LOCALLY NON-CONVEX CONTACT MODELS AND SOLUTION METHODS FOR ACCURATE PHYSICAL SIMULATION IN ROBOTICS.

By

Binh Nguyen

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

________________________
John E. Mitchell, Member

________________________
Christopher D. Carothers, Member

________________________
Barbara Cutler, Member

Rensselaer Polytechnic Institute
Troy, New York

June 2011
(For Graduation August 2011)
CONTENTS

LIST OF TABLES .......................................................... vi
LIST OF FIGURES ......................................................... vii
ABSTRACT ............................................................... xiii
1. Introduction .......................................................... 1
   1.1 Contributions and thesis organization ......................... 4
2. Physical and mathematical background of rigid body simulation .... 9
   2.1 Physical simulation ............................................... 9
   2.2 Physical models .................................................. 10
      2.2.1 Law of motion .............................................. 11
      2.2.2 Rigid body model ......................................... 12
      2.2.3 Friction model ............................................. 12
         2.2.3.1 Coulomb friction ................................... 13
      2.2.4 Restitution models ........................................ 15
   2.3 Mathematical models ............................................ 16
      2.3.1 Rigid body dynamics background .......................... 16
         2.3.1.1 Position .............................................. 16
         2.3.1.2 Orientation ......................................... 18
         2.3.1.3 Velocity ............................................ 20
         2.3.1.4 Torque and force .................................. 22
      2.3.2 Complementarity problem .................................. 24
      2.3.3 Complementarity formulation of physical simulation ....... 26
         2.3.3.1 Newton-Euler equation ............................... 26
         2.3.3.2 Bilateral (joint) constraints ......................... 27
         2.3.3.3 Non-penetration constraint ............................ 29
         2.3.3.4 Friction constraint .................................. 31
      2.3.4 Instantaneous formulations of constrained dynamics .......... 36
         2.3.4.1 Nonlinear DCP formulation ............................ 36
         2.3.4.2 Linear DCP formulation ............................... 36
         2.3.4.3 Comparison with physical models ....................... 37
   2.4 Computer models of physical simulation .......................... 38
4.2.2.1 Simple contact .................................................. 76
4.2.2.2 General contact case ........................................... 80
4.2.2.3 General contact case with predefined contact face . 81
4.3 Mathematical properties and solution methods .................. 82
  4.3.1 Solutions method for extended contact ......................... 82
  4.3.2 Solution method for extended contact with predefined acting
      normal .......................................................... 83
4.4 Numerical results .................................................. 84
  4.4.1 Box-ramp ....................................................... 84
  4.4.2 Peg in hole ..................................................... 85
4.5 Conclusion and future works ....................................... 86
5. Collision detection in physical simulation ........................ 88
  5.1 Introduction ....................................................... 88
  5.2 Problem with current contact model ............................ 90
  5.3 Collision detection methods .................................... 94
    5.3.1 Collision detection: The Method of Separating Axes (SAT) . 94
    5.3.2 Collision detection: The Gilbert-Johnson-Keerthi and Expand-
        ing Polytope Algorithm method (GJK/EPA) ..................... 98
    5.3.3 Custom collision detection routines and other collision detec-
        tion methods .................................................. 100
  5.4 Collision detection for non-convex contact model ............... 103
    5.4.1 Basic rules for the new collision detection ................. 103
    5.4.2 Extended contact representation ............................ 104
  5.5 Implementation of new collision detection ....................... 107
    5.5.1 Acting edge in extended contact ........................... 113
    5.5.2 Negative bound $\epsilon^-$ in extended contact ............. 114
  5.6 Extension of new collision detection in 3D ..................... 115
  5.7 Conclusion and future works .................................... 117
6. Solution methods for physical simulation problems .............. 124
  6.1 Introduction ....................................................... 124
  6.2 Levenberg-Marquardt based solution methods .................... 126
    6.2.1 Simple semi-smooth solver ................................ 127
    6.2.2 Over-constrained semi-smooth solver ....................... 130
LIST OF TABLES

3.1 List of popular physics engines. . . . . . . . . . . . . . . . . . . . . . 44

6.1 Min,max and average running time of three solvers: PATH, normal and over-constrained Levenberg-Marquardt. . . . . . . . . . . . . . . . . . 138
LIST OF FIGURES

2.1 Steps of physical simulation. ........................................ 10

2.2 A force moves a free body to a new position and changes its velocity. 11

2.3 Coulomb friction: The relation for friction force, normal force, relative velocity and coefficient of friction. ......................... 14

2.4 Position and orientation of a rigid body specified by a body inertia (or fixed) reference frame $A$ with origin at the center of mass of $A$, denoted $A^\ast$. ................................................................. 17

2.5 System of external forces acting on a body. ............................ 22

2.6 Popular mechanical joints ............................................. 28

2.7 A contact between two bodies. ......................................... 29

2.8 Quadratic friction cone and a linearized version with 8 friction directions. 34

3.1 A vibrational part feeder that shakes parts into recesses to be picked up by a manipulator. The system exhibits very transient contact modes (image taken from Brian Mirtich thesis [Mir96]) ......................... 41

3.2 Two similar cases with different simulation outcomes. In both cases, the big box was dropped into two thin rectangular obstacles below. The normal impulses are generated along the penetration distance which is defined as the shortest distance to separate two bodies. In 3.2a, the outcome is as expected but in 3.2b, the direction that the box and the left obstacle achieve shortest penetration distance is not the correct normal direction. ................................................................. 47

3.3 Screenshots at different times of the box stack scene running in dVC3d 48

3.4 Screenshots at different times of box stack scene in Bullet in three runs with same initial condition like 3.3 and the same model parameters. One row contains frames of a simulation and the columns correspond to time $t=2,4,6,8$ s. ................................................................. 48

3.5 A design of PPODs: Programmable Parts-feeding Oscillatory Devices by Tom Vose et al. [VUL09b], [VUL09a]. It can be shown that every small-amplitude periodic vibration of the plate is associated with a unique velocity field. Parts on the plate’s surface can be modeled as flowing through the velocity field. Different plate motions generate different fields. ................................................................. 51
3.6 The plate is kinematically controlled by the vector function \( g(t) \). There are three forces acting on the part: the force due to gravity \( \lambda_{\text{app}} \), the non-penetration constraint force \( W_n \), and the frictional force \( W_f \).

3.7 Direct comparison of the particle’s position between our simulation results and those of Vose et al., an experiments verified, closed-form simulator, for the Centrifuge motion.

3.8 Direct comparison of the particle’s velocity between our simulation results and those of Vose et al., an experiments verified, closed-form simulator, for the Centrifuge motion.

3.9 Difference between AP and ST formulation of the gap between two bodies in an active contact.

3.10 A rigid body with sphere shape stays inside two rectangular obstacles. In AP, the system has solution but not in ST.

3.11 Main position update loop of optimization-based animation.

3.12 The ”house of card” test case in [KSJP08]. A house of cards is made out of rigid cards with high friction. An accurate friction model is needed to keep it stable. (Image taken from [KSJP08]).

3.13 prox function for a convex set.

3.14 Speculative contact (S) and one-shot (O) collision detection tests in dVC3d. First row: no S, no O, second row: S only, third row: O only, fourth row: both S and O. Each column corresponds to \( t=\{0,0.33,0.67,1\} \) s.

3.15 A case in cylinder-box collision with 4 contact points.

3.16 Same setup as in 3.14 but using box instead of cylinder.

3.17 A simple friction scene with 9 rigid bodies.

3.18 Running time of 10 steps of different friction models measured with setup in figure 3.17.

3.19 Convex friction model failed when the second box is close enough to the first one. Both linearized Coulomb and gaming friction models work as expected.

3.20 A ghosting trail of 200 frames when simulating a triangle sliding on the ground. Convex friction model shows anomalous behavior where the triangle jumps instead of slides on the ground.

3.21 Linearized Coulomb friction model shows correct result when simulating the same scene in figure 3.20.
4.1 Locally non-convex free space .............................................. 71

4.2 A more complicated scene resembling a robot moving through a room .............................................. 72

4.3 A test case where robot need to insert a rectangular block into a closely fitted hole .............................................. 72

4.4 Left: current contact model (P,2) virtually extends edge 2, thus prevent P from entering region below the dotted line. Right: new extended contact model (P,[1,2,3]) accurately capture the shape of obstacle and P can move freely outside dotted region .............................................. 74

4.5 There are cases where user can use current information to choose the set of edges to minimize the chance of penetration.(a): the vertex P of a polygon moves relatively slowly and is close to the obstacle so only edge 1 is needed.(b):the vertex moves fast and the object below is rotating so edge 1,2 and 3 need to be included.(c):Speeds of the vertex and object’s spinning are very high, so we may need to consider adding ALL edges .............................................. 76

4.6 New method, time step $h = 0.008$ .............................................. 84

4.7 Old method, time step $h = 0.008$ .............................................. 85

4.8 New method, time step = 0.0125 .............................................. 85

4.9 Old method, time step = 0.0125 .............................................. 86

5.1 Diagram of physical simulation .............................................. 88

5.2 Bullet Physics engine [Cou06] collision detection and dynamics. Each square correspondents to a small steps in Forward Dynamics and Collision Detection modules. Both modules operate on various parts of the system of rigid bodies states. There are some parts are shared between the two .............................................. 89

5.3 The particle is inside the disk centers at a vertex of the obstacle with $\epsilon$ radius .............................................. 90

5.4 Effective region of the obstacle (shaded) if only (particle,left edge) contact is active .............................................. 91

5.5 Effective region of the obstacle (shaded) if only (particle,right edge) contact is active .............................................. 91

5.6 Effective region of the obstacle (shaded) if both (particle,left edge) and (particle,right edge) contacts are active .............................................. 92

5.7 Effective region of the obstacle (shaded) if only the special (particle,vertex) contact is active .............................................. 92
5.8 Havok® Physics engine demo simulating a large box stacking scene in realtime. ................................................................. 93

5.9 Second-order information can help estimate future position. .............................................................. 94

5.10 A separating axis between two non-penetrating shapes. ................................................................. 95

5.11 There are no separating axis between two penetrating shapes. ................................................................. 95

5.12 Hard (fuzzy 1,2 and 3) and very hard (extreme fuzzy 1 and 2) cases for SAT contact generation (Erin Catto, Game Developer Conference 2007 [Cat07]). ................................................................. 97

