment with friction.

Videos of experiments

The following videos are presently available online. These videos include experiments from this section as well as additional demonstrations comparing PEG to other methods.

2D

- Box of polygons - PEG `http://www.youtube.com/watch?v=Lq_O4CED-ks`(Date Last Accessed, June, 30, 2014)
- Staggered box drop `http://www.youtube.com/watch?v=JXKK6Xlwk2U`(Date Last Accessed, June, 30, 2014)
- Grasp experiment `http://www.youtube.com/watch?v=pOfJ3D3ebc8`(Date Last Accessed, June, 30, 2014)
- Triangles - SM `http://www.youtube.com/watch?v=pi0MoQuo5rA`(Date Last Accessed, June, 30, 2014)
- Triangles - Corrective `http://www.youtube.com/watch?v=Ju38NdDAKQI`(Date Last Accessed, June, 30, 2014)
- Triangles - PEG `http://www.youtube.com/watch?v=VJYNzZr66XI`(Date Last Accessed, June, 30, 2014)
- Triangles - LNC `http://www.youtube.com/watch?v=igGXULnmwoQ`(Date Last Accessed, June, 30, 2014)

3D

- PEG - Icosahedra `http://www.youtube.com/watch?v=Q3iWrmk7K-U`(Date Last Accessed, June, 30, 2014)
- PEG - Sliding vert-edge `http://www.youtube.com/watch?v=VbFDdHhUJzQ`(Date Last Accessed, June, 30, 2014)
- PEG - vertex-edge `http://www.youtube.com/watch?v=wfa1su7aBFQ`(Date Last Accessed, June, 30, 2014)
- PEG - vertex-vertex `http://www.youtube.com/watch?v=ewgOAkXWdGE`(Date Last Accessed, June, 30, 2014)

5.3 Rigid body rotation and implicit time-stepping

We notice in Figures 5.10b and 5.14b that PEG tremendously outperforms the popular corrective method. However, if we look closely at the error for PEG in Figure 5.23, we observe that it does increase as the time step increases. Why is this

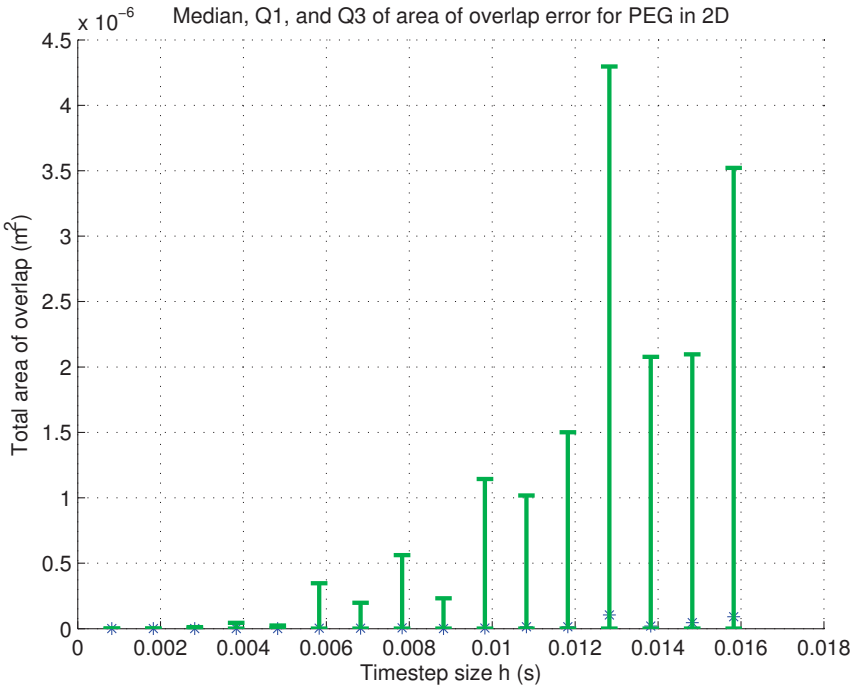


Figure 5.23: Area of overlap error for PEG in 2D experiment. Although these small errors do appear to increase with time step size, we will see that this is a consequence of the time-stepping method used, and not of PEG.

the case since PEG should not allow interpenetration, regardless of the time step size? The answer is that the time stepping method we incorporated PEG into, in this case Stewart-Trinkle, is a semi-implicit method and is therefore susceptible to error due to rotations over the time step. To understand the source of this error, we must look at the time stepping method at the lowest level.

Polytope overlap

The points of contact at the surface of each body, as well as the contact normal directions, are identified and determined at the beginning of the time step. The gap

distances for the constraints written in terms of this contact information is part of the solution to the time-stepping problem which is solved for at the end of the time step. Because of this time discrepancy of when these values are determined for, a subsequent error can manifest due to rotations and sliding during the time step.

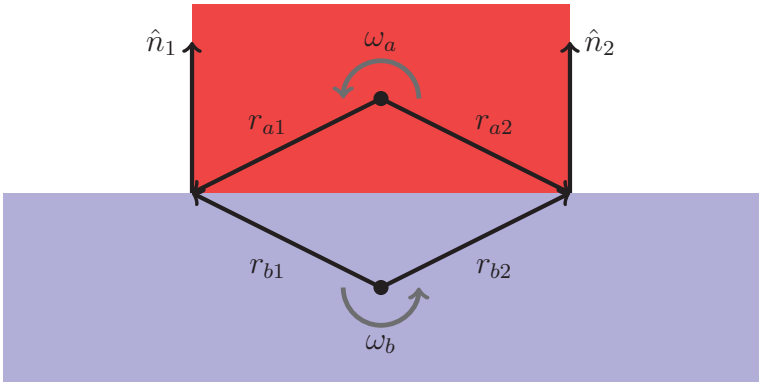
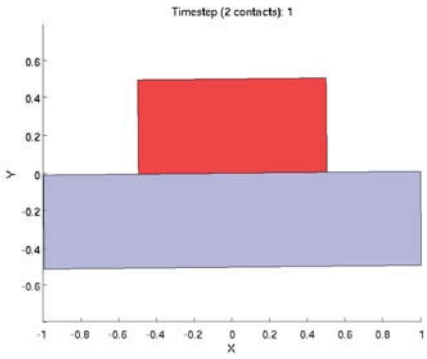


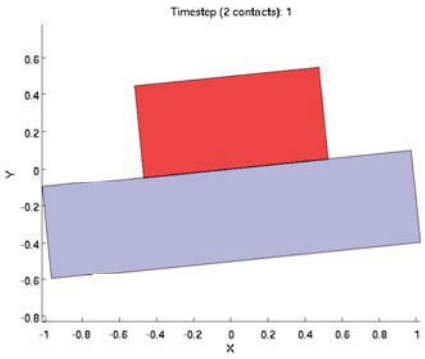
Figure 5.24: Initial configuration of two rectangular bodies with nonzero rotational velocity. Over the next time step, both bodies will rotate due to their rotational velocities as well as their interactions with one another. However, the contact frames do not rotate since Stewart-Trinkle is semi-implicit.

Consider Figure 5.24, in which two rectangular blocks are horizontal and in contact at the beginning of the time step. The contact normals are given by $\hat{\mathbf{n}}_1 = \hat{\mathbf{n}}_2 = \begin{bmatrix} 0 & 1 \end{bmatrix}^T$. Figure 5.25 depicts the final position of both bodies at the end of a single simulation step for various step sizes where both bodies have rotational velocities of $\omega = 1$ rad/s. Gravity and friction are not considered. In the initial configuration, two contacts are identified: one with each of the bottom vertices of the top body with the top edge of the bottom body. The combination of the contact information having been determined at the beginning of the time step, with the rotation *and* relative sliding of the bodies, results in the configurations depicted, which clearly have larger error corresponding to larger time steps. Equivalently, larger rotational velocities could result in larger error for the same time step.

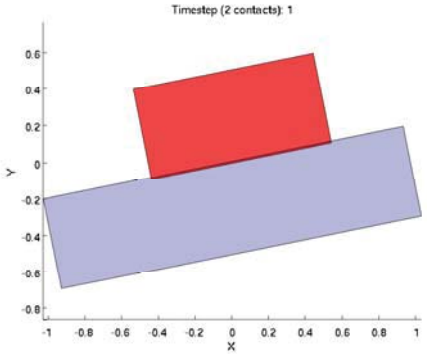
Although a more subtle error, it is also possible if there were additional external forces for the constraints to be over satisfied as opposed to the under satisfied which we saw in Figure 5.25. For example, if the top body were forced to slide left while the two bodies undergo the same rotations, it would be constrained with similar



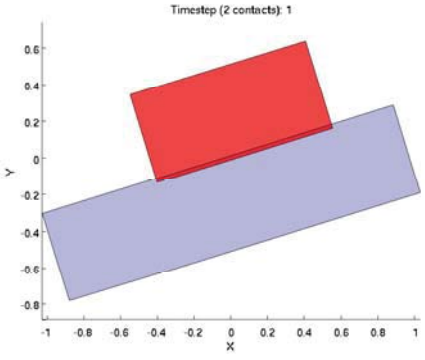
(a) h = 0.01, area error of 0.000025



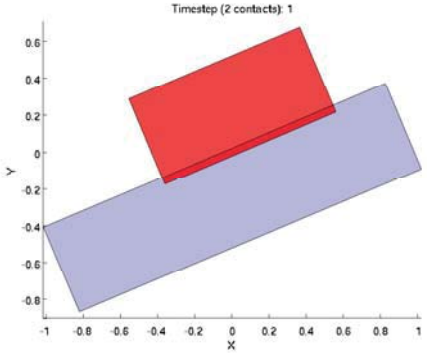
(b) h = 0.1, area error of 0.0025



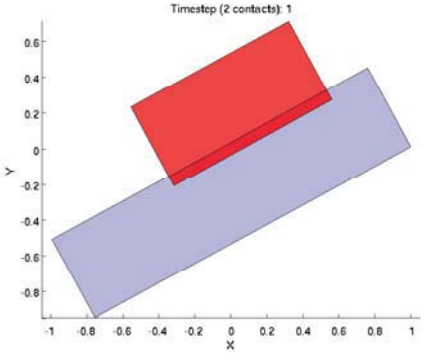
(c) h = 0.2, area error of 0.01



(d) h = 0.3, area error of 0.0223



(e) h = 0.4, area error of 0.0395



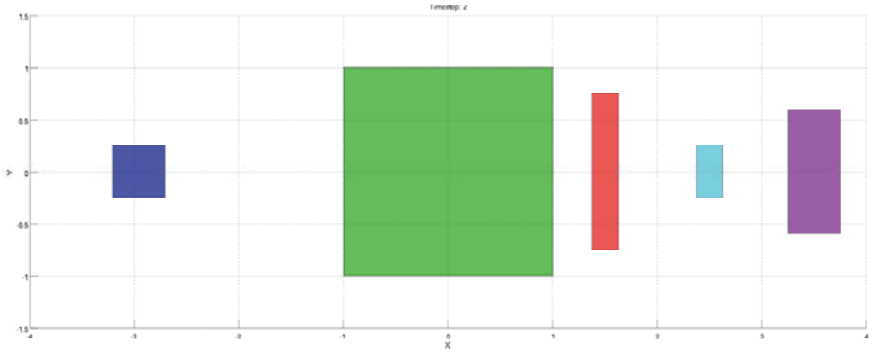
(f) h = 0.5, area error of 0.0612

Figure 5.25: Area of overlap error in semi-implicit time stepping due to rotation and sliding. Clearly, the area of overlap error is proportional to the rotation, which is dependent on the body velocities and time step size.