5.13 Three frames show how we can build a good contact manifold (two contact points) between a box and the ground (images taken from Erin Catto GDC2007 talk [Cat07]). ................................................................. 99

5.14 In SIGGRAPH ASIA 2009 course, Jumpei Tsuda [Tsu09] advised to leave a small penetrations for stable simulation. ................................................................. 100

5.15 Bullet collision matrix which lists the method the collision detection routine between two types of shapes. ................................................................. 101

5.16 The new contact proposed in chapter 4 can accurately model non-convex polyhedral free space. ................................................................. 103

5.17 Example of extended contact: Particle vs Triangle. ................................................................. 105

5.18 Example of extended contact: Triangle vs Rectangle. ................................................................. 106

5.19 Example of extended contact: Triangle vs Quadrilateral with penetration. ................................................................. 106

5.20 Example of extended contact: Triangle vs Quadrilateral with deep penetration. ................................................................. 107

5.21 Two cases when calculating distances of vertex \( P \) to the segment defined by \( P_0, P_1 \). \( d \) is the distance between two vertices. ................................................................. 111

5.22 Distance from \( P \) to edge \( e \) is 0 but the vertex is not close to the rectangle. ................................................................. 111

5.23 200 frames of two simulations with two different \( \epsilon^+ \). Initially, both the box and the triangle are dropped from above. The triangle, which is lower, hits the ground first and then collide with the falling box. From there, the two shapes are separated and move to two different walls. ................................................................. 119

5.24 150 frames of two simulations with two different heuristic of choosing acting edge. ................................................................. 120
5.25 Distances from vertex $P$ to three edges $e_1, e_2, e_3$ in non-touching mode of extended contact. .................................................. 120

5.26 Possible future position in simulation of $P$ with different acting edge. .................................................. 121

5.27 A particle is moving close to the box’s vertex. .................................................. 121

5.28 200 frames of two simulations with two different $\epsilon^-$. You can see that in the case where $\epsilon^- = 0$, the slight push moves the box a little farther to the left than the case with $\epsilon^- \neq 0$. .................................................. 121

5.29 The two shapes are intersected even when there s no vertex of them is inside the other shape. .................................................. 122

5.30 Edge-Edge intersection in 3D. .................................................. 122

5.31 Cases of different number of new vertices generated to handle 3D edge-edge contacts. .................................................. 123

6.1 The value of $\phi_{FB}(a, b)$ function is not a good indication of the complementarity gap $\max(a, 0) \max(b, 0)$ in the first orthant. .................................................. 130

6.2 List of softwares to solve sparse linear systems using direct methods. Screen captured from Xiaoye Sherry Li’s survey [Li10]. .................................................. 134

6.3 dVC2d simulation results (320 time steps) of a scene contains a box and a triangle using non-convex formulation and collision detection. .................................................. 137

6.4 [Example one] The running time of different solvers of the simulation in figure 6.3. .................................................. 138

6.5 [Example one] The problem size at different time steps of the simulation in figure 6.3. .................................................. 139

6.6 [Example one] The values of merit function of normal and over-constrained Levenberg-Marquardt solvers of the simulation in figure 6.3. .................................................. 140

6.7 dVC2d simulation results (320 time steps) of a scene containing 4 boxes and a ramp using Stewart-Trinkle formulation. .................................................. 141

6.8 [Example two] The running time of different solvers of the simulation in figure 6.7. .................................................. 142

6.9 [Example two] The problem size at different time steps of the simulation in figure 6.7. .................................................. 143

6.10 [Example two] The values of merit function of normal and over-constrained Levenberg-Marquardt solvers of the simulation in figure 6.7. .................................................. 143
6.11 A box stacking scene with large number of boxes simulated using dVC3d inside Blender 3D................. 144
6.12 [Example three] The size of the main matrix in during simulation of the scene in figure 6.11........... 144
6.13 [Example three] The running time of the two solvers in 121 time steps of the scene in figure 6.11........ 145
6.14 [Example three] The complementarity gap of the two solvers in 121 time steps of the scene in figure 6.11 145
7.1 dVC3d simulating a robotics hand design inside Blender 3D 2.51. The fingers were created and setup inside Blender........... 148
7.2 dVC3d Matlab................................................................. 149
7.3 A screenshot of simple dVC3d Matlab simulation of two boxes sitting on the ground............................ 150
7.4 Simulation result of the simple Matlab example above.......................... 152
ABSTRACT

In recent years, physical simulation has been becoming increasingly popular. In robotics, new designs of robots, new autonomous multi-robots motion planning, as well as new grasping strategies can be easily tested on computer thanks to physical simulation. In entertainment industry, physical simulation plays a significant role in the success of many computer games, block buster movies. Despite such achievements, in general, simulating multi-rigid bodies with friction is far from a solved problem. In fact, most current physical simulation libraries typically focus on only one type of application by making certain modeling choices. Such decisions affect the accuracy as well as performance of the simulations. I start the thesis by the discussions of physical, mathematical and computational backgrounds of physical simulation. Each of them represents a level of abstraction that involves a certain set of modeling choices. One main question I try to answer in this thesis is, what are the effects of different decisions on contact and friction approximation models and solution methods to the simulation’s performance and accuracy. To answer this question, I present my studies on the modeling choices of existing simulation methods and the effects on the outcomes. The second focus of the thesis is to develop new models and solution methods that are faster and more accurate than current state-of-the-art methods. The new methods’ development was inspired by the understandings of the advantages and disadvantages of existing methods. Generally, most computational tasks of the new methods proposed in this thesis are designed to run in parallel to be aligned with current multi-core evolution of computer hardware.
CHAPTER 1
Introduction

Physical simulation has become increasingly popular in the last ten years. In robotics, it enabled researchers to study robots’ interactions in more complex environments through computer simulation. Unlike ten years ago, most current robotics software platforms, such as Player [GV01], Webots [MI04], OpenRAVE [DK08], and OpenHRP3 [AT07], are now capable of testing their robots’ systems on computer using physical simulation. In biotechnologies, physical simulation has been used to study many important problems like protein folding, drug making and testing. One notable biology simulation library, Simbios [BcaS04], was used in a number of large number of biomedical projects. However, the most visible applications of physical simulation are in the gaming and film industries. Ten years ago, a simple multi rigid body simulation library could usually cost thousands of dollar, but now many capable gaming physics engines are free. Some are even open source. As the result of that, many important content creation editors for computer animations and games have built-in physics engines. For example, Blender 3D, a popular open-source 3D content creation tool, supports Bullet Physics engine [Cou06] internally. The Bullet Physics engine is also an open source, free for commercial use and has been used in an impressive number of successful commercial games as Grand Theft Auto 4, Madagascar and developments of movies such as 2012, Hancock and Bolt.

Given such success stories, simulating frictional multi-rigid bodies with intermittent contacts is far from a solved problem. Most of the achievements to date are due to the rapid increase of computational power, increased software engineering maturity of the libraries and better understanding of the fifteen years old driving theories behind multi rigid body simulation [ST96], [AP97]. In fact, most current
physical simulation libraries use a variance of the method proposed in Stewart-Trinkle [ST96] and Anitescu-Potra [AP97]. For example, popular physics engines such as NVidia PhysX, Havok and Bullet deviate from the theories by using a simplification of Coulomb friction to get a mathematical model that is much easier to solve. While these simplifications give the simulations a plausible sense of friction, they do not have an underlying scientific theory to back up. Consequently, the mentioned physics engines are unsuitable to use in a robotics or machine design applications that usually require high level of accuracy. The other problem with scientific usage of these engines is that they may not produce deterministic results. For example, in Bullet, two consecutive simulations of the same block stacking scene give different outcomes although both results look plausible to human eyes. Moreover, there are even problems with these physics engines when used in their intended applications. One of their biggest requests for improvement is more stable boxes-stacking simulation. Interestingly, the Stewart-Trinkle method [ST96], if correctly implemented, has already solved this problem. Another pitfall when using a physics engine is what described as tunneling effect where a fast object can unrealistically move past a wall in simulation. Collision detection is also an area that needs improvements. Right now there are no precise rules on how collision detection should work. Contact generation is a challenging task that usually involves guessing future positions of the bodies. Each physics engine has its own and incompatible collision detection routines which are incompatible with other engines. Collision detection is also the main obstacle to move the engines into the parallel domain while the other important component of a physics engine, the iterative constraint solver, could already be run in parallel.

Compared to the gaming physics engines mentioned above, there are much lesser efforts to develop scientific oriented physics engines. Many of them are usually not in a form of user friendly library that can be used in wide varieties of problems
but just focus mainly on the tasks the developers interested in. Currently, beside dVC2d, an effort led by Jeff Trinkle to implement theories described in Stewart-Trinkle [ST96] and Anitescu-Potra [AP97], there is no other free and open source scientific-focused physical simulation engine. The only other free for educational use but not open-source physics engine of such a type is Chrono:Engine developed by Alessandro Tasora and Mihai Anitescu, which employs convex relaxation of Coulomb friction and Bullet for collision detection. The limited number of scientific physical simulation libraries forces robotics research library mentioned in the first paragraph to use (or misuse) gaming oriented physics engines. The main obstacle to implement a three-dimensional version of Stewart-Trinkle method is the lack of proper collision detection. Collision detection libraries in gaming physics engines do not work well with this method directly.

There are researches on new theories of physical simulation of multi rigid bodies. Leine and Glocker have been leading an attempt to reformulate the physical simulation problems with prox-functions [LG03]. The method has the advantage of making use of convex function analysis instead of less developed complementarity theory, but there is no known attempt to provide a general physics engine implementation. Also, Saeed Ebrahimi, in his PhD thesis [Ebr07], found no clear advantages in using prox-functions over complementarity theory. Another relevant study by Nilanjan Chakrabory, Steven Berard led by Jeff Trinkle and Srinivas Akella [CBAT07b] on a geometrically implicit, nonlinear simulation method for hyper quadric shapes pushes the accuracy limit of physical simulation higher. The main problem for this method is that its mathematical model is very hard to solve. So a general physics engine using this method will not be available any time soon.

In conclusion, physical simulation has become a standard in robotics, biotechnologies, machine design and many other scientific fields involved interaction between physical objects. It is especially popular in the entertainment industries.
Almost all new computer games and films make use of physical simulation. This increase in popularity is mostly credited to better implementation techniques and the increase of computing power. Stewart-Trinkle and Anitescu-Potra methods are still the main underlying theories for accurate rigid body physical simulation.