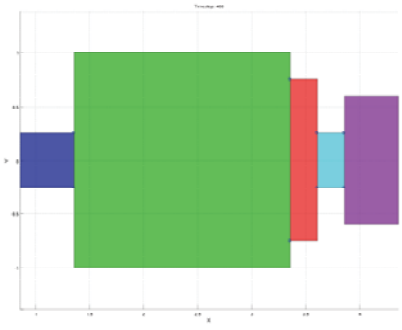
error but off of the bottom body.

5.4 Conservation of momentum

Whether or not momentum is conserved in multibody simulation is not directly dependent on the contact model, but on the dynamic model used. In all cases thus far, PEG has been integrated into the Stewart-Trinkle time-stepping method with inelastic contact. To show how momentum can be conserved using this model, consider the experimental setup depicted in Figure 5.26. The left most body has an initial velocity of 2 m/s to the right, while all other bodies begin at rest. There is no gravity, nor friction. The masses of the bodies, from left to right, are 1.5, 1, 0.25, 0.12, and 0.8 kg.



(a) Setup of example experiment to test conservation of momentum. The left most body has an initial velocity of 2 m/s to the right, while all other bodies are at rest.



(b) Final configuration after all collisions.

Figure 5.26: Simulation experiment to test conservation of momentum with the Stewart-Trinkle time-stepping method.

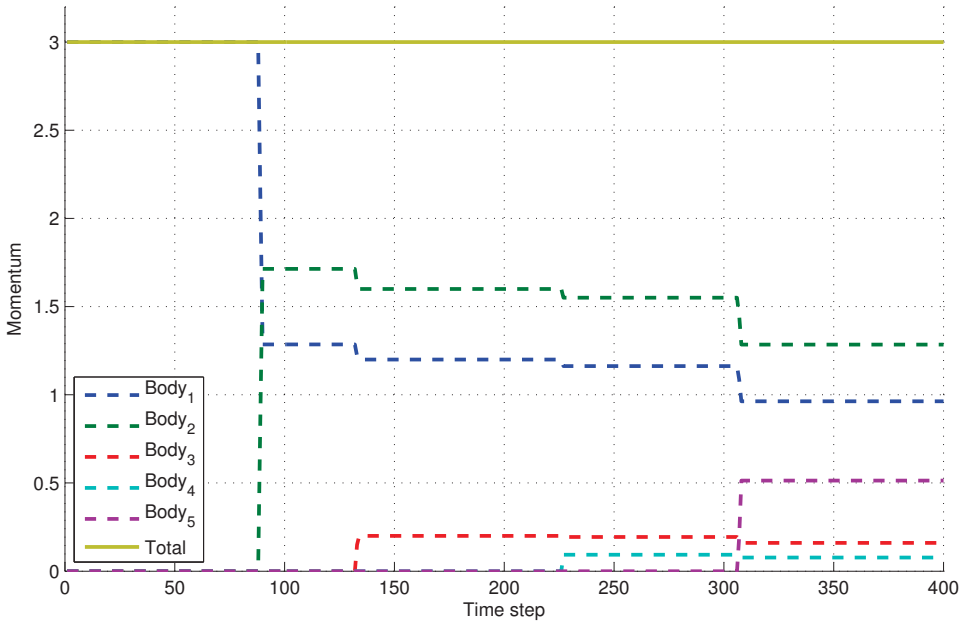


Figure 5.27: Momentum experiment results. Each step in the curves above corresponds to an inelastic impact. Note that the total momentum is conserved over all collisions.

Figure 5.27 depicts the results of the momentum simulation experiment. Each of the discernible steps in the colored curves corresponds to impact with a new body. At each inelastic impact, a body with zero momentum at the beginning of the time step has a non-zero momenum at the end of the time step. Also, each body that had non-zero momentum at the beginning of the time step loses some momentum as its velocity is lowered due to the added mass of the new body. We see that the total momentum of all bodies is constant throughout the simulation, demonstrating conservation of momentum.

5.5 Numerical tolerances

The number of contacts identified in either 2D or 3D is dependent on the body geometries, the body configurations, the time step size, and the values used for the various types of ϵ. Because contact for polygons and polyhedra are determined for specific feature combinations, the more of these features there are within some ϵ of another bodies features, the larger the time-stepping problem will be. In other words, the more complex the body geometries, the more complex the dynamics problem is that needs to be solved to determine accurate interactions.

There are some simulation tolerances that may be and frequently are hard coded. These include such values as ϵ, the distance outside of which we do not consider potential contacts. This value naturally requires tuning, at least to some degree, and is particularly sensitive to time step size as well as body velocities. Consider that the larger the time step or body velocity, the larger the distance a body will travel per time step, requiring an ϵ value large enough to prevent interpenetration over the time step. The purpose of using ϵ is primarily to reduce computational time by spatially limiting the contacts that are considered as well as limiting the complexity of the dynamics step. However, the trade off is clear that while reducing the size of ϵ can help improve performance, it can also break if it is made too small and subsequent interpenetrations occur.

We can dynamically set ϵ values using the time step size h and body properties. A conservative estimate is given by

$$\epsilon = \frac{1}{2}\frac{||\mathbf{f}^\ell||}{m}h + ||\boldsymbol{\nu}^\ell||h + \frac{1}{2}\left(\frac{||\boldsymbol{\tau}^\ell||}{\mathbf{J}} - ||\boldsymbol{\omega}^\ell \times \mathbf{J}\boldsymbol{\omega}^\ell||\right)\mathbf{r}h \tag{5.1}$$

which is the estimate of the maximum distance a point on a body can move in a single time step.

PEG introduces other tolerances, such as ϵ_θ for the test of applicability in section 3.2. The tuning of this parameter is much less critical however, since ϵ_θ is dimensionless and therefore scales nicely. That is, ϵ_θ is practically independent of time step size since it is a measure of radians. The exception to this practical independence would be in the case of very high rotational velocities. Such cases

simply would require smaller time steps.

5.6 Floating point error and contact degeneracies

All computer simulations that intend to represent one or more dimensions of space suffer from floating point errors, and must address these issues to some extent. Additionally, some mathematical models degenerate when contacts have gap distances of less than or equal to zero, *e.g.,* classical applicability. Many degeneracies are due to such weaknesses as sensitivity to dot products or cross products which evaluate to zero. Unfortunately, it is when such values approach their regions of degeneracy that the result is most important!

Particularly prevalent is the case when two bodies are in contact yet the distance calculated is within machine precision, that is, $0 \pm \epsilon_{machine}$. It is then important for any geometric tests that involve the gap distance to gracefully allow for such errors to occur in order to maintain stability. Further, it demonstrates that determining contact normal direction, for example by taking the vector $\mathbf{b} - \mathbf{a}$ for points $\mathbf{a}$ on body $\mathcal{A}$ and $\mathbf{b}$ on body $\mathcal{B}$, is not robust and can be unstable due to the "flipping" of the normal direction or having a zero normal when $\mathbf{a} = \mathbf{b}$.

Chapter summary

In this chapter, we analyzed the approach and behavior of PEG as compared with the standard model in Stewart-Trinkle and a penalty based corrective method. We were presented with precise arguments regarding the inability of heuristics to accurately determine a contact subset for accurate interactions of bodies, underlining the necessity of PEG. We observed instantaneous cases where traps may occur, as well as behavior over entire simulations in both 2D and 3D. PEG consistently outperformed other methods in terms of stability and multiple error metrics. Additionally, we discussed the source of instability in some mathematical models of contact.

CHAPTER 6
RPI-MATLAB-SIMULATOR
A tool for efficient research and practical teaching in multibody dynamics

RPI-MATLAB-Simulator (RPIsim) is an open source simulation framework for research and education in multibody dynamics available from rpi-matlab-simulator(Date Last Accessed, June, 30, 2014) [84]. RPIsim is designed and organized to be extended. Its modular design allows users to edit or add new components without worrying about extra implementation details. RPIsim has two main goals:

1. Provide an intuitive and easily extendable platform for research and education in multibody dynamics

2. Maintain an evolving code base of useful algorithms and analysis tools for multibody dynamics problems.

In this chapter, we will take a look at many of the features of RPIsim and also see some examples.

MATLAB is a popular numerical computing environment developed by MathWorks and available for purchase from `www.mathworks.com/products/matlab/`(Date Last Accessed, June, 30, 2014). Follow MATLAB's installation instructions for your environment.

GNU Octave is a free MATLAB alternative that is available for Linux and Windows systems [85]. Windows users must use Cygwin to run Octave, which is available from `www.cygwin.com/`(Date Last Accessed, June, 30, 2014). Octave is available from `www.gnu.org/software/octave/download.html`(Date Last Accessed, June, 30, 2014), but on many popular Linux systems, Octave can be quickly installed by running

---

Portions of this chapter previously appeared as: J. Williams, Y. Lu, S. Niebe, M. Andersen, K. Erleben, J.C. Trinkle. *RPI-MATLAB-Simulator: A tool for efficient research and practical teaching in multibody dynamics*. Workshop on Virtual Reality Interaction and Physical Simulation (VRIPHYS).

RPI-MATLAB-Simulator is available online at `code.google.com/p/rpi-matlab-simulator/`. In a Linux environment, the RPIsim source code can be downloaded using subversion by running

```
 $ svn checkout
http://rpi-matlab-simulator.googlecode.com/svn/simulator
     rpi-matlab-simulator
```

If you do not have subversion, you will have to install it. For Windows, we suggest TortoiseSVN, available from `tortoisesvn.net/`. For Linux systems, simply run

```
 $ sudo apt-get install subversion
```

Getting started

For the remainder of the chapter, we will discuss using RPIsim in the context of MATLAB, however everything should be similarly executable in Octave.