1.1 Contributions and thesis organization

The number of physical simulation applications has grown very fast due to the high availability of commercial: NVidia PhysX, Intel Havok and CMIlabs as well as high quality free or open source physics engines: AMD Bullet, Open Dynamics Engine and Newton Engine. That brings along a problem as most users tend to look for and then only observe certain traits of physical simulation while the engines are in fact designed with different purposes. Over time, the users perceive physical simulation through the outcomes and fail to understand the basic principles and models of simulation. The complexities of physics engines also force most users see them as black boxes. In addition, most of the academic papers in simulation tend to focus solely on a subset of physical simulation models and often skip defining the underlying theories. In this thesis, I start with the definition of physical simulation and the choices of physical models that ultimately define the expected outcomes. Thus, given any simulation scenes, we should be able to tell the correctness of the implementations based on the outcomes. Unfortunately, the number of physical models is quite large for this thesis to address, so I only focus on some of the main ones such as contact, friction, rigid body models.

The main contributions of this thesis are three fold. First, a novel advanced contact model is proposed. The main advantage of the model lies on its ability to capture exactly the free spaces around the bodies while existing model can only approximate. Second, a simple, more accurate and parallel friendly collision de-
tection is developed. This collision detection adheres to a clear and simple set of rules to make it easy to understand and verify. Lastly, two new types of solution methods for physical simulation are proposed. They should improve accuracy and performance of existing solvers. These solution methods are also capable of running in parallel to fully utilize modern computer hardware. In between, I discuss current physical simulation methods and libraries, their advantages and disadvantages and compare them under different settings. At the end, I will describe dVC3d, the first three dimensional physics engine that faithfully implements Stewart-Trinkle time stepper.

The thesis is organized as follows.

In chapter two, I define physical simulation and the set of physical models that ultimately define the outcomes of the system. Any physics engine should pick a set of physical laws, converts them to mathematical equations and constraints then develop methods to solve them using computers. Usually, the more steps needed, the farther away the simulation outcomes is from reality. Because it is a very complicated process that involves a numerical library, it is not trivial to tell if a certain simulation scene outcome is correct. But with the abstract understanding through physical models, we should be able to determine the right results in most cases using our physical interpretations. These abstract views of physics engines can help users to choose the best one for their applications.

Chapter three discusses the modeling decisions of current notable physics engines. Some of the popular choices are contact non-penetration model (that lead to collision detection requirements), friction models (fully implicit Coulomb or linearized friction cone) and solution methods (speed vs accuracy, iterative vs direct). The models studied in this chapter vary from basic penalty method to variants of Anitescu-Potra that being used in most current physics engines to the current state-of-the-art Stewart-Trinkle method. Then, I show how each model decision affects
the simulation outcome through examples. At the end of the chapter, I discuss a number of necessary improvements in order to implement the three-dimensional version of the Stewart-Trinkle formulation.

In chapter four, a new formulation based upon an advanced contact that can model locally non-convex free spaces exactly is proposed. Under current single vertex-feature contact, these non-convex spaces can only be approximated. I present a number of simulation artifacts resulted from these modeling inaccuracies. Finally, I show that the cost of the advanced contact model is approximately the same as current contact model.

With the new advanced contact model, new collision detection methods are needed. Chapter five discusses the underlying principles and implementation details of these methods. Compares to existing collision detection, the new methods are simpler, easier to understand and verify. They also avoid the need to guess future bodies’s positions that has caused many problems to existing methods. I also discuss a simple 2D implementation as well as 3D extension of the methods. Because of their simplicity, the new methods can be run efficiently in parallel on modern hardware.

Chapter six starts with the discussion of current popular physics engines’s solution methods. They can be classified into three categories: iterative based, optimization based and direct solvers. Each of them has its own strengths and weaknesses that make it suitable for certain types of applications. Then, in the next part of the chapter, I propose two types of solution methods that are more adapted to physical simulation problems. One main objective of these new solvers is to take advantage of physical interpretations of the problem to get the solution faster and more accurately. They are built on top of existing numerical solvers that have been developed for many years. All these solvers are also capable of running in parallel to embrace modern hardware’s multi-core development. The first physical simulation solution method starts with a Mixed Linear Complementarity Problem, coverts it
to a system of nonlinear equations through a special function called NCP-function, then uses modern Newton’s method variant Levenberg-Marquardt method to solve the equations. This approach is backed by strong theoretical results where both fast local and global convergence can be established. The second solver takes ideas from the most current optimization based solvers and improves them. It solves the problems by iterating between two convex sub problems. For formulations where friction couples with normal force like Stewart-Trinkle, this type of methods may not be able to guarantee convergence to the original solution. However, we can obtain a bound on the error. For formulations without the friction-normal force coupling, the methods yield exact solutions. The two sub problems of these methods are usually in the form of convex Quadratic Programs, which is an advantage because there are more capable solvers available than complementarity problems. Most of the convex Quadratic Programming solvers for large problem perform matrix decompositions followed by multiple forward and backward solves. Throughout one iteration, the matrix parts of the convex Quadratic Programming associated with sub problems are constant, that means the methods only need to perform two matrix decompositions per time step.

In chapter seven, I describe dVC3d, the accurate physics engine that implements many of the theories mentioned in previous chapters. It can be used as a stand alone C++ physics engine, a Matlab library or as a bundled physics engine inside Blender. At its heart, dVC3d is implemented on top of Bullet Physics engine, the most popular open source physics engines backed by Advanced Micro Devices (AMD), that are being used by thousands of developers. The main reason I choose to use Bullet interface is to make dVC3d appealed to the current large Bullet user base and to take advantage of its well designed application programming interfaces. dVC3d can also be used inside Matlab through a .NET interface. The Matlab toolbox can be used to solve and research small physical simulation problems. The final
piece of software that will be covered in this chapter is a 3D content creation tool, Blender, bundles with dVC3d. It can be used to quickly and intuitively, through Blender user interface, create a simulation scene, run it and then collect the results. In the chapter, I also discuss details of dVC2d Matlab implementation. It serves as the proof of concept of the new advanced contact model, the new collision detection and Levenberg-Marquardt based solution methods.

Chapter eight concludes the thesis and presents a number of directions for future works.
CHAPTER 2
Physical and mathematical background of rigid body simulation

Multi-rigid-body dynamics with intermittent frictional contact is one of the most challenging physical events to simulate. It is an active area of research in robotics, gaming, and graphics. This background chapter begins with the definition of physical simulation and the process of choosing physical models. Then a brief review of rigid body motion will be presented. Next, it incorporates more physical interaction rules into the study of free motion, leading to the famous Newton-Euler equations of motion. Afterwards, a discussion on the complementarity problem, its natural mathematical representation is provided. Lastly, the complementarity based time-stepping formulation of the dynamics is presented.

2.1 Physical simulation

The word *simulation* is defined in the Computing Dictionary as: *attempting to predict aspects of the behavior of some systems by creating an approximate (mathematical) model of it*. So the first step is to build a model for the system we want to simulate. Because we are interested in the real world physical systems, the first model should contain physics laws. Then we need to transform these physical models to mathematical equations, so that the machines can help solving them. However, in our case, the first set of mathematical equations are unsolvable in close form. So we have to simplify and approximate them using a time-stepping method. There are many possible time-stepping methods. They differ in how far their models are from the original. There is a trade-off the physics engine has to make. Either solve a
close approximation of the original equations to get more accurate simulation results using more computing power or achieve less accurate outcomes but work faster.

So we can see that physical simulation is a well defined process of choosing the right models for the computer software to implement. I believe that all physics engines should clearly state their underlying model and stay true to it. For example, a gaming physics engine can choose simple models to achieve desired speed and good enough accuracy while the one aims at machine design application can choose the most accurate models their computer system can solve in acceptable time. Note that, there is no better model in general between the two because they serve different audiences.

In the following sections, I will provide more details for the components of physical simulation.

2.2 Physical models

Here, we are interested in the physical system of rigid bodies so the governing rules are the law of physics. All of the known physics engines share most of the important physical models.

- Newton-Euler as the laws of motion (first and second laws).
• Rigid body as geometric model.

• Coulomb friction as frictional model.

• Either Newton or Poisson hypothesis for restitution model.

2.2.1 Law of motion

Because we are only interested in simulating systems of rigid bodies, so the natural choice for the law of motion is Newton-Euler, an extension of Newton second law. In term of physics, the law can be stated as:

\[
\begin{pmatrix}
\mathbf{F} \\
\mathbf{T}
\end{pmatrix} = \begin{pmatrix}
m & 0 \\
0 & \mathbf{J}
\end{pmatrix} \begin{pmatrix}
\ddot{\mathbf{q}} \\
\dot{\omega}
\end{pmatrix} - \begin{pmatrix}
0 \\
\omega \times \mathbf{J} \omega
\end{pmatrix}
\] (2.1)

where \(\mathbf{F}\) and \(\mathbf{T}\) are the total force and torque acting on the body center of mass, \(m\) is the mass, \(\begin{pmatrix}
\ddot{\mathbf{q}} \\
\dot{\omega}
\end{pmatrix} = \mathbf{v}\) is the velocity, \(\mathbf{I}\) is the identity matrix, \(\mathbf{J}\) is the principle moment of inertia and \(\omega\) is the angular velocity around the center of mass.

When the bodies are not in contact, the only active physics law is Newton-Euler.

Figure 2.2: A force moves a free body to a new position and changes its velocity.
2.2.2 Rigid body model

In physics, a rigid body is an idealization of a solid body of finite size in which deformation is neglected. In other words, the distance between any two given points of a rigid body remains constant in time regardless of external forces exerted on it. Even though such an object cannot physically exist, many common objects can normally be assumed to be perfectly rigid or as a jointly connected component of many perfectly rigid objects.

The position of a rigid body is the position of all the particles of which it is composed. To simplify the description of this position, we exploit the property that the body is rigid, namely that all its particles maintain the same distance relative to each other. If the body is rigid, it is sufficient to describe the position of at least three non-collinear particles. This makes it possible to reconstruct the positions of all the other particles, provided that their time-invariant position relative to the three selected particles is known.

2.2.3 Friction model

Friction is the force resisting the relative motion of solid surfaces sliding against each other. We will only focus on dry friction here because it is the hard part. We can include viscous friction into the model without changing the form of the equations or make the final equations any harder. Dry friction resists relative lateral motion of two solid surfaces in contact. Dry friction is subdivided into static friction between non-moving surfaces, and kinetic or dynamic friction between moving surfaces.