Once you have a system setup and have downloaded RPIsim, open MATLAB and change directories to RPIsim and run *setupRPIsim.m*:

```
 >> cd ~/rpi-matlab-simulator/
 >> setupRPIsim
```

setupRPIsim.m simply adds all of the necessary directories to the path. Now run a test script such as

```
 >> simtest
```

You should be presented with the graphical representation of a small sample scene that will run automatically and look similar to the following image. The blue wire-frame cubes are axis-aligned bounding boxes and will turn red when the corresponding body has active contacts with another body (visualization of bounding boxes is not on by default). The largest cube is static, and will not move. The

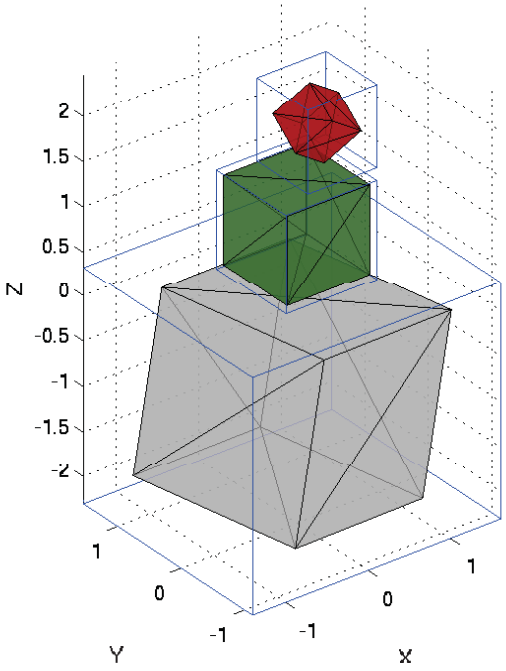


Figure 6.1: Initial configuration of three bodies in an example simulation with RPIsim. The blue cubes represent axis-aligned bounding boxes.

other two bodies will fall with gravity and slide off of the static cube. You should see contacts plotted in green as they are identified between bodies.

6.1 Creating and running scenes

In this section we introduce the basics of writing a simulation script and running it. We cover several of the common options in 6.1.1, then we will learn by example in 6.1.2.

6.1.1 Simulator options

There are several options when running a simulation including whether to use the GUI, whether to show contacts or bounding boxes, what step size to use, whether to record the simulation, etc. Many of these options are exemplified in section 6.1.2, but we will describe several of the options in Table 6.1 for reference. You can find all of these fields in *Simulation.m*. When used below, *sim* represents a *Simulator*

struct.

Table 6.1: Simulator options (See *Simulator.m* for the complete list).

Option syntax	Default value	Description
$sim.draw = [BOOL]$	*true*	Sets whether to graphically display the simulation as it runs.
$sim.drawBoundingBoxes = [BOOL]$	*false*	Sets whether to graphically display bounding boxes of bodies.
$sim.drawContacts = [BOOL]$	*false*	Sets whether to graphically display contacts as they occur during simulation.
$sim.drawJoints = [BOOL]$	*false*	Sets whether to graphically display joints.
$sim.gravity = [BOOL]$	*true*	Turns gravity on (true) and off (false). This must be done before calling *sim_run()*.
$sim.gravityVector = [DOUBLE_{3\times1}]$	$\begin{bmatrix} 0 \\ 0 \\ -9.81 \end{bmatrix}$	A 3×1 vector that defines the magnitude and direction of the gravitational force.
$sim.H\_collision\_detection = @[function]$	*collision_detection*	The function handle that points to the function which performs collision detection.
$sim.H\_dynamics = @[function]$	*mLCPdynamics*	The function handle that points to the function that will formulate the time-stepping subproblem at each simulation time step.
$sim.num\_fricdirs = [INT]$	7	The number of friction directions to use when linearizing the friction cone during the dynamics formulation stage.

6.1.2 Example scripts

Hello world

```
1    sim = Simulator();                      % Initialize the simulator
2    sim = sim_addBody( sim, mesh_cube() );  % Add a cube to the scene
3    sim = sim_run( sim );                   % Run the simulator
```

Script 1. Hello world.

Script 1 depicts a minimal simulation with only a single cube at the origin. The first line initializes a simulator structure, the second line adds a default cube to the simulator, and the third line runs the simulator.

A rolling cart

Let's create a rectangular chassis and put four wheels on it!

```
1    sim = Simulator(.01);
2    sim.H_dynamics = @mLCPdynamics;
3    sim.drawContacts = true;
4    sim.drawJoints = true;
5
6    % Ground
7    ground = Body_plane([0; 0; 0],[0; .1; 1]);
8        ground.color = [.7 .5 .5];
9        ground.mu = 1;
10
11   % Chassis
12   chassis = mesh_rectangularBlock(1,3,0.25);
13       chassis.u = [0; 0; 1];
14       chassis.color = [.3 .6 .5];
15
16   % Wheels
17   wheel = mesh_cylinder(20,1,0.25,0.2);
18   wheel.quat = qt([0 1 0],pi/2);
19   w1 = wheel;
20   w2 = wheel;                 % Copy wheel struct to four wheels
```

```
21    w3 = wheel;
22    w4 = wheel;
23    w1.u = [ .5;  1.3; .75];    % Position wheels around chassis
24    w2.u = [-.5;  1.3; .75];
25    w3.u = [ .5; -1.3; .75];
26    w4.u = [-.5; -1.3; .75];
27
28    % Add bodies to simulator
29    sim = sim_addBody(sim, [ground chassis w1 w2 w3 w4]);
30
31    % Create joints
32    sim = sim_addJoint( sim, 2, 3, w1.u, [1;0;0], 'revolute');
33    sim = sim_addJoint( sim, 2, 4, w2.u, [1;0;0], 'revolute');
34    sim = sim_addJoint( sim, 2, 5, w3.u, [1;0;0], 'revolute');
35    sim = sim_addJoint( sim, 2, 6, w4.u, [1;0;0], 'revolute');
36
37    % Run the simulator
38    sim = sim_run( sim );
```

Script 2. A rolling cart.

In lines 1-4, we initialize the simulator and set some properties for run-time. Then we create a ground plane (7-10), chassis (12-14), and four wheels (17-26). Notice that when we created the four wheels, we re-used a generic "wheel" struct that had the rotation and body properties we wanted to share between all four wheels. The one attribute we needed to change for each wheel was its starting position u (23-26). Line 29 adds the bodies to the simulator. Lines 32-35 create the four revolute joints between the wheels and chassis. Unfortunately at the time of this writing, there is no "nice" way to reference the bodies when creating a joint, so we must use the bodyID attribute. For example, line 32 specifies a joint between body 2 (the chassis) and body 3 (w1), where body 1 was the ground plane. This requires the user to be able to keep track of what order bodies were added.

6.2 Recording and replaying simulations

Basic record and playback

Recording a simulation is done by setting the *Simulation* attribute *record* to true as on line 2 of the script below. To improve performance, you may turn off the GUI during simulation. MATLAB graphics tend to be quite slow, often taking up about half the time of simulation!

```
1    sim = Simulator();       % Create a simulator
2    sim.record = true;       % Turn on recording
3    sim.MAX_STEP = 500;
4    sim.draw = false;
5    ...
6    sim = sim_run( sim );    % Run the simulator
```

Script 3. Turning on recording

Each time a simulator is run with record enabled, a directory is created with a name formatted as "*sim_data_*"[*year*]_[*month*]_[*day*]_[*hour*]_[*minute*]_[*second*]. This looks like a long name, but it will help identify simulation data. For example, *sim_data*_13_10_14_18_22_59 was created at 6:22:59 pm on October 10, 2013.

Playing a recorded simulation is done by running *sim_replay*([directory]), for example

```
>> sim_replay( 'sim_data_13_10_14_18_22_59' )
```

Figure 6.2 depicts the GUI with which the user is presented when re-animating a simulation using *sim_replay()*. The slider allows the user to browse frames of the simulation, and the "Play" button animates over all frames.

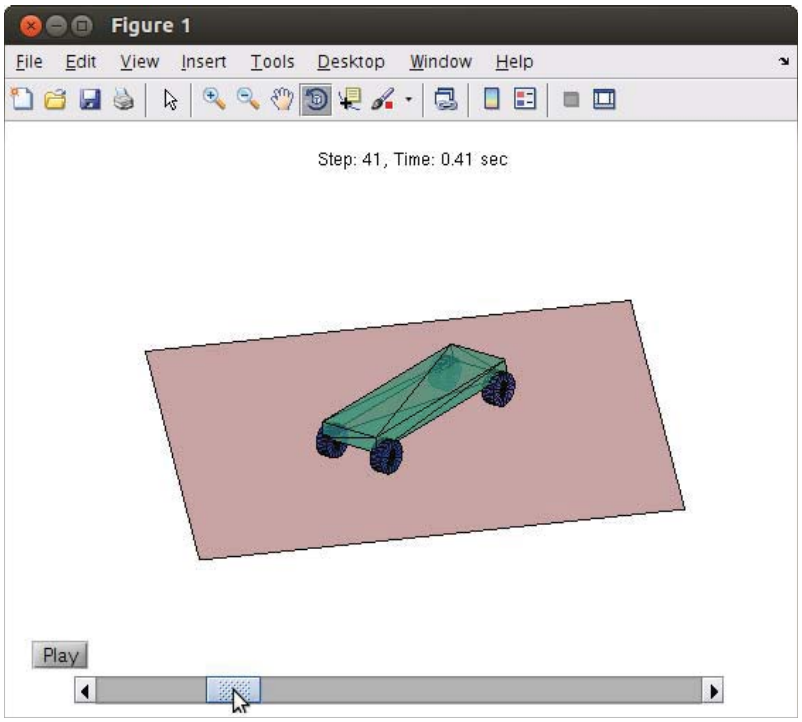


Figure 6.2: The *sim_replay()* GUI. The "Play" button will animate the simulation, or dragging the slider will set the simulation time manually.

Incorporating user functions

There are two fields in the Simulator struct that make interacting with data during simulation straightforward: $userFunction$ and $userData$. $userFunction$ allows the user to to specify a function to be evaluated at the end of every time step. $userData$ is a struct where values generated in the user function can be stored. The $userFunction$ allows the user to put in place a custom function such as a controller or functionality for plotting. A controller could be an explicit time-based position or velocity controller, or a proportional-integral-derivative (PID) controller for joint control of a robotic arm. Although all simulation variables are available and editable at this stage, it is recommended that bodies be controlled only by setting external forces and allowing the dynamics to solve for the next step. This is similar to the idea behind energy functions in simulation [86].

The following sections exemplify how to use the $userFunction$ and $userData$.

Position, velocity, acceleration

Script 4 demonstrates a user function for plotting the vertical position, velocity, and acceleration of a body. Script 5 is an example simulation that utilizes Script 4. Notice that Script 4 takes *sim* as an argument and also returns *sim*. Important things to remember when writing your userFunction:

- userFunction should take *sim* as an argument and return *sim*.
- All variables that you wish to store go in $sim.userData$.
- When plotting, you may have to check if it is the simulation's first step in order to initialize things.
- You don't just have to use custom functions for plotting!

```
function sim = plotPVA( sim )
 body = sim.bodies(2);   % The body we'll plot P, V, and A for.

 P = body.u(3);              % Verticle position
 V = body.nu(3);             % Verticle velocity
 A = body.Fext(3)/body.mass; % Verticle acceleration

 if sim.step == 1
   % On the first time step, the plot doesn't exist yet,
   % so we'll create it.
   figure();
   sim.userData.P = plot(sim.time, P, 'r');   hold on;
   sim.userData.V = plot(sim.time, V, 'g');
   sim.userData.A = plot(sim.time, A, 'b');
   xlabel('Time (s)');
   legend('Position','Velocity', 'Acceleration',2);
 else
   % After the first time step the plots exist,
   % so we'll just update their X and Y data.
   set(sim.userData.P,'xdata',[get(sim.userData.P,'xdata') sim.time]);
   set(sim.userData.P,'ydata',[get(sim.userData.P,'ydata') P]);

   set(sim.userData.V,'xdata',[get(sim.userData.V,'xdata') sim.time]);
   set(sim.userData.V,'ydata',[get(sim.userData.V,'ydata') V]);

   set(sim.userData.A,'xdata',[get(sim.userData.A,'xdata') sim.time]);
   set(sim.userData.A,'ydata',[get(sim.userData.A,'ydata') A]);
 end
end
```

Script 4. A user function for plotting position, velocity, and acceleration.

```
function sim = hangingRod()
  % Initialize simulator
  sim = Simulator();
  sim.H_dynamics = @mLCPdynamics;
  sim.userFunction = @plotPVA;

  % Create an invisible static body
  staticBody = mesh_cylinder(7,1,0.2,1);
```

```
9         staticBody.dynamic = false;
10        staticBody.visible = false;
11
12    % Create a hanging rod
13    angle = pi/5;
14    hangingBody = mesh_cylinder(7,1,0.1,0.5);
15        hangingBody.u = [-0.25*sin(angle); 0; 0.25-0.25*cos(angle)];
16        hangingBody.quat = qt([0;1;0], angle);
17
18    % Gather simulation bodies and add to simulator
19    bodies = [staticBody hangingBody];
20    sim = sim_addBody( sim, bodies );
21
22    % Create joint
23    sim = sim_addJoint(sim, 1, 2, [0;0;0.25], [0;1;0], 'spherical');
24
25    % Run simulation
26    sim = sim_run( sim );
27  end
```

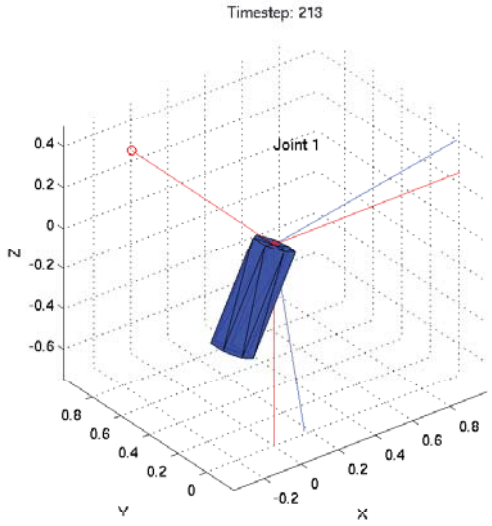
Script 5. A simulation that uses plotPVA().

Running hangingRod will generate two plots: the simulation (Figure 6.3a) and the plot generated by the user function (Figure 6.3b).

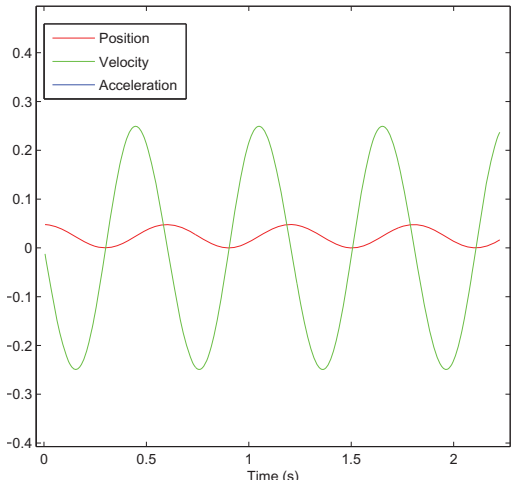
Joint Error

Let's take a look at joint error for a ranging rod pendulum. The simulation is depicted in Figure 6.3a. We can plot the joint error during simulation with the following function:

```
1    function sim = plotJointError( sim )
2        [C,~] = joint_constraintError(sim,1);  % Error in FIRST joint
3        if sim.step == 1
4           figure(2);
5           sim.data.error = plot(sim.time,norm(C));
```



(a) **hangingRod(), a simulation that calls the userFunction plotPVA.**



(b) **Position and velocity (acceleration is constant -9.81) in vertical direction.**

Figure 6.3: A hanging rod simulation with userFunction plotting. While the simulation on the left runs, plots are simultaneously being generated during simulation with the userFunction.

```
6           xlabel('Time (s)');
7           ylabel('norm(Joint Error)');
8       else
9           set(sim.data.error,'xdata', ...
10                [get(sim.data.error,'xdata') sim.time]);
11          set(sim.data.error,'ydata', ...
12                [get(sim.data.error,'ydata') norm(C)]);
13      end
14  end
```

Script 6. Function for plotting joint error.

To tell the simulator to call this function during simulation, we add the single line

```
    sim.userFunction = @plotJointError;
```

to our simulation script. Figure 6.4 depicts the magnitude of the joint error for hangingRod.m over five seconds of simulation.

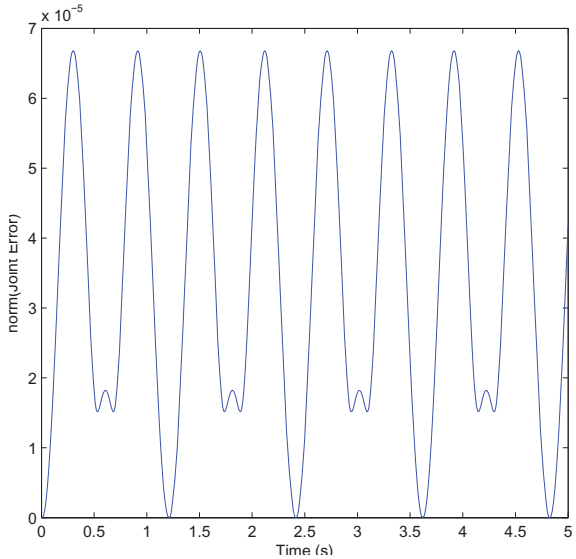


Figure 6.4: Error in joint position for the hanging rod pendulum. Due to the stabilization term in Stewart-Trinkle, the joint error is bound, in this case below 7×10^{-5}.

6.3 The default simulator

This section discusses the workings of RPIsim "out of the box" and goes into greater detail about what functionality is already implemented.

The file structure of the simulator is depicted in Figure 6.5. This structure is meant to be intuitive and help guide the user when editing or adding new components. The "examples" directory contains several examples of scenes which demonstrate how to specify options, and run a simulation. The simulator itself is entirely contained in the "engine" directory. The directories found there are fairly self-explanatory. The "dynamics" directory contains the functions defining each available time-stepping formulation, all of which construct a time-stepping subproblem formulated as a complementarity problem (CP). The "solvers" directory contains various functions for solving the complementarity problem: linear complementarity problem (LCP), mixed linear complementarity problem (mLCP), and nonlinear

complementarity problem (NCP). See [39] for a comprehensive review of these topics.

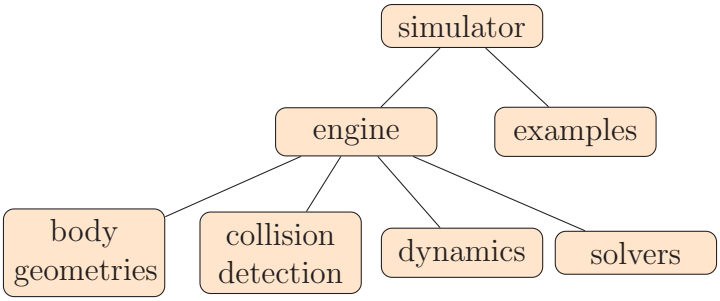


Figure 6.5: RPIsim file structure. Simulator code is organized in order to reflect the stages of simulation and to give the user intuition about the connectedness of these stages. The existing code serves as templates for extending the simulator with custom modules.

The flow of simulation is depicted in Figure 6.6. As soon as a simulation script is executed, the scene is rendered so that it may be inspected. When run, the simulator proceeds through the various stages of the simulation loop.

Body representations

In previous versions of RPIsim, each body type, *i.e.,* cubes, spheres, etc., was an instance of a corresponding MATLAB object. This has changed in the current version for multiple reasons (in part because Octave does not use objects in the MATLAB sense), and now all simulation bodies are represented by the *Body* struct. All common body attributes such as position, rotation, mass, inertia, etc. are contained in Body, but because all bodies are represented by this one struct, it also contains type-specific attributes such as radius for spheres, point and normal for planes, etc. See *Body.m* for more details.

Triangle meshes

All mesh bodies are represented as triangle meshes. There are several meshes already available including mesh_cube, mesh_cylinder, mesh_rectangular_block, and the five Platonic solids.