The properties of dry friction were discovered by experiment in the 15th to 18th centuries and were expressed as three empirical laws:

- Amontons’ First Law: The force of friction is directly proportional to the applied load.
• Amontons’ Second Law: The force of friction is independent of the apparent area of contact.

• Coulomb’s Law of Friction: Kinetic friction is independent of the sliding velocity.

Without going into much details of friction model, which is a field of its own, we now will focus solely on Coulomb’s friction model.

2.2.3.1 Coulomb friction

As stated above, dry friction resists relative lateral motion of two solid surfaces in contact. The two regimes of dry friction are static friction between non-moving surfaces, and kinetic friction (sometimes called sliding friction or dynamic friction) between moving surfaces.

Coulomb friction, named after Charles-Augustin de Coulomb [dC21], is an approximate model used to calculate the force of dry friction. It is governed by the equation:

\[ F_f \leq \mu N \] (2.2)

where:

• \( F_f \) is the force exerted by friction by each surface on the other. It is parallel to the surface, in a direction opposite to the relative movement.

• \( \mu \) is the coefficient of friction, which is an empirical property of the contacting materials.

• \( N \) is the normal force exerted by each surface onto the other, directed perpendicular (normal) to the surface.
Figure 2.3: Coulomb friction: The relation for friction force, normal force, relative velocity and coefficient of friction.

The Coulomb model is an approximation of friction. It mathematically follows from the assumptions that surfaces are in atomically close contact only over a small fraction of their overall area, that this contact area is proportional to the normal force, and that frictional force is proportional to the applied normal force, independently of the contact area (Leonardo Da Vinci’s friction experiments). Such reasoning aside, however, the approximation is fundamentally an empirical construction. It is a rule of thumb describing the approximate outcome of an extremely complicated physical interaction. The strength of the approximation is its simplicity and versatility though in general the relationship between normal force and frictional force is not exactly linear (and so the frictional force is not entirely independent of the contact area of the surfaces), the Coulomb approximation is an adequate representation of friction for the analysis of many physical systems.

When the surfaces are conjoined, Coulomb friction becomes a very poor approximation (for example, adhesive tape resists sliding even when there is no normal force, or a negative normal force). In this case, the frictional force may depend
strongly on the area of contact. Some drag racing tires are adhesive in this way. However, despite the complexity of the fundamental physics behind friction, the relationships are accurate enough to be useful in many applications.

### 2.2.4 Restitution models

A restitution model defines the reactions after two rigid bodies collide. In physics, the most popular model is the Newton hypothesis which relates the incoming and outgoing relative speed between two objects.

\[ \nu_i = -e \nu_o \]  \hspace{1cm} (2.3)

where:

- \( \nu_i \) is the relative incoming speed between two bodies.
- \( e \) is the coefficient of restitution.
- \( \nu_o \) is the relative speed after collision.

Another popular restitution model is Poisson hypothesis where the speed is replaced by impulse.

\[ \rho_i = -e \rho_o \]  \hspace{1cm} (2.4)

with \( \rho_i \) and \( \rho_o \) are compression and restitution impulses between two bodies respectively.

Usually, the coefficient of restitution is a number between 0 (perfectly inelastic collision) and 1 (elastic collision) inclusive. Like friction, coefficient of restitution is purely empirical and can vary due to many factors.

Both Newton and Poisson hypotheses can increase total energy. There is a better alternative, Stronges restitution law, that conserves total energy but is harder
to solve and can have no solution in some cases [Dje08].

Newton hypothesis is the easiest to implement and tends to work well in simple cases but when the collision involves multiple bodies, it may generate unrealistic outcomes. Poisson hypothesis is a little more expensive but works well in many cases. It is even better than Stronge’s model [Dje08] in practice. Note that restitution law can become much more complicated, especially when we take into account Coulomb friction. In this thesis, I will treat simple Poisson hypothesis as the default model, unless otherwise stated. Interested readers can refer to [Dje09a] and [Dje09b] for more discussions on these three restitution models.

2.3 Mathematical models

In previous section, I discuss the physical models of rigid body physics engines. In order to get solutions on computer, the next step is to convert these physical laws to mathematical equations. We will call these equations mathematical models of physical simulation.

2.3.1 Rigid body dynamics background

Before working on the mathematical model, I need to define various mathematical terms related to rigid body.

2.3.1.1 Position

During simulation, we want to keep track of any rigid body. One way to achieve that is to provide the position of any part attached to the body. The rigid body model allows us to only have to keep track a predefined coordinate frame fixed to the body. From the position and orientation of the body-fixed frame, one can easily compute the states of any part attached to it.

An unconstrained rigid body has six degrees of freedom: three translational
Figure 2.4: Position and orientation of a rigid body specified by a body inertia (or fixed) reference frame $A$ with origin at the center of mass of $A$, denoted $A^*$. 

and three rotational. The configuration (i.e. position and orientation) of the body can be represented by an orthonormal reference frame attached to a point on the body (see figure 2.4). For convenience, this frame is usually attached to the center of mass of the body, but this is not a requirement. By convention, we also assume that the frame is right-handed. To globally locate a body, we also need to define a fixed world frame $N$ (see 2.4).

The frame $A$ can be defined as three vector tuple $(\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3)$ where $(\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3)$ is right hand and orthogonal:

- $\|\mathbf{a}_i\| = 1, \forall i = 1, 2, 3$ (orthonormal)
- $\mathbf{a}_i \cdot \mathbf{a}_j = 0, i, j = 1, 2, 3, i \neq j$ (orthonormal)
- $\mathbf{a}_1 \times \mathbf{a}_2 = \mathbf{a}_3$ (right hand rule)

Let $^A\mathbf{R} = [\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3]$ be a $3 \times 3$ matrix. Then $^A\mathbf{R}$ can be the orthonormal basis for 3-dimensional Euclidean space $\mathbb{E}^3$. Or any vector $^A\mathbf{v} \in \mathbb{E}^3$ can be uniquely
represented by another vector $\mathbf{v} \in \mathbb{R}^3$:

$$\mathbf{v} = ^A R^T A \mathbf{v}$$  \hspace{1cm} (2.5)$$

with $^A \mathbf{v} \in \mathbb{R}^3$ is the representation of $\mathbf{v}$ in coordinate frame $A$. Note that if we have another orthonormal frame $B$ then we can also have coordinates of $\mathbf{v}$ in $B$:

$$\mathbf{v} = ^A R^T A \mathbf{v} = ^B R^T B \mathbf{v}$$  \hspace{1cm} (2.6)$$

It follows that,

$$^A \mathbf{v} = ^A R^B \mathbf{v}$$  \hspace{1cm} (2.7)$$

So we can convert the representations between two frames $A$ and $B$ by a $3 \times 3$ rotation matrix $^A R^B = ^A R^B R^T$.

The position of the rigid body is described by the position of the body fixed frame with respect to the global world frame (also known as inertial or Newtonian). The orientation of the rigid body can represented by the rotation matrix associated with the body inertia frame and the fixed world frame. There are, however, many possible ways to parameterize the rotation, and the next section briefly discusses some of them.

2.3.1.2 Orientation

All rotation matrix like the one described previously belong to a group called the special orthogonal group of dimension 3, denoted $SO(3)$:

$$SO(3) = \{ R | R \in \mathbb{R}^{3 \times 3}, R R^T = R^T R = I, \det(R) > 0 \}$$  \hspace{1cm} (2.8)$$

The condition $\det R > 0$ enforces the right hand rule.
There are many other possible representations of $SO(3)$ in addition to rotation matrices, for example Euler angle, Euler parameters, Gibbs-Rodrigues’ representation etc. In this thesis, we will focus on Euler parameters or quaternions because they are compact and do not suffer from singularities.

Quaternions or Euler parameters were developed over a hundred years ago by William Hamilton. A quaternion $q = (x, y, z, w)$ is an extended complex value:

$$q = w + ix + jy + kz \quad (2.9)$$

where $x, y, z, w$ are real values and the following equations hold for $i, j, k$:

$$i^2 = j^2 = k^2 = -1 \quad ij = k = -ji$$

Here, another interpretation can be used: A quaternion consists of one real scalar $w$ and an imaginary vector $v$. It is typically written as a 4-dimensional vector:

$$q = \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix} = \begin{bmatrix} w \\ v \end{bmatrix} \quad (2.10)$$

Quaternions can be normalized to become unit quaternions. Unit quaternions have a magnitude of 1:

$$|q| = \sqrt{w^2 + x^2 + y^2 + z^2} = 1 \quad (2.11)$$

Conveniently, A quaternion describes a rotation by an angle $\alpha$ about an axis given by the unit vector $u$ is:

$$q = \begin{bmatrix} \cos \frac{\alpha}{2} \\ \sin \frac{\alpha}{2} u \end{bmatrix} \quad (2.12)$$
2.3.1.3 Velocity

The first order time derivative of the body’s motion is the velocity. We can separate the body’s configuration into position and orientation and define two velocity components respectively: linear $\nu = [v_x \ v_y \ v_z]^T$ and angular velocity $\omega = [w_x \ w_y \ w_z]^T$. The linear velocity is the time derivative of the position vector

$$\nu = \frac{d\mathbf{r}}{dt}$$

(2.13)

where $\mathbf{r} = \frac{d\mathbf{r}}{dt}$. The vector $\nu$ is the velocity at the origin of the moving frame attached to the body’s center of gravity.

Similarly, the angular velocity is the time rate of change of the body’s orientation. However, the notion of orientation vector does not exist so the angular velocity is not the time derivative of the orientation. The angular velocity of a reference frame $\{B\} = \{\text{vect}b_1, \text{vect}b_2, \text{vect}b_3\}$ in a reference frame $\{A\} = \{\text{vect}a_1, \text{vect}a_2, \text{vect}a_3\}$, denoted $A\omega^B$, is defined as:

$$A\omega^B \equiv b_1 \frac{d}{dt} b_2 \cdot b_3 + b_2 \frac{d}{dt} b_3 \cdot b_1 + b_3 \frac{d}{dt} b_1 \cdot b_2$$

(2.14)

From (2.13) and (2.14), we can obtain the rate of change of any fixed vector attached to the body. Given any $B\mathbf{v} = [v_x \ v_y \ v_z]^T$ as a vector in body B’s frame, then

$$\frac{A\mathbf{d}v}{dt} = A\omega^B \times B\mathbf{v} = \begin{bmatrix} 0 & \omega_z & -\omega_y \\ -\omega_z & 0 & \omega_x \\ \omega_y & -\omega_x & 0 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}$$

(2.15)

Physically, given the angular velocity $\omega$, the magnitude of $\omega$ is the rotation rate and the direction is the rotation axis.

Given a linear velocity, the new position can be determined by integration.
However, the orientation can follow many possible parameterizations. In general, if \( \mathbf{q} \in \mathbb{R}^k \) is a configuration of the body, for each parameterizations, we need to find the representation Jacobian \( \mathbf{G}(\mathbf{q}) \in \mathbb{R}^{k \times 3} \) such that

\[
\frac{d\mathbf{q}}{dt} = \mathbf{G}(\mathbf{q})\mathbf{\omega}
\]

(2.16)

If \( \mathbf{G} \) is not invertible, the \( \mathbf{q} \) cannot be solved for given \( \mathbf{\omega} \). So we normally pick the orientation parametrization that has invertible \( \mathbf{G} \). In this thesis, we will mostly work with Euler parameters or quaternion. So, the orientation vector \( \mathbf{q} = [e_w \ e_x \ e_y \ e_z] \), and the Jacobian is given by [Cho92, Spr86]

\[
\mathbf{G}(\mathbf{q}) = \frac{1}{2} \begin{bmatrix}
-e_x & -e_y & -e_z \\
e_w & e_z & -e_y \\
-e_z & e_w & e_x \\
e_y & -e_x & e_w
\end{bmatrix}
\]

(2.17)

In practice, we don’t treat linear and angular velocity separately. By stacking them in one vector, we arrives at the notion of generalized velocity of commonly referred to as the *twist* of the body.

\[
\mathbf{\nu} = \begin{bmatrix}
\mathbf{v} \\
\mathbf{\omega}
\end{bmatrix}
\]

(2.18)

Using homogeneous coordinates and twists allow us to compactly represent the motion of rigid bodies. For example, the velocity of a point \( P \) that has displacement vector from the body origin \( \mathbf{B}^p \), can be written as

\[
A^p = A_B \begin{bmatrix}
\mathbf{B}^p \\
\mathbf{1}
\end{bmatrix}
\]

(2.19)
where $^A_BH \in \mathbb{R}^{4 \times 4}$ is the matrix

$$^A_BH = \begin{bmatrix}
^A\omega^B & ^A\nu^B \\
0 & 0
\end{bmatrix}$$

(2.20)

with $^A\omega^B$ is the skew-symmetric operator defined based on the angular velocity of the body $\omega$ as

$$^A\omega^B = \begin{bmatrix}
0 & \omega_z & -\omega_y \\
-\omega_z & 0 & \omega_x \\
\omega_y & -\omega_x & 0
\end{bmatrix}$$

(2.21)

2.3.1.4 Torque and force

![System of external forces acting on a body.](image)

Figure 2.5: System of external forces acting on a body.

In Newtonian world, bodies are moved by forces and moments. Normally, the system of forces and moments applied to a body can be reduced to a single resultant force applied to the body’s center of mass and single resultant moment. For example, in Figure 2.5 the body was being acted upon by $f$ forces $F_1, F_2, \cdots, F_f$ and
moments $M_1, M_2, \ldots, M_m$. The resultant force $F$ and moment $M$ are given by:

$$
F = \sum_{i=1}^{m} F_i \\
M = \sum_{i=1}^{m} M_i + \sum_{i=1}^{f} r_i \times F_i
$$

(2.22)

where $r_i$ is the vector from the body center of mass to the acting point of $F_i$.

For convenience, we usually group both force and moment together in a single resultant vector of force and moment. The final term is usually referred to as the wrench applied.

$$
\lambda = \begin{bmatrix}
F \\
M
\end{bmatrix}
$$

(2.23)

It is important to note that the wrenches are always associated with a certain frame on the body, the force $F$ is applied to the origin and the moment $M$.

We can transform generalized velocity and force between frames $\{A\}$ and $\{B\}$ using the adjoint matrix $[\text{Mas01}]$.

$$
\frac{A}{B}\text{Ad} = \begin{bmatrix}
\frac{A}{B}R & \frac{A}{B}R \\
0 & [\frac{A}{B}\dot{r}_O] \frac{A}{B}R
\end{bmatrix}
$$

(2.24)

The generalized velocity $^B\nu$ and $^A\nu$ represented in frames $\{B\}$ and $\{A\}$, respectively, are related by:

$$
^A\nu = \frac{A}{B}\text{Ad}^B\nu
$$

(2.25)

The transpose of the adjoint transforms the wrench as in $[\text{Mas01}]$ is:

$$
^A\lambda = \frac{A}{B}\text{Ad}^{TB}\lambda
$$

(2.26)
2.3.2 Complementarity problem

Now that we have introduced the rigid body background, we can start translating the physical models described in 2.2 to mathematical equations. But first, we need to introduce the complementarity problem, the main mathematical tool for rigid body contact and friction.

As we will show in 2.3.3, a natural way to mathematically model the dynamics of a rigid body system with intermittent contact and friction is as a differential complementarity problem (DCP) [FP03a, FP03b]. In this section, we provide basic definitions of various forms of complementarity problems.

Differential Complementarity Problem (DCP) Let \( g(u, v) : \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \mapsto \mathbb{R}^{n_1} \) and \( f(u, v) : \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \mapsto \mathbb{R}^{n_2} \) be given vector function of \( u \in \mathbb{R}^{n_1} \) and \( v \in \mathbb{R}^{n_2} \), with \( n_1 + n_2 = n \). Find \( u \) and \( v \) such that:

\[
\begin{align*}
\dot{u} &= g(u, v), \text{ } u \text{ is free} \\
0 &\leq v \perp f(u, v) \geq 0
\end{align*}
\]

Usually, we cannot solve general DCP directly so we have to solve for an approximation instead.

Mixed Complementarity Problem (MCP) Given a continuously differentiable function \( F : \mathbb{R}^n \mapsto \mathbb{R}^n \), and lower and upper bounds

\[
\begin{align*}
l &\in \{\mathbb{R} \cup \{-\inf\}\}^n \\
u &\in \{\mathbb{R} \cup \{+\inf\}\}^n
\end{align*}
\]

with \([l, u]\), the Cartesian product of the closed (possibly unbounded) intervals \([l_i, u_i]\), nonempty. Then the mixed complementarity problem [FM96, FKM99], denoted \( MCP(F, l, u) \), is to find a vector \( z \in \mathbb{R}^n \) such that precisely one of the following
holds for each $i \in \{1, \cdots, n\}$:

$$F_i(z) = 0 \text{ and } l_i \leq z_i \leq u_i$$
$$F_i(z) > 0 \text{ and } z_i = l_i$$
$$F_i(z) < 0 \text{ and } z_i = u_i$$

Mixed Linear Complementarity Problem (MLCP) Given a squared matrix $M \in \mathbb{R}^{n \times n}$, a vector $q \in \mathbb{R}^n$, and lower and upper bounds

$$l \in \{\mathbb{R} \cup \{-\text{inf}\}\}^n$$
$$u \in \{\mathbb{R} \cup \{+\text{inf}\}\}^n$$

with $[l, u]$, the Cartesian product of the closed (possibly unbounded) intervals $[l_i, u_i]$, nonempty. Then the mixed complementarity problem [FM96, FKM99], denotes $MLCP(M, q, l, u)$, is to find a vector $z \in \mathbb{R}^n$ such that precisely one of the following holds for each $i \in \{1, \cdots, n\}$:

$$M_i \cdot z + q_i = 0 \text{ and } l_i \leq z_i \leq u_i$$
$$M_i \cdot z + q_i > 0 \text{ and } z_i = l_i$$
$$M_i \cdot z + q_i < 0 \text{ and } z_i = u_i$$

where $M_i \in \mathbb{R}^n$ is the $i$-th column of $M$.

In general, complementarity problems are NP-hard (A simple $MLCP$ was proved to be NP-hard in [JFR02]). But experiments show that, most of the time, we can find solutions for the physical simulation mixed linear complementarity problems. Many times, we can convert the problem into a monotone complementarity problem, which exposes a polynomial running time in theory and can be solved very fast in practice. But then the approximation may introduce model errors that are not acceptable in some applications. We will discuss more at length on this topic in
2.3.3 Complementarity formulation of physical simulation

Section 2.2 presents a list of physical models that the simulation follows. In this part, we will show how to translate those physics laws into mathematical equations.

The Newton-Euler laws and kinematic update form a system of ordinary differential equation. The contact and rigid body conditions can be written mathematically as a system of complementarity problems. The joint constraints constitute a system of algebraic equations. Putting all of them together, the dynamic model is a differential complementarity problem (DCP) as defined in definition 2.3.2. The rest of this section will discuss the steps to formulate the mathematical models of physical simulation.

To describe the dynamic model, we will need to introduce some system-wide notations. Let $q_j$ be the position and orientation of a frame attached to the center of gravity of body $j$-th in an inertial frame and $\nu_j$ be the generalized velocity of the body’s frame. The system generalized coordinates, $q$, and generalized velocity, $\nu$, are formed by concatenating $q_j$ and $\nu_j$.

2.3.3.1 Newton-Euler equation

The Newton-Euler equations (2.1) can be written in matrix form as:

$$M(q, t)\dot{\nu} = \lambda_{vp}(q, \dot{q}, t) + \lambda_{app}(q, t)$$  \hspace{1cm} (2.27)
where
\[
M = \begin{bmatrix}
  m_1 I & 0 & 0 & 0 \\
  0 & I_1 & 0 & 0 \\
  \ddots & & & \\
  0 & 0 & m_n I & 0 \\
  0 & 0 & 0 & I_n
\end{bmatrix}
\]

We also separate the applied force and torque into two parts, those that are dependent only on configuration, \( \lambda_{\text{app}} \), and those that also depend on velocity, \( \lambda_{\text{vp}} \). For example, gravity force would be a part of \( \lambda_{\text{app}} \) while the velocity product term in the Euler equation \( (\omega \times I \omega) \) would appear in \( \lambda_{\text{vp}} \).