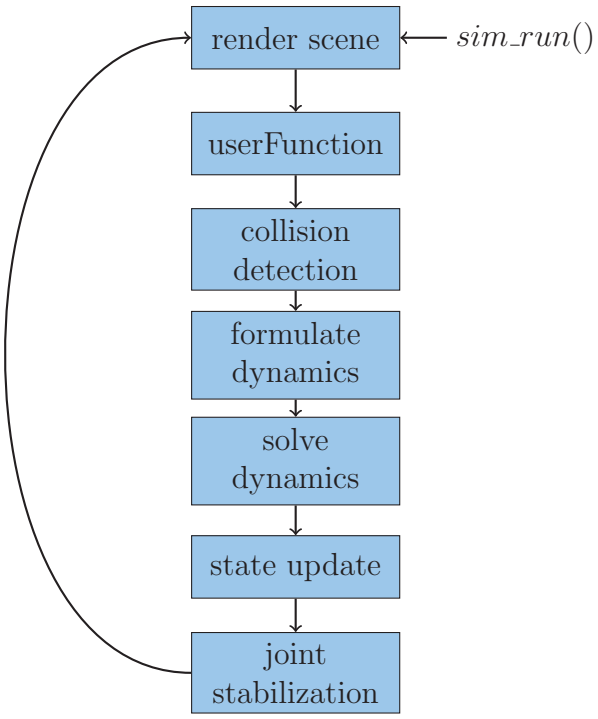


Figure 6.6: Simulation loop in the RPI-MATLAB-Simulator. At each stage, a user can replace or plug in custom modules.

New meshes can be created by either generating the vertex and face data, or using *mesh_read_poly_file.m* to read in a text file like *cube.poly*. The format of this file must first list the vertices in order, then list faces as triples of vertex indices. The function *test_mesh_object.m* is useful when creating your own meshes as it will plot your mesh along with the face normals. It's important for collision detection to make sure that the normals are all pointing "out" of the body.

6.4 Dynamics formulations

There are several ways available for modelling the dynamic behaviour of the simulation. The default formulation is in *LCPdynamics.m* and constructs the matrices in vectors needed to solve the linear complementarity problem [18]

$$\mathbf{0} \leq \begin{vmatrix} \mathbf{G}_n^T\mathbf{M}^{-1}\mathbf{G}_n & \mathbf{G}_n^T\mathbf{M}^{-1}\mathbf{G}_f & \mathbf{0} \\ \mathbf{G}_f^T\mathbf{M}^{-1}\mathbf{G}_n & \mathbf{G}_f^T\mathbf{M}^{-1}\mathbf{G}_f & \mathbf{E} \\ \mathbf{U} & -\mathbf{E}^T & \mathbf{0} \end{vmatrix} \begin{vmatrix} \mathbf{p}_n^{\ell+1} \\ \mathbf{p}_f^{\ell+1} \\ \mathbf{s}^{\ell+1} \end{vmatrix} + \begin{vmatrix} \mathbf{G}_n^T(\nu^\ell + \mathbf{M}^{-1}p_{ext}^\ell) + \frac{\mathbf{\Psi}_n^\ell}{h} + \frac{\partial \mathbf{\Psi}_n^\ell}{\partial t} \\ \mathbf{G}_f^T(\nu^\ell + \mathbf{M}^{-1}p_{ext}^\ell) + \frac{\partial \mathbf{\Psi}_f}{\partial t} \\ \mathbf{0} \end{vmatrix} \perp \begin{vmatrix} \mathbf{p}_n^{\ell+1} \\ \mathbf{p}_f^{\ell+1} \\ \mathbf{s}^{\ell+1} \end{vmatrix} \geq 0 \tag{6.1}$$

where after a solution is found for $\mathbf{p}_n^{\ell+1}$ and $\mathbf{p}_f^{\ell+1}$, new velocities are calculated for all bodies with

$$\boldsymbol{\nu}^{\ell+1} = \boldsymbol{\nu}^\ell + \mathbf{M}^{-1}\mathbf{G}_n\mathbf{p}_n^{\ell+1} + \mathbf{M}^{-1}\mathbf{G}_f\mathbf{p}_f^{\ell+1} + \mathbf{M}^{-1}\mathbf{p}_{ext} \tag{6.2}$$

Note that (6.1) does not include bilateral constraints. Currently when simulating joints, you must use *mLCPdynamics.m* which constructs the mixed linear complementarity problem as

$$\begin{vmatrix} \mathbf{0} \\ \boldsymbol{\rho}_b^{\ell+1} \\ \boldsymbol{\rho}_n^{\ell+1} \\ \boldsymbol{\rho}_f^{\ell+1} \\ \boldsymbol{\sigma}^{\ell+1} \end{vmatrix} = \begin{vmatrix} -\mathbf{M} & \mathbf{G}_b & \mathbf{G}_n & \mathbf{G}_f & \mathbf{0} \\ \mathbf{G}_b^T & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{G}_n^T & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{G}_f^T & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{E} \\ \mathbf{0} & \mathbf{0} & \mathbf{U} & -\mathbf{E}^T & \mathbf{0} \end{vmatrix} \begin{vmatrix} \boldsymbol{\nu}^{\ell+1} \\ \mathbf{p}_b^{\ell+1} \\ \mathbf{p}_n^{\ell+1} \\ \mathbf{p}_f^{\ell+1} \\ \mathbf{s}^{\ell+1} \end{vmatrix} + \begin{vmatrix} \mathbf{M}\boldsymbol{\nu}^\ell + \mathbf{p}_{ext} \\ \frac{\mathbf{\Psi}_b^\ell}{h} + \frac{\partial \mathbf{\Psi}_b^\ell}{\partial t} \\ \frac{\mathbf{\Psi}_n^\ell}{h} + \frac{\partial \mathbf{\Psi}_n^\ell}{\partial t} \\ \frac{\partial \mathbf{\Psi}_f^\ell}{\partial t} \\ \mathbf{0} \end{vmatrix} \tag{6.3}$$

$$\mathbf{0} \leq \begin{bmatrix} \boldsymbol{\rho}_b^{\ell+1} \\ \boldsymbol{\rho}_n^{\ell+1} \\ \boldsymbol{\rho}_f^{\ell+1} \\ \boldsymbol{\sigma}^{\ell+1} \end{bmatrix} \perp \begin{bmatrix} \mathbf{p}_b^{\ell+1} \\ \mathbf{p}_n^{\ell+1} \\ \mathbf{p}_f^{\ell+1} \\ \mathbf{s}^{\ell+1} \end{bmatrix} \geq \mathbf{0} \tag{6.4}$$

where $\boldsymbol{\rho}_b$, $\boldsymbol{\rho}_n$, $\boldsymbol{\rho}_f$, and $\boldsymbol{\sigma}$ are slack variables.

Solving dynamics

Several solvers are available including an implementation of Lemke's method [87], the PATH solver [40], and a suite of solvers num4lcp [58]. Having numerous solvers available offers choices and also the opportunity to compare their performance [88, 89]. These solvers are classified into pivoting methods and iterative

methods.

Pivoting method

Lemke's algorithm is one robust pivoting method, although there is no general solution method which guarantees the solution of any given LCP except for a total enumeration of 2^n for a problem of size n. Lemke's algorithm works well for a small problem size while for larger-size problem, it tends to be slow.

Iterative method

The iterative method, such as Projected Gauss Seidel (PGS), generalized Newton's method and fixed-point iteration methods are included in the RPIsim. The following table shows briefly about the available solvers inside the RPIsim:

Table 6.2: Solvers available with RPIsim.

Name	Applicable Model	Category
Lemke	LCP	pivotal method
PATH	mLCP	mixed pivotal and iterative method
PGS	LCP, mLCP	iterative method with projection
Projected Jacobi	LCP, mLCP	iterative method with projection
Fixed-point	LCP, NCP	iterative method
Interior-point	LCP	iterative method
Fischer-Newton	LCP, NCP	non-smooth Newton with line search
Minmap-Newton	LCP, NCP	non-smooth Newton with line search

To call a specific solver, say, to use PATH solver, you need to call it by setting:

```
sim.H_solver = @pathsolver;
```

Customizing the simulator

One of the main goals of RPIsim is to enable easy customizations and extensions through its modular structure. This section describes some of the default

modules and gives examples on how to replace them with custom modules. Essentially this comes down to understanding the interface for a given component in the simulation loop (Figure 6.6).

Collision detection

The default collision detection function in RPIsim is *collision_detection.m*, which can find contacts between combination pairs of planes, spheres, and triangle meshes (of course we never test plane against plane). Unfortunately, the algorithms implemented in the default function are slow, so it is up to you to write optimized routines (and commit them to the repository!).

A collision detection routine in RPIsim will iterate over all bodies contained in *sim.bodies* and determine and store a set of contacts in *sim.contacts*.

Important things to remember when writing a collision detection routine in RPIsim:

- When adding a contact between bodies A and B, make sure to "activate" those bodies with

  sim = sim_activateBodies(sim, Aid, Bid);

  This marks each body as dynamically "active" for the current time step, and is important for all of the indexing that will happen later in the simulation loop (when building matrices for the dynamics).

- Some bodies may be static, so improve efficiency by checking:

  if ~A.dynamic && ~B.dynamic, continue; end

- Some bodies are set to not collide (e.g. two joined bodies), so avoid errors by checking:

  if any(A.doesNotCollideWith == B.bodyID), continue; end

There are several functions included in *engine/collision_detection/body_tests* that may be helpful in writing a collision detection routine.

Formulating dynamics and solving

In order to incorporate a custom dynamics formulation or solver, the user should be familiar with how dynamics components are passed from one block of the simulation loop to the next. By default, the function *preDynamics.m* puts together the necessary matrices and vectors including the mass-inertia matrix and the Jacobians for unilateral contact, bilateral joints, and friction constraints. These matrices are stored in the struct *sim.dynamics* in order to pass them along.

A custom dynamics formulation may choose to use the value in *sim.dynamics* or not. If not, then it is of course most efficient to remove the call to *preDynamics*() inside of *sim_run.m*.

Using Bullet collision detection

Unfortunately, the Bullet interface will only work with MATLAB 32-bit versions, and will not work with Octave nor MATLAB 64-bit versions. This is because 1. it requires use of MEX files, which Octave does not support and 2. it requires use of 32-bit MEX files and 64-bit MATLAB cannot compile 32-bit MEX files.

Installing Bullet

In order to use the Bullet collision detection library, you first need Bullet installed. This can be done by following the instructions on `bulletphysics.org/mediawiki-1.5.8/index.php/Installation`.