The kinematic update law (2.16) can be written compactly for the whole system as:
\[
\dot{q} = G(q) \nu
\]  
(2.28)

where
\[
G(q) = \begin{bmatrix} G(q_1) & 0 & \cdots & 0 \\ \cdots & \ddots & \cdots & \cdots \\ 0 & \cdots & G(q_n) \end{bmatrix},
\]
and \( J(q_i) \in \mathbb{R}^{4 \times 3} \) was defined in equation (2.17) and relates the rate of change of body’s Euler parameters to its angular velocity.

### 2.3.3.2 Bilateral (joint) constraints

Equations (2.27) and (2.17) together are enough to simulate unconstrained system of rigid bodies. Joint constraints (equality constraints) like the ones in figures 2.6 are important in robotics and mechanical systems. Mathematically, they are modeled with algebraic equations:
\[
\Phi_i(q, t) = 0
\]  
(2.29)
where $\Phi_i$ is the constraint function of joint $i$ and $q$ is the vector of all bodies' configurations concatenated together.

At velocity level, the joint constraint looks like:

$$\frac{d\Phi_i}{dt} = \frac{\partial \Phi_i(q, t)}{\partial q} \dot{q} + \frac{\partial \Phi_i(q, t)}{\partial q} \dot{t} = 0 \quad (2.30)$$

Using equation (2.28) to substitute $\dot{q}$ with $G(q)\nu$, the constraint (2.30) becomes:

$$0 = W_i \nu + \frac{\partial \Phi_i(q, t)}{\partial q} \dot{t} \quad (2.31)$$

where $W_i \nu = \frac{\partial \Phi_i(q, t)}{\partial q} G(q)$ is the constraint Jacobian matrix (wrench matrix). The constraint Jacobian $\frac{\partial \Phi_i(q, t)}{\partial q} G(q)$ value depends on the type of the joint.

The physical rigid body 2.2.2 and friction models 2.2.3 are usually enforced mathematically through the contact constraint model. Basically, the contact constraint acts between two bodies to prevent them from inter-penetrating and apply a friction law at the point of contact. Depending on the effect, the contact constraint model can be separated into two parts: non-penetration and friction constraint.
2.3.3.3 Non-penetration constraint

The main role of non-penetration constraint is to prevent the spaces occupied by the bodies from overlapping. Thus, it mimics the rigidity of those bodies. Mathematically, non-penetration constraints are difficult to deal with. The main reason is that the impact between the bodies is non-smooth. During the impact phase, the change in a body velocity has to be instantaneous. Thus, we cannot simply integrate a body velocity to get the contact force during impact. The other difficulty comes from the intermittent nature of contacts. They appear when the bodies touch, and disappear immediately when they separate. That basically means the non-penetration constraint prevents penetration but has to allow separation.

\[ \hat{\phi}_{in}(q,t) > 0: \text{separation} \]
\[ \hat{\phi}_{in}(q,t) = 0: \text{touching} \]
\[ \hat{\phi}_{in}(q,t) < 0: \text{penetration} \]
Because we want to prevent penetration, the constraint for contact $i$ will be:

$$
\phi_{in}(q, t) \geq 0. \quad (2.32)
$$

To maintain such constraint, we rely on the normal contact force that act along the direction of $n_i$ with magnitude $\lambda_{in}$. Physically, the contact force is compressive, meaning it cannot pull bodies together. Thus,

$$
\lambda_{in} \geq 0. \quad (2.33)
$$

To model intermittency at contact $i$, we only allow at most one of the contact distance $\phi_{in}(q, t)$ or contact force $\lambda_{in}$ to be positive at the same time, or equivalently:

$$
\phi_{in}(q, t)\lambda_{in} = 0. \quad (2.34)
$$

When the system contains $n_c$ contacts, we can stack them together to form the system of non-penetration constraints in vector form as follow:

$$
\Phi_n(q, t) \geq 0 \quad (2.35)
$$

$$
\lambda_n \geq 0 \quad (2.36)
$$

$$
\Phi_n(q, t) \odot \lambda_n = 0 \quad (2.37)
$$

where $\odot$ denotes the entry-wise Hadamard product \cite{RAH85}.

In conjunction with equations (2.35) and (2.36), (2.37) can be simplified to

$$
\Phi_n(q, t)^T \cdot \lambda_n = 0. \quad (2.38)
$$

Finally, equations (2.35), (2.36) and (2.38) represent the non-penetration con-
straint of the system and can be written in complementarity form as:

\[
0 \leq \Phi_n(q,t) \perp \lambda_n \geq 0. \tag{2.39}
\]

with \(\perp\) is an operator defined as: Given two vectors \(a, b \in \mathbb{R}^n\), \(0 \leq a \perp b \geq 0 \iff a \geq 0, b \geq 0, a^Tb = 0\).

### 2.3.3.4 Friction constraint

The physical Coulomb model presented in 2.2.3 can be written mathematically using the maximum dissipation principle. It states that given a normal force magnitude and relative slipping velocity at the contact, the friction constraint force is the one that maximizes the rate of energy dissipation. Or specifically

\[
F_i(\mu_i, \lambda_{in}) = (\lambda_{it}, \lambda_{io}) : \mu_i^2 \lambda_{in}^2 - \lambda_{it}^2 - \lambda_{io}^2 \geq 0. \tag{2.40}
\]

where \(F_i(\mu_i, \lambda_{in})\) is the friction cone, \(\mu_i\) is the coefficient of friction, \(\lambda_{it}\) and \(\lambda_{io}\) are friction forces along the two orthogonal tangential directions at contact \(i\). Define orthogonal sliding velocity components \(v_{it} = W_t^T \nu + \frac{\partial \phi_i}{\partial t}\) and \(v_{io} = W_o^T \nu + \frac{\partial \phi_o}{\partial t}\). Then using the maximum dissipation principle, Coulomb’s law at contact \(i\) can be written as:

\[
(\lambda_{it}, \lambda_{io}) \in \arg \max_{(\lambda_{it}, \lambda_{io}) \in F_i(\mu_i, \lambda_{in})} (-\lambda_{it}v_{it} - \lambda_{io}v_{io}) \tag{2.41}
\]

The Karush-Kuhn-Tucker (KKT) necessary conditions [Rus06, MSB93] for optimization problem (2.41) cannot be used because with \(\lambda_{in} = 0\) or \(\mu_i = 0\), there are no applicable regularity conditions. Thus, we have to use lesser restricted Fritz-John condition [Rus06, MSB93] as already mentioned in [TPSL97, STKP04]. The Fritz-John optimality conditions that the friction forces at contact \(i\) must satisfy are:
where $u_{i0}$ and $u_i$ are Lagrange multipliers.

The extra multiplier $u_{i0}$ can be proved to be equal to $\mu_i \lambda_{in}$. Because $\mu_i \lambda_{in} \geq 0$, a non-zero $u_i$ will satisfy equation (2.45). So we can effectively remove (2.45).

The proof involves a number of algebraic steps. First, if $u_i = 0$ then $u_{i0} > 0$. If $u_i > 0$, we can solve for $\lambda_{it}$ and $\lambda_{io}$ from equations (2.42) and (2.43):

$$\lambda_{it} = \frac{u_{i0} v_{it}}{2u_i}$$
$$\lambda_{io} = \frac{u_{i0} v_{io}}{2u_i}$$

Also, from complementarity condition (2.44) we know that its right hand side has to be equal to 0. Or:

$$\mu_i^2 \lambda_{in}^2 = \left( \frac{u_{i0} v_{it}}{2u_i} \right)^2 - \left( \frac{u_{i0} v_{io}}{2u_i} \right)^2.$$  

From there, solve for $u_{i0}$ we have:

$$u_{i0} = \frac{2u_i \mu |i\lambda_{in}|}{\sqrt{v_{it}^2 + v_{io}^2}}$$
Substitute $u_{i0}$ into equations (2.42) and (2.43) we arrive at:

$$0 = \frac{2u_i \mu_i \lambda_{in}}{\sqrt{v_{it}^2 + v_{io}^2}} v_{it} + 2u_i \lambda_{it}. \quad (2.50)$$

$$0 = \frac{2u_i \mu_i \lambda_{in}}{\sqrt{v_{it}^2 + v_{io}^2}} v_{io} + 2u_i \lambda_{io}. \quad (2.51)$$

which can be further simplified to:

$$0 = \mu_i \lambda_{in} v_{it} + \sigma_i \lambda_{it}. \quad (2.52)$$

$$0 = \mu_i \lambda_{in} v_{io} + \sigma_i \lambda_{io}. \quad (2.53)$$

with $\sigma_i = \sqrt{v_{it}^2 + v_{io}^2}$, the relative slip speed at contact $i$. Due to algebraic relations, we can replace $u_i$ with $\sigma_i$ in equation (2.44) because:

- $u_i > 0$, the right hand side of (2.44) which is $\mu_i^2 \lambda_{in}^2 - \lambda_{it}^2 - \lambda_{io}^2 = 0$. So $\lambda_{it}^2 - \lambda_{io}^2 = \mu_i^2 \lambda_{in}^2 > 0$. Along with (2.42) and (2.43), we can see that $v_{it}$ and $v_{io}$ cannot both equal to 0. Thus, $\sigma_i = \sqrt{v_{it}^2 + v_{io}^2} > 0$.

- $u_i < 0$, from (2.42) and (2.43), we have both $v_{it} = v_{io} = 0$. So $\sigma_i = \sqrt{v_{it}^2 + v_{io}^2} = 0$

Finally, we arrive at the commonly seen formulation:

$$0 = \mu_i \lambda_{in} v_{it} + \sigma_i \lambda_{it} \quad (2.54)$$

$$0 = \mu_i \lambda_{in} v_{io} + \sigma_i \lambda_{io} \quad (2.55)$$

$$0 \leq \sigma_i \perp \mu_i^2 \lambda_{in}^2 - \lambda_{it}^2 - \lambda_{io}^2 \geq 0 \quad (2.56)$$
Or compactly, for the whole system as:

\[
0 = (U\lambda_n) \circ (v_t) + \lambda_t \circ \lambda \\
0 = (U\lambda_n) \circ (v_o) + \lambda_o \circ \lambda \\
0 \leq \sigma \perp (U\lambda_n) \circ (U\lambda_n) - \lambda_t \circ \lambda_t - \lambda_o \circ \lambda_o \geq 0
\]

where \( U \) is the diagonal matrix with \( i^{th} \) diagonal element equal to \( \mu_i, \lambda_n, \lambda_t, \lambda_o \) and \( \sigma \) are the concatenations of \( \lambda_{ii}, \lambda_{it}, \lambda_{io} \) and \( \sigma_i \) respectively for all the contacts in the system.

Equations (2.57)-(2.59) are nonlinear in the unknowns. We can devise a linear version by approximating the quadratic Coulomb friction cone. Let \( n_d \) friction direction vectors \( d_j \) be chosen such that they positively span the space of all possible friction forces as in figure 2.8, and let \( (\lambda_{if})_j \) be the friction force components in those directions. Also, let \( (\psi_{if}(q, t))_j \) be the corresponding local tangential displacement function.

The friction force can be approximated as

\[
F_i(\mu_i, \lambda_{in}) = \{\lambda_{if} \mid \mu_i\lambda_{in} - e^T\lambda_{if} \geq 0\}
\]
where $\mathbf{e} \in \mathbb{R}^{n_d}$ is vector of ones,

$$
W_f(q, t) = \begin{bmatrix}
\cdots & W_{if} & \cdots
\end{bmatrix} \in \mathbb{R}^{n_c \times n_d}
$$

$$
W_{if} = \begin{bmatrix}
\mathbf{d}_1 & \cdots & \mathbf{d}_{n_d} \\
\mathbf{r}_i \times \mathbf{d}_1 & \cdots & \mathbf{r}_i \times \mathbf{d}_{n_d}
\end{bmatrix}
$$

$r_i$ is the vector from body’s center of mass to contact $i$ position. The approximate dissipation condition becomes:

$$
\lambda_{if} \in \arg \max_{\lambda \in \mathcal{F}_i} \left( -\lambda_{if}^T W_{if}^T \nu \right) \quad (2.61)
$$

From there, the equivalent LCP formulation of the maximum dissipation condition for the approximate friction cone can be written as:

$$
0 \leq (\lambda_{if})_j \perp (v_{if})_j + \sigma_i \geq 0, \quad j = 1 \ldots n_d \\
0 \leq \sigma_i \perp \mu_i \lambda_{in} - \mathbf{e}^T \lambda_{if} \geq 0 \quad (2.62)
$$

Or compactly written for all contacts as:

$$
0 \leq \mathbf{\lambda}_i \perp W_i^T \nu + \mathbf{E} \sigma + \frac{\partial \Psi_i}{\partial t} \geq 0 \quad (2.64)
$$

$$
0 \leq \mathbf{\sigma} \perp U\lambda_n - \mathbf{E}^T \lambda_f \geq 0 \quad (2.65)
$$

where $\mathbf{E}$ is now a block diagonal matrix with $i^{th}$ block on the main diagonal given by $\mathbf{e}$. 
2.3.4 Instantaneous formulations of constrained dynamics

If the quadratic form of the friction mathematical model is used, we arrive at a nonlinear DCP formulation. If instead the linearized version of the law is used, we end up with a linear DCP formulation.

2.3.4.1 Nonlinear DCP formulation

The nonlinear DCP formulation is as follow:

\[
\begin{align*}
M(q)\nu &= W_n(q)\lambda_n + W_t(q)\lambda_t + W_o(q)\lambda_o + \lambda_{app}(q,t) + \lambda_{vp}(q,\nu,t) \\
\dot{q} &= G(q)\nu \\
0 &= \Phi(q,t) \\
0 &= (U\lambda_n) \circ (v_t) + \lambda_t \circ \lambda \\
0 &= (U\lambda_n) \circ (v_o) + \lambda_o \circ \lambda \\
0 &\leq \lambda_n \perp \psi_{in}(q,t) \geq 0 \\
0 &\leq \sigma \perp (U\lambda_n) \circ (U\lambda_n) - \lambda_t \circ \lambda_t - \lambda_o \circ \lambda_o \geq 0
\end{align*}
\]

(2.66)

2.3.4.2 Linear DCP formulation

\[
\begin{align*}
M(q)\nu &= W_n(q)\lambda_n + W_t(q)\lambda_t + W_o(q)\lambda_o + \lambda_{app}(q,t) + \lambda_{vp}(q,\nu,t) \\
\dot{q} &= G(q)\nu \\
0 &= W_n^T\nu + \frac{\partial \psi_{in}}{\partial t} + \frac{\partial \psi_{in}}{\partial q} \frac{\Delta q}{\partial t} \\
0 &\leq \lambda_n \perp \psi_{in}(q,t) \geq 0 \\
0 &\leq \lambda_t \perp W_t^T\nu + E\sigma + \frac{\partial \psi_{if}}{\partial t} \geq 0 \\
0 &\leq \sigma \perp (U\lambda_n) - E^T\lambda_t \geq 0
\end{align*}
\]

(2.67)
2.3.4.3 Comparison with physical models

We have developed mathematical models from the physical ones described in section 2.2. It is one step further from the abstraction so there might be errors introduced by approximation. Let look at them carefully.

For the law of motion 2.2.1, both of our mathematical models follow the law exactly. So there is no error introduced here.

The rigid body physical model 2.2.2 is approximated in the mathematical model by a system of contact points. There was no notion of contact points in the rigid body physical model but in order to translate the law to mathematical language, we have to introduce them. In practice, we collect contact points from collision detection module, a completely new component in physical simulation. Even though the contact model makes the mathematical model of rigidity simple, implementing the right collision detection is very challenging and could introduce errors to the system. Basically, a contact constraint will enforce the projection of the two contact points in two bodies not to overlap along its normal direction. There are two major problems with it. First, if the collision detection misses a contact point then we will see penetration. Second, we can only prevent penetration in one direction along the normal. Nevertheless, the point contact model is still popular due to its simplicity.

For the Coulomb friction model, it is easy to observe that the nonlinear DCP formulation models the rules exactly while the linear version introduces errors. A more in depth comparison between nonlinear and approximate friction model can be found in [BNT09] and [BNAT10].
2.4 Computer models of physical simulation

We have developed a set of mathematical equations that provide a good approximation of our system physical models. Solving these equations give us the states of the system of rigid bodies at anytime. But both the nonlinear and linear DCP are unsolvable. So again, we need to devise an approximation model that can be solved by computer.

The main reason we cannot solve DCP formulation directly is because the combination of differential equation and non-smooth reaction forces in the variables. To overcome that, we can use a technique called time-stepping that treat the continuous time space as a series of discrete time events with distance equal to time step $h$. When $h \to 0$, then the series converges to the solution of the continuous model.

Let $t_\ell$ denotes the current time and $h$ be the time step or step size. Use the superscripts $\ell$ and $\ell + 1$ to denote quantities at the beginning and the end of the $\ell$th time step respectively. Also, let $\dot{\nu} \approx \frac{\nu^{\ell+1} - \nu^\ell}{h}$ and $\dot{q} \approx \frac{q^{\ell+1} - q^\ell}{h}$, we get the following nonlinear and linear discrete time systems that can be solved by computer.

The results of applying time-stepping technique to equations (2.66) and (2.67) are first described in a paper by David Stewart and J.C Trinkle [ST96]. Because of that, we usually refer to the them as Stewart-Trinkle methods.
2.4.1 Mixed nonlinear complementarity formulation

\[ M \nu^{\ell+1} = M \nu^\ell + h(W_n \lambda_n^{\ell+1} + W_t \lambda_t^{\ell+1} + W_o \lambda_o^{\ell+1}) + \lambda_\text{app}^\ell + \lambda_\text{vp}^\ell \]

\[ q^{\ell+1} = q^\ell + hG \nu^{\ell+1} \]

\[ 0 \leq \lambda_n^{\ell+1} \perp \psi_n(q^{\ell+1}) \geq 0 \]

\[ 0 = (U \lambda_n^{\ell+1}) \circ (v_t^{\ell+1}) + \lambda_t^{\ell+1} \circ \sigma^{\ell+1} \]

\[ 0 = (U \lambda_n^{\ell+1}) \circ (v_o^{\ell+1}) + \lambda_o^{\ell+1} \circ \sigma^{\ell+1} \]

\[ 0 \leq \sigma^{\ell+1} \perp (U \lambda_n^{\ell+1}) \circ (U \lambda_n^{\ell+1}) - \lambda_t^{\ell+1} \circ \lambda_t^{\ell+1} - \lambda_o^{\ell+1} \circ \lambda_o^{\ell+1} \geq 0 \]

2.4.2 Mixed linear complementarity formulation

\[ M \nu^{\ell+1} = M \nu^\ell + h(W_n \lambda_n^{\ell+1} + W_t \lambda_t^{\ell+1} + \lambda_\text{app}^\ell + \lambda_\text{vp}^\ell) \]

\[ q^{\ell+1} = q^\ell + hG \nu^{\ell+1} \]

\[ 0 \leq \lambda_n^{\ell+1} \perp \psi_n(q^{\ell+1}) \geq 0 \]

\[ 0 \leq \lambda_t^{\ell+1} \perp v_t^{\ell+1} + E \sigma^{\ell+1} \geq 0 \]

\[ 0 \leq \sigma^{\ell+1} \perp U \lambda_n^{\ell+1} - E^T \lambda_t^{\ell+1} \geq 0 \]

where \( \psi_n(q^{\ell+1}, t_{\ell+1}) \approx \psi_n(q^\ell, t_{\ell+1}) + hW_n^T \nu^{\ell+1} \).
Or in matrix form:

\[
\begin{bmatrix}
0 \\
0 \\
\rho_n^{\ell+1} \\
\rho_f^{\ell+1} \\
s^{\ell+1}
\end{bmatrix}
= \begin{bmatrix}
-M & W_b & W_n & W_f & 0 \\
W_b^T & 0 & 0 & 0 & 0 \\
W_n^T & 0 & 0 & 0 & 0 \\
W_f^T & 0 & 0 & 0 & E \\
0 & 0 & U & -E^T & 0
\end{bmatrix}
\begin{bmatrix}
\nu^{\ell+1} \\
\phi_b^{\ell+1} \\
\phi_n^{\ell+1} \\
\phi_f^{\ell+1} \\
\sigma^{\ell+1}
\end{bmatrix}
+ \begin{bmatrix}
M\nu^{\ell} + p_{\text{app}} + p_{\text{vp}} \\
\frac{\psi_b^{\ell}}{h} + \frac{\partial \psi_b^{\ell}}{\partial t} \\
\frac{\psi_n^{\ell}}{h} + \frac{\partial \psi_n^{\ell}}{\partial t} \\
\frac{\partial \psi_f^{\ell}}{\partial t} \\
0
\end{bmatrix}
\] (2.75)

\[
0 \leq \begin{bmatrix}
\rho_n^{\ell+1} \\
\rho_f^{\ell+1} \\
s^{\ell+1}
\end{bmatrix} \perp \begin{bmatrix}
p_n^{\ell+1} \\
p_f^{\ell+1} \\
\sigma^{\ell+1}
\end{bmatrix} \geq 0
\] (2.76)

where \( p_{(\cdot)} = h\lambda_{(\cdot)} \) and \( \rho_{(\cdot)}^{\ell+1} = \frac{\psi_{(\cdot)}^{\ell+1}}{h} \).