Under Linux, I was able to do this with revision 2672 of Bullet:

```
$ svn checkout bullet.googlecode.com/svn/trunk/ bullet-read-only
$ cd bullet-read-only
$ mkdir bullet-build
$ cd bullet-build
$ cmake .. -G "Unix Makefiles" -DINSTALL_LIBS=ON
                          -DBUILD_SHARED_LIBS=ON
$ make -j4
$ sudo make install
```

Using Bullet with RPIsim

Once Bullet is installed, we need to build the interface that will pass the RPIsim body information to Bullet. This is done in RPIsim with

```
>> Compile_BULLET
```

One possible error may occur if you do not have a compiler set in MATLAB for compiling MEX functions.

6.5 Sample application: PD controlled robotic arm

In this section, we will demonstrate how to use RPIsim to simulate a robotic arm and a proportional-derivative controller in the joint space for the arm.

Modeling the arm

Many thanks go to Rohinish Gupta who did the work of importing the mesh of the robotic arm into MATLAB. The arm we model is the Schunk Powerball arm, and the files are contained in *examples/powerball*. Specifically the script powerball.m loads the mesh bodies, defines the joints between all bodies, and initializes the simulator. Following the convention of RPIsim, the geometric model is a triangle mesh.

Kinematics

Directly controlling the arm by setting the joints to specific angles is straight forward as it is handled by the forward kinematics in the model. The variable *jointAngle* in powerball.m is initially set to all zeros, but may be altered to any set of joint angles.

Dynamic simulation

Dynamic control of the arm is achieved through a proportional-derivative controller utilizing the userFunction functionality as described in section 6.2. Because the userFunction is called at the beginning of each time step, we can observe the

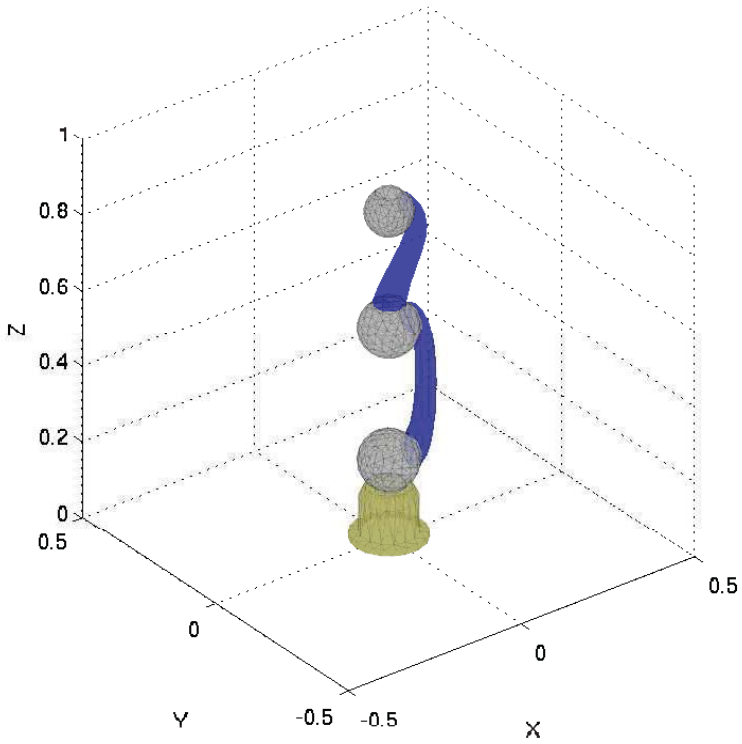


Figure 6.7: Simulation of a robotic arm in RPIsim. A PD-controller is used to control the arm in joint-space. Here, the arm is depicted in its zero-configuration, *i.e.*, all joint angle values are zero.

current state of each joint and apply external torques to any dynamic bodies. Our PD-controller *userFunction* will look like

```
1  function sim = powerball_controller( sim )
2   theta_desired = ones(length(sim.joints),1) * sin(sim.step*.01);
3   % Clear body torques
4   for j=1:length(sim.joints)
5    sim.bodies(sim.joints(j).body1id).Fext(4:6) = 0;
6    sim.bodies(sim.joints(j).body2id).Fext(4:6) = 0;
7   end
8
9   % Calculate new torques
10  for j=1:length(sim.joints)
11   J = sim.joints(j);
```

```
12
13    Perror = theta_desired(j) - J.theta;         % Proportional error
14    Deriv  = (J.theta - J.theta_prev) / sim.h;   % Derivative
15    joint_frame_torque = 50*Perror - 5*Deriv;    % PD controller
16
17    t1 = J.T1world(1:3,1:3) * [0;0; -joint_frame_torque];
18    sim.bodies(J.body1id).Fext(4:6)=sim.bodies(J.body1id).Fext(4:6)+t1;
19
20    t2 = J.T2world(1:3,1:3) * [0;0; joint_frame_torque];
21    sim.bodies(J.body2id).Fext(4:6)=sim.bodies(J.body2id).Fext(4:6)+t2;
22
23    % For now, we need to keep track of this in the controller.
24    sim.joints(j).theta_prev = J.theta;
25   end
26  end
```

Script 7. PD-control userFunction.

Figure 6.8 depicts several stages of dynamic simulation of the Powerball arm executing a joint space trajectory from a start position (zero configuration) to a goal position.

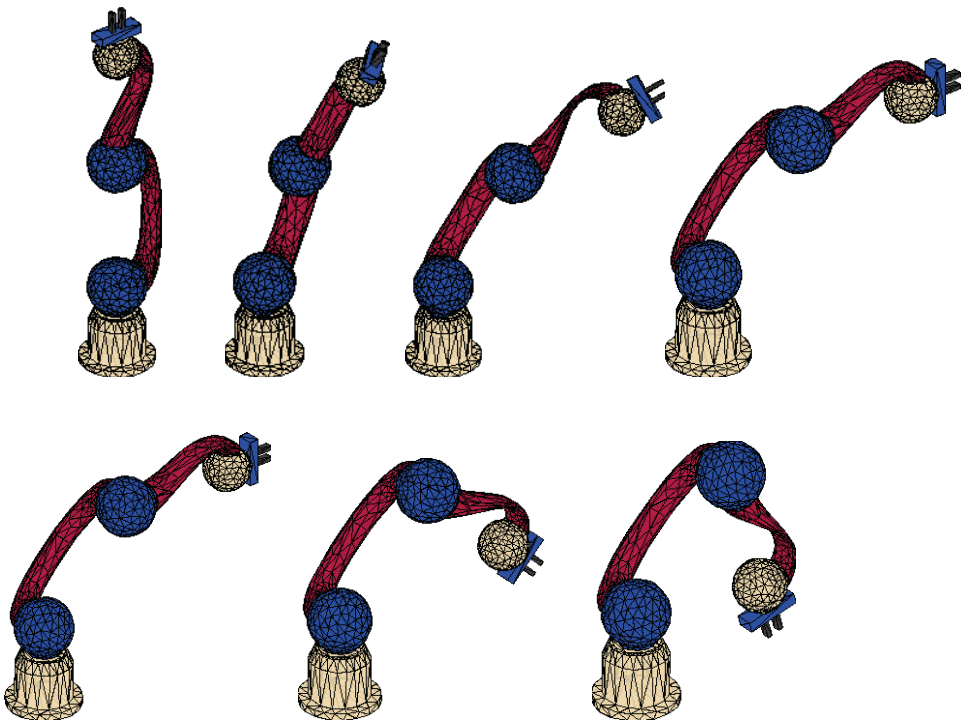


Figure 6.8: Various stages of simulation of Schunk Powerball arm executing a grasp trajectory. The trajectory is a set of joint angles interpolated between start and goal configurations over a given time.

Chapter summary

In this chapter, we were introduced to the RPI-MATLAB-Simulator (RPIsim), a fully functioning multibody dynamics tool for 2D and 3D simulation. We covered several of its features and saw example scripts, from a "hello world" to a fully dynamically simulated PD controlled Powerball arm, demonstrating its modularity and ease of use. RPIsim's code structure mimics the flow of a time-stepping simulation scheme, making it easy to incorporate new code or replace existing modules within the simulator. User defined functions provide an easy way to do real-time plotting of simulation parameters, and also provide a template for instructors to build custom dynamics or simulation assignments.

CHAPTER 7
CONCLUSIONS AND FUTURE WORK

7.1 Conclusion

The goal of this thesis is to provide mathematical techniques and a set of simulation tools for the advancement of multibody simulation and the benefit of such fields as robotics which require the improved physical accuracy and stability these tools provide. Along the way, I have attempted to give insight into common pitfalls in multibody simulation.

In Chapter 3, I enumerated several classical ideas in computational geometry and showed why geometric relaxations must be applied to these ideas in order to successfully determine sets of potential contacts before they occur. It is important to detect potential contacts because we must prevent (non-trivial) interpenetration in order to avoid the contact degeneracies and dynamic instabilities common to many state-of-the-art simulators. I also presented new geometric tests for further improving identification of these potential contacts, as well as determining accurate contact normals.

Chapter 4 contained the most significant contribution, a contact model which was first derived in 2D and then extended to 3D. The new model, which we refer to as Polytope Exact Geometry (PEG), formulates non-penetration constraints as combinations of all potential contacts in the form of a complementarity problem so that only physically feasible body configurations may result at the end of the time step. In Chapter 5, I analyzed the behavior of PEG and compared its performance in several benchmarks against other well known and used models.

In Chapter 6, I introduced the RPI-MATLAB-Simulator (RPIsim). RPIsim is a tool for research and education in simulation and multibody dynamics. RPIsim has ample documentation with many examples, and can be used with little programming experience. RPIsim is also compatible with the open source math package Octave.

7.2 Future Work

There are many interesting avenues of possible future work extending the work presented here. Below, I describe several of these in some detail.

Proof of solvability

As mentioned in 4.8, I can currently offer no proof as to the existence of solution for the general PEG formulation. Although I have deomonstrated proof of concept via the examples and simulation experiments, and in practice I have never seen a simulation with PEG fail, it would be very interesting (and satisfying) to achieve a formal proof of solution existence or be able to identify any cases which could break PEG or are degenerate. Ideally, we would like to achieve a proof of global convergence similar to what David Stewart achieved for the standard contact model in Stewart-Trinkle [18]. Of course, the main matrix of the time-stepping problem with PEG is different than the standard model. Similar to what was pointed out by Nguyen in [74], the inclusion of sub-matrices similar to $\mathbf{G}_{\mathcal{I}}$ in equation (4.50) break the co-positivity of the main matrix of the LCP.

Reduce constraints

An interesting question that is extremely difficult to prove for non-trivial cases is that of whether PEG constraints might be redundant and could be reduced to achieve a smaller problem size. Such a case, if it were to exist, would be dependent on configurations of likely three or more bodies. One way to detect the existence of such cases is to determine all constraints between all body pairs, and then attempt to reduce the full problem before solving. If linearly dependent constraints are found, it is possible that one could reverse engineer the configuration(s) which lead to the redundancy. This would be useful in generating smaller problem sizes more efficiently, which would be subsequently easier to solve.

Extention to non-polytopes

We briefly discussed in section 4.6 utilizing the underlying mathematical framework of PEG for non-polytope bodies such as those defined by semi-algebraic sets,

or non-uniform rational basis splines (NURBs) which include Bézier curves and B-splines. It would be straight forward to develop PEG alternatives which would be useful for alternative body geometries. As a first step, we could utilize collision detection routines which first determine an initial contact between convex bodies [90], then determine which additional contacts are necessary to prevent interpenetration.

Optimized implementation

Nearly all of the testing done with PEG was completed in MATLAB, and was therefore unable to achieve speeds similar to state-of-the-art simulators written in C/C++. As such, it is difficult to make any significant comparisons regarding the speed performance of PEG. Fortunately, the vast majority of the computation necessary for formulating PEG lies in the collision detection and configuration identification, which is highly conducive to parallelization. An excellent next step with this work would be a more optimized implementation, possibly utilizing newer GPU technologies such as CUDA.

Tuning of geometric parameters

In Chapter 3, I introduced geometric relaxations and other ideas which are dependent on parameters such as ϵ distances or angles of relaxation. It is possible that these parameters to could be more tightly tuned based on simulation attributes such as the sizes of bodies, or body attributes such as velocity. For the sake of computation, we wish to include as few potential contacts as possible while not allowing interpenetrations. There is more that can be done regarding this tricky tuning, and possibly more insight to be gained regarding the geometric tests.

REFERENCES

[1] C. Ericson, *Real-Time Collision Detection*. San Francisco, CA: Morgan Kaufmann Publishers Inc., 2004.

[2] D. H. Eberly, *Game Physics*, 2nd ed. Miamisburg, OH: Elsevier Science, 2010.

[3] I. Millington, *Game Physics Engine Development: How to Build a Robust Commercial-Grade Physics Engine for Your Game*, 2nd ed. San Francisco, CA: Morgan Kaufmann Publishers Inc., 2010.

[4] E. Catto. *Box2D, a 2D physics engine for games* [Online]. Available: http://box2d.org/ (Date Last Accessed, June 30, 2014).

[5] D. Terzopoulos, J. Pltt, A. Barr, D. Zeltzer, A. Witkin, and J. Blinn, "Physically-based modeling: Past, present, and future," *SIGGRAPH Comput. Graph.*, vol. 23, no. 5, pp. 191 – 209, July 1989.

[6] D. Baraff, "Fast contact force computation for nonpenetrating rigid bodies," in *Proc. of the 21st Annu. Conf. on Comp. Graph. and Interactive Techn.*, SIGGRAPH '94, Orlando, FL, 1994, pp. 23–34.

[7] N. Foster and D. Metaxas, "Realistic animation of liquids," *Graphical Models and Image Process.*, vol. 58, no. 5, 1996, pp. 471–483.

[8] A. Witkin, D. Baraff. (1997, Sept.). *Physically Based Modeling: Principles and Practice* [Online]. Available: http://www.cs.cmu.edu/˜baraff/sigcourse/ (Date Last Accessed, June 20, 2014).

[9] K. Erleben, J. Sporring, K. Henriksen, and H. Dohlmann, *Physics-based Anmiation*. Rockland, MA: Charles River Media, Inc., 2005.

[10] F. Pfeiffer and C. Glocker, *Multibody Dynamics with Unilateral Contacts*. Weinheim, Germany: Wiley, 1996.

[11] S. Berard, "Using Simulation for Planning and Design of Robotic Systems with Intermittent Contact," Ph.D. dissertation, Dept. of Comp. Sci., RPI, Troy, NY, 2009.

[12] S. Berard, B. Nguyen, K. Anderson, and J. Trinkle, "Sources of error in a rigid body simulation of rigid parts on a vibrating rigid plate," *ASME J. of Computational and Nonlinear Dynamcis*, vol. 5, no. 4, Jan. 2010.

[13] W. Macaluso, "Exploring the Domain of Applicability of Simulated 2D Rigid Body Dynamical Systems," M.S. thesis, Dept. of Comp. Sci., RPI, Troy, NY, 2012.

[14] E. Haug, S. Wu, and S. Yang, "Dynamic mechanical systems with coulomb friction, stiction, impact and constraint addition-deletion—i: Theory," *Mechanisms and Mach. Theory*, vol. 21, no. 5, pp. 407–416, Sept. 1986.

[15] R. I. Leine and H. Nijmeijer, *Dynamics and Bifurcations of Non-Smooth Mechanical Systems*. Lecture Notes in Applied and Computational Mechanics, New York, NY: Springer, 2004.

[16] P. Lötstedt, "Coulomb friction in two-dimensional rigid body systems," *ZAMM - J. of Appl. Mathematics and Mechanics / Zeitschrift fr Angewandte Mathematik und Mechanik*, vol. 61, no. 12, pp. 605 – 615, Nov. 1981.

[17] J. Moreau and P. Panagiotopoulos, *Unilateral Contact and Dry Friction in Finite Freedom Dynamics*. Montpellier, France: Springer, 1988.

[18] D. Stewart and J. C. Trinkle, "An implicit time-stepping scheme for rigid body dynamics with inelastic collisions and coulomb friction," *Int. J. for Numerical Methods in Eng.*, vol. 39, no. 15, pp. 2673 – 91, Aug. 1996.

[19] A. Kurdila, J. Junkins, and S. Hsu, "Lyapunov stable penalty methods for imposing holonomic constraints in multibody system dynamics," *Nonlinear Dynamics*, vol. 4, no. 1, pp. 51 – 82, Feb. 1993.

[20] J. M. Goicolea and J. C. G. Orden, "Dynamic analysis of rigid and deformable multibody systems with penalty methods and energymomentum schemes," *Comp. Methods in Appl. Mechanics and Eng.*, vol. 188, no.4, pp. 789 – 804, Aug. 2000.

[21] E. Drumwright, "A fast and stable penalty method for rigid body simulation," *IEEE Trans. on Visualization and Comp. Graph.*, vol. 14, no. 1, pp. 231–240, Jan. 2008.

[22] E. Kokkevis, "Practical physics for articulated characters," in *Game Developers Conf.*, San Jose, CA, 2004.

[23] D. Baraff, "Non-penetrating rigid body simulation," in *SIGGRAPH '95 Course Note 34. ACM SIGGRAPH*, Los Angeles, CA, 1993.

[24] N. Koenig, A. Howard, OSRF. *Gazebo Simulator* [Online]. Available: http://gazebosim.org/ (Date Last Accessed, May 27, 2014).

[25] K. Hauser. (2013). *Robust Contact Generation for Robot Simulation with Unstructured Meshes* [Online]. Available: http://www.iu.edu/ motion/klampt/ (Date Last Accessed, May 20, 2014).

[26] E. Coumans. *Bullet Physics Library: An open source collision detection and physics library* [Online]. Available: http://code.google.com/p/bullet/ (Date Last Accessed, July 1, 2014).

[27] R. Smith. *Open Dynamics Engine* [Online]. Available: http://www.ode.org/ (Date Last Accessed, May 27, 2014).

[28] Havok Inc. *Havok Physics Engine* [Online]. Available: http://www.havok.com/products/physics (Date Last Accessed, Apr. 27, 2014).

[29] NVIDIA. *PhysX* [Online]. Available: http://www.geforce.com/hardware/technology/physx (Date Last Accessed, May 27, 2014).

[30] MSC Software. *Adams* [Online]. Available: http://www.mscsoftware.com/product/adams (Date Last Accessed, May 27, 2014).

[31] J. J. Craig, *Introduction to Robotics: Mechanics and Control*, 3rd ed. Upper Saddle River, NJ: Pearson Prentice Hall, 2005.

[32] J. Wen. (2010). *Robotics 1 course notes* [Online]. Available: http://cats-fs.rpi.edu/ wenj/ECSE448F10/ (Date Last Accessed, Dec. 20, 2010).

[33] J. C. Trinkle. (2011). *Robotics 2 course notes* [Online]. Available: http://www.cs.rpi.edu/ trink/Courses/RoboticsII/RoboticsII.html (Date Last Accessed, May 15, 2011).

[34] D. Prattichizzo and J. C. Trinkle, "Grasping," in *Handbook on Robotics*, New York, NY: Springer, 2008, pp. 671–700.

[35] I. Newton, *Philosophi Naturalis Principia Mathematica*. London, England: J. Societatis Regiae ac Typis J. Streater, 1687.

[36] J. Hermann, *Phoronomia, Sive De Viribus et Motibus Corporum Solidorum et Fluidorum Libri Duo*. Amstelædami, apud R. & G Wetstenios, 1716.

[37] L. Euler, *Theoria Motus Corporum Solidorum seu Rigidorum*. Ghent, Belgium: Rostock & Greifswald, 1765.

[38] S. C. Billups and K. G. Murty, "Complementarity problems," *J. Comput. Appl. Math.*, vol. 124, no. 12, pp. 303–318, Dec. 2000.

[39] J. Bender, K. Erleben, and J. C. Trinkle, "Interactive simulation of rigid body dynamics in computer graphics," *Comput. graph. forum*, vol. 33, no. 1, pp. 246–270, Feb. 2012.

[40] M. Ferris and T. Munson, "Complementarity problems in gams and the path solver," *J. of Econ. Dynamics and Control*, vol. 24, no. 2, pp. 165 – 88, July 2000.

[41] R. Cottle, J. Pang, and R. Stone, *The Linear Complementarity Problem*. Classics in Applied Mathematics, Philadelphia, PA: Society for Industrial and Applied Mathematics, 1992.

[42] K. Erleben, "Numerical methods for linear complementarity problems in physics-based animation," in *ACM SIGGRAPH 2013 Courses*, SIGGRAPH '13, Anaheim, CA, 2013, pp. 8:1–8:42.

[43] E. Todorov, "A convex, smooth and invertible contact model for trajectory optimization," in *IEEE Int. Conf. on Robotics and Automation*, Shanghai, China, 2011, pp. 1071–1076.

[44] A. Signorini, "Questioni di elasticità non linearizzata e semilinearizzata.," *Rendiconti Di Matematica e Delle Sue Applicazioni, V. Serie*, vol. 18, no. 5, pp. 95–139, 1959.

[45] C.-A. de Coulomb, *Thorie des Machines Simples, en Ayant gard au Frottement de Leurs Parties et la Roideur des Cordages*. Paris, France: Bachelier, 1821.

[46] P. Song, J. Trinkle, V. Kumar, and J.-S. Pang, "Design of part feeding and assembly processes with dynamics," in *IEEE Int. Conf. on Robotics and Automation*, New Orleans, LA, 2004, pp. 39–44.

[47] N. I. Badler, K. H. Manoochehri, and D. Baraff, "Multi-dimensional input techniques and articulated figure positioning by multiple constraints," in *Proc. of the 1986 Workshop on Interactive 3D Graph.*, I3D '86, Chapel Hill, NC, pp. 151–169, 1987.

[48] P. M. Isaacs and M. F. Cohen, "Controlling dynamic simulation with kinematic constraints," in *Proc. of the 14th Annu. Conf. on Comp. Graph. and Interactive Techn.*, SIGGRAPH '87, Anaheim, CA, 1987, pp. 215–224.

[49] R. Barzel and A. H. Barr, "A modeling system based on dynamic constraints," in *Proc. of the 15th Annu. Conf. on Comp. Graph. and Interactive Techn.*, SIGGRAPH '88, Atlanta, GA, 1988, pp. 179–188.

[50] M. Anitescu and F. A. Potra, "Formulating dynamic multi-rigid-body contact problems with friction as solvable linear complementarity problems," *Nonlinear Dynamics*, vol. 14, pp. 231–247, 1997.

[51] M. Anitescu and F. Potra, "A time-stepping method for stiff multibody dynamics with contact and friction," *Int. J. for Numerical Methods in Eng.*, vol. 55, no. 7, 2002, pp. 753–84.

[52] J. Bender and A. Schmitt, "Fast dynamic simulation of multi-body systems using impulses," in *Workshop on Virtual Reality Interactions and Physical Simulations*, Madrid, Spain, 2006, pp. 81–90.

[53] F. Gauss, *Theoria Combinationis Observationum Erroribus Minimis Obnoxiae*. Ghent, Belgium: H. Dieterich, 1823.

[54] P. L. von Seidel, "Uber ein verfahren die gleichungen, auf welche die methode der kleinsten quadrate fhrt, sowie linere gleichungen berhaupt durch successive annherung aufzulsen," *Abhandlungen der Bayerischen Akademie, Dritte Abteilung,*, vol. 11, 1873, pp. 81–108.

[55] W. Kahan, "Gauss-Seidel Methods of Solving Large Systems of Linear Equations," Ph.D. dissertation, Dept. of Math., Univ. of Toronto, Toronto, Canada, 1958.

[56] C. G. J. Jacobi, "Uber eine neue auflosungsart der bei der methode der kleinsten quadrate vorkommenden linearen gleichungen," *Astronomische Nachrichten*, vol. 22, pp. 297–306, 1845.

[57] J. L. Morales, J. Nocedal, and M. Smelyanskiy, "An algorithm for the fast solution of symmetric linear complementarity problems," *Numerische Mathematik*, vol. 111, no. 2, pp. 251–266, Nov. 2008.

[58] K. Erleben. (2010, Nov.) *Linear Complementarity Problems, A short Introduction to Definitions and Numerical Methods* [Online]. Available: http://image.diku.dk/kenny/download/vriphys10_course/lcp.pdf (Date Last Accessed, Apr. 27, 2014).

[59] D. M. Young Jr., "Iterative Methods for Solving Partial Difference Equations of Elliptic Type," Ph.D. dissertation, Dept. of Math., Harvard Univ., Cambridge, MA, 1950.

[60] L. F. Richardson, "The approximate arithmetical solution by finite differences of physical problems involving differential equations, with an application to the stresses in a masonry dam," *Philosophical Trans. of the Royal Soc. A*, vol. 210, pp. 307–357, Jan. 1910.

[61] D. Baraff, "Dynamic simulation of non-penetrating rigid bodies," Ph.D. dissertation, Dept. of Comp. Sci., Cornell Univ., Ithica, NY, 1992.

[62] J. D. Cohen, M. C. Lin, D. Manocha, and M. Ponamgi, "I-collide: An interactive and exact collision detection system for large-scale environments," in *Proc. of the 1995 Symp. on Interactive 3D Graph.*, I3D '95, Monterey, California, 1995, pp. 189–196.

[63] S. Gottschalk, M. C. Lin, and D. Manocha, "Obbtree: a hierarchical structure for rapid interference detection," in *Proc. of the 23rd Annu. Conf. on Comp. Graph. and Interactive Techn.*, SIGGRAPH '96, New Orleans, LA, 1996, 171–180.

[64] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, "A fast procedure for computing the distance between complex objects in three-dimensional space robotics and automation," *IEEE J. of Robotics and Automation*, vol. 4, no. 2, pp. 193–203, Apr. 1988.

[65] S. Cameron, "Enhancing GJK: computing minimum and penetration distances between convex polyhedra," in *IEEE Int. Conf. on Robotics and Automation*, Albuquerque, NM, 1997, pp. 3112–3117.

[66] B. Mirtich, "V-clip: fast and robust polyhedral collision detection," *ACM Trans. Graph.*, vol. 17, no. 3, pp. 177–208, Jul. 1998.

[67] M. Lin and J. Canny, "A fast algorithm for incremental distance calculation," in *IEEE Int. Conf. on Robotics and Automation*, Cincinnati, OH, 1991, vol. 2, pp. 1008–1014.

[68] S. A. Ehmann and M. C. Lin, "Accelerated proximity queries between convex polyhedra by multi-level voronoi marching," in *Int. Conf. on Intelligent Robots and Syst.*, Takamatsu, Japan, 2000, pp. 2101–2106.

[69] S. M. LaValle, *Planning Algorithms*. New York, NY: Cambridge University Press, 2006.

[70] B. R. Donald, "Local and global techniques for motion planning," Ph.D. dissertation, Dept. of Elec. Eng. and Comp. Sci., MIT, Cambridge, MA, 1984.

[71] J.-C. Latombe, *Robot Motion Planning*. Norwell, MA: Kluwer Academic Publishers, 1991.

[72] M. C. Lin, "Efficient Collision Detection for Animation and Robotics," Ph.D. dissertation, Dept. of Comp. Sci., Univ. of California, Berkeley, CA, 1993.

[73] P. Jiménez and F. Thomas and C. Torras, "Collision Detection Algorithms for Motion Planning," in *Lecture Notes in Control and Information Sciences*, vol. 4. New York, NY: Springer, 1998, pp. 305–343.

[74] B. Nguyen, "Locally Non-Convex Contact Models and Solution Methods for Accurate Physical Simulation in Robotics," Ph.D. dissertation, Dept. of Comp. Sci., RPI, Troy, NY 2011.

[75] D. M. Flickinger and J. C. Trinkle, "Evaluating the performance of constraint formulations for multibody dynamics simulation," in *Proc. of the ASME Int. Design Eng. Tech. Conf. & Comp. and Inform. in Eng. Conf.*, Portland, OR, 2013.

[76] J. Williams. (2014, Feb.). *A complementarity based contact model for physically accurate treatment of polytyopes in simulation* [Online]. Available: http://www.birs.ca/events/2014/5-day-workshops/14w5147/videos/watch/201402201024-Williams.mp4 (Date Last Accessed, Mar. 4, 2014).

[77] J. Hu, J. E. Mitchell, J. Pang, K. P. Bennett, and G. Kunapuli, "On the global solution of linear programs with linear complementarity constraints," *SIAM J. on Optimization*, vol. 19, no. 1, pp. 445 – 471, May 2008.

[78] N. Chakraborty, S. Berard, S. Akella, and J. Trinkle., "A geometrically implicit time-stepping method for multibody systems with intermittent contact," *Int. J. of Robotics Res.*, vol. 32, no. 10, pp. 426 – 445, Oct. 2013.

[79] D. E. Stewart, "Convergence of a timestepping scheme for rigidbody dynamics and resolution of painlev's problem," *Archive for Rational Mechanics and Anal.*, vol. 145, no. 3, pp. 215 – 260, Dec. 1998.

[80] D. E. Stewart, "Rigid-body dynamics with friction and impact," *SIAM Rev.*, vol. 42, no. 1, pp. 3–39, Mar. 2000.

[81] M. T. Mason and Y. Wang, "On the inconsistency of rigid-body frictional planar mechanics," in *IEEE Int. Conf. on Robotics and Automation*, Philadelphia, PA, 1988, vol. 1, pp. 524–528.

[82] F. Gnot and B. Brogliato, "New results on painlev paradoxes," *European J. of Mechanics - A/Solids*, vol. 18, no. 4, pp. 653 – 677, July 1999.

[83] E. Drumwright and D. A. Shell, "A robust and tractable contact model for dynamic robotic simulation," in *Proc. of the 2009 ACM Symp. on Appl. Computing*, SAC '09, 2009, pp. 1176–1180.

[84] J. Williams, Y. Lu, S. Niebe, M. Andersen, K. Erleben, and J. Trinkle, "Rpi-matlab-simulator: A tool for efficient research and practical teaching in multibody dynamics," in *Workshop on Virtual Reality Interaction and Physical Simulation*, Lille, FR, 2013.

[85] J. W. Eaton, *GNU Octave Manual*. Bristol, United Kingdom: Network Theory Limited, 2002.

[86] A. Witkin, K. Fleischer, and A. Barr, "Energy constraints on parameterized models," in *Comp. Graph.*, vol. 21, no. 4, pp. 225–232, Aug. 1987.

[87] J. Burkardt. *LEMKE solver for linear complementarity problems* [Online]. Available: http://people.sc.fsu.edu/~jburkardt/m_src/lemke/lemke.html (Date Last Accessed, May 20, 2014).

[88] Y. Lu, C. Lacoursiere, J. Williams, and J. Trinkle, "Standard interface for data analysis of solvers in multibody dynamics," in *Canadian Conf. on Nonlinear Solid Mechanics (CanCNSM)*, Montreal, Quebec, Canada, 2013.

[89] Y. Lu, J. Williams, C. Lacoursiere, and J. Trinkle, "A framework for problem standardization and algorithm comparison in multibody system," in *ASME Int. Design and Eng. Tech. Conf. and Comp. and Inform. in Eng.*, Buffalo, NY, 2014.

[90] N. Chakraborty and J. Peng, "Proximity queries between convex objects: An interior point approach for implicit surfaces," in *IEEE Int. Conf. on Robotics and Automation*, Orlando, FL, 2006, pp. 1910–1916.