2.4.3 Solution methods for mixed complementarity problems

With time-stepping technique, our linear mathematical models can be solved by computer. David Stewart in [ST96] proved that the matrix form of the formulation is co-positive so using Lemke [Mur88, CPS92] method can find the solution in finite time. Another important result that solidify the use of time-stepping method in physical simulation is that when the time step \( h \to 0 \) then the time-stepping formulation converges to the DCP formulation. The result was also proved by David Stewart in [Ste03] and later by Bogdan Gavrea et.al [GAP08].
CHAPTER 3

Historical review of physical simulation methods and libraries

3.1 Penalty and event detection methods

3.1.1 Penalty methods

The machine in figure 3.1 shakes parts into recesses so that they can be picked by the manipulators. This is the motivational example to choose penalty method over constrained-based before Stewart-Trinkle [ST96] and Anitescu-Potra [AP97] methods. This example suits penalty methods is because it requires modeling of extremely transient contact between parts and feeders. Impulse based simulation originates from the works of James Hahn [Hah88], [Hah89] and later by Brian Mirtich [Mir96]. They were later used extensively for rigid body simulation in [MW88], [RH91] and for [MZ90] graphics animation in [WGW90].

Figure 3.1: A vibrational part feeder that shakes parts into recesses to be picked up by a manipulator. The system exhibits very transient contact modes (image taken from Brian Mirtich thesis [Mir96]).
Impulse-based methods proposed a simple way to maintain non-penetration during simulation. For each contact in penetration, the methods apply a normal force with magnitude proportion to the penetration depth. That is the reason they were also called penalty methods. With the methods, we do not have to maintain contact constraints information. The methods work in the same way as placing a linear spring between two contact points of the bodies in penetration. They are simple, fast and very easy to implement given penetration depths information. For that reason, many early physics engines implement penalty methods. At the top level, impulse-based simulation methods are comprises of three simple steps which are listed in Brian Mirtich’s thesis [Mir96]:

- **Collision detection.** Determine the maximum time interval over which the physical system may be integrated such that no collision between two bodies will be missed. At the end of the interval a pair of bodies called the critical pair must be checked for collision.

- **Forward dynamics.** Update the system forward in time by integrating the equations of motion of all bodies.

- **Collision response.** Resolve any pair of existing collisions by applying appropriate impulses.

It works reasonably well in a simple scene. One famous example is Tomohide Kano’s rigid body demo in 2000. However, the methods suffer from many problems, such as it is difficult to maintain a stable resting contact, penetration is unavoidable and penetration depth calculation can be very costly compares to distance computation. With the development of recent complementarity constraint based formulations, there are not many reasons for us to consider using these penalty methods.
3.1.2 Event detection method for contact handling

One big problem with penalty methods is that very large spring constants are needed to keep the penetration sufficiently small. Consequently, these springs may generate large forces leading to stiff equations that are difficult to integrate over. Furthermore, collision detection is done at discrete times rather than continuous intervals. As a result, collision only gets resolved after a significant amount of penetration.

At the time, the only alternative to the penalty methods are constraint-based methods. After the collision detection algorithms report a list of contacts, constrained-based methods create a constraints system to prevent any further penetration and maintain steady contacts. It is important for these types of methods to find the time of impact for all contacts. During the time interval between two changes, we can treat the system as a constrained dynamical system and use classical mechanics theories to solve for contact and friction forces \cite{KTR85}. The method works well for simple systems with a small number of rigid bodies where the number of events is not too large. It may require exponential number, with respect to the number of bodies, of checks to find a consistent state. That renders the methods unsuitable for simulating scenes with a large number of bodies interacting with each other.

3.2 Gaming and computer graphics physics engines

This section discusses the methods behind many current popular physics engines. Even some of the engines listed here are designed with different objectives, they still share the same underlying principles.
<table>
<thead>
<tr>
<th>Engine</th>
<th>Website</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zlib license</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bullet</td>
<td><a href="http://www.bulletphysics.com">www.bulletphysics.com</a></td>
<td>Erwin Coumans(SCEI)</td>
</tr>
<tr>
<td>Box2D</td>
<td>box2d.org</td>
<td>Erin Catto(Blizzard)</td>
</tr>
<tr>
<td>IBDS</td>
<td><a href="http://www.impulse-based.de">www.impulse-based.de</a></td>
<td>Jan Bender</td>
</tr>
<tr>
<td>OpenTissue</td>
<td><a href="http://www.opentissue.org">www.opentissue.org</a></td>
<td>Kenny Erleben et.al</td>
</tr>
<tr>
<td>BSD license</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ODE</td>
<td><a href="http://www.ode.org">www.ode.org</a></td>
<td>Russel Smith</td>
</tr>
<tr>
<td>Tokamak</td>
<td><a href="http://www.tokamakphysics.com">www.tokamakphysics.com</a></td>
<td>David Lam</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dVC2d</td>
<td>robotics.cs.rpi.edu/dvc</td>
<td>Steve Berard et.al</td>
</tr>
<tr>
<td>DYNAMechs</td>
<td>dynamic.soungorge.net</td>
<td>Scott McMillan</td>
</tr>
<tr>
<td>MBSim</td>
<td>mbsim.berlios.de</td>
<td>TUM</td>
</tr>
<tr>
<td>Moby</td>
<td>physsim.sourceforge.net</td>
<td>Evan Drumwright</td>
</tr>
<tr>
<td>Springhead</td>
<td>springhead.info/index.en.php</td>
<td>S. Hasegawa et.al</td>
</tr>
<tr>
<td>Jiglib</td>
<td><a href="http://www.rowhouse.co.uk/jiglib">www.rowhouse.co.uk/jiglib</a></td>
<td>Danny Chapman</td>
</tr>
<tr>
<td>Aero</td>
<td><a href="http://www.aero-simulation.de">www.aero-simulation.de</a></td>
<td>H. Keller et.al</td>
</tr>
<tr>
<td>nV</td>
<td><a href="http://www.theophysicengine.com">www.theophysicengine.com</a></td>
<td>Unknown</td>
</tr>
<tr>
<td>SOFA</td>
<td><a href="http://www.sofa-framework.org">www.sofa-framework.org</a></td>
<td>3 INRIA teams</td>
</tr>
<tr>
<td>Proprietary</td>
<td>Proprietary license, Closed source</td>
<td></td>
</tr>
<tr>
<td>Chrono::Engine</td>
<td>ied.unipi.it/tasora</td>
<td>A. Tasora et.al</td>
</tr>
<tr>
<td>Vortexsim</td>
<td><a href="http://www.vxsim.com">www.vxsim.com</a></td>
<td>MathEngine team</td>
</tr>
<tr>
<td>NVidia PhysX</td>
<td><a href="http://www.ageia.com">www.ageia.com</a></td>
<td>PhysX team</td>
</tr>
<tr>
<td>AgX</td>
<td><a href="http://www.algoryx.se">www.algoryx.se</a></td>
<td>Ume university</td>
</tr>
<tr>
<td>Havok</td>
<td><a href="http://www.havok.com">www.havok.com</a></td>
<td></td>
</tr>
<tr>
<td>Newton</td>
<td><a href="http://www.newtondynamics.com">www.newtondynamics.com</a></td>
<td>Julio Jerez</td>
</tr>
</tbody>
</table>

Table 3.1: List of popular physics engines.

### 3.2.1 Introduction

Most physics engines in table 3.1 were implemented based on the theories proposed in [AP97] and [ST96]. First, their formulation are velocity-impulse time-stepping instead of the acceleration-force method proposed by David Baraff in his 1994 paper [Bar94]. Second, they use constraint-based method to handle contacts instead of penalty method. With strong focus on performance, these engines use a simplified linearized Coulomb friction. The simplifications included: removing the coupling between normal and friction impulse and limit the linearized friction.
direction to 2 or 4. With such implications, the final equations system forms a convex MLCP that is much easier to solve.

At the beginning, these physics engine use the collision detection routines from of penalty methods directly. That could be the reason why Anitescu-Potra \[AP97\] method, which shares the same contact information with penalty methods, is more popular than the earlier Stewart-Trinkle \[ST96\] method. Stewart-Trinkle formulation requires different active set of contacts from collision detection. In principle, the gaming physics engines ask the collision detection for a set of contact points that are already in penetration. Then, they form a system of contact constraints to stop further penetrations. After that, the stabilization steps can be used to resolve penetrations.

Performance is very important for these physics engines so most of them use a linear solver to find the solutions of the convex MLCP problems. One early reference for this linear solver can be found in the book ”Linear Complementarity, Linear and Nonlinear Programming” by Katta G. Murty \[Mur88\]. Basically, because the MLCP is monotone, in theory, an iterative scheme such as Gauss-Seidel can be used the problems. A recent similar approach unofficially called ”sequential impulses” was introduced in \[GBF03\]. It was then implemented and made popular by Erwin Coumans (Bullet Physics engine) and Erin Catto (Box2d engine). The approach iterates through each constraint, updates and clamps the accumulated impulse value to make sure the constraint is satisfied locally. Warm-start strategy is usually used to speed up the process. With a mechanism to retain contact information between two consecutive steps, the sequential impulse method works very well in practice. More implementation details can be found in Erin Catto GDC 2006 \[Cat06\] and 2009 \[Cat09\] talks. It can be shown that ”sequential impulses” method is equivalent to Gauss-Seidel method, but it gives developers more control over solution time and accuracy of the simulation.
3.2.2 Strengths

Gaming physics engines, such as Havok, PhysX, Bullet and Open Dynamics Engine, have contributed to the success of many games and films. Havok was even honored at the 59th Emmy in 2006 with a Technology & Engineering award. Besides the obvious software engineering advantages of these engines, the main reason for their success is the fact that they made the right compromise between speed and accuracy when picking the underlying theories to implement. For example, a simple linear friction approximation is enough for human eyes to have some senses of motion resisting forces without having to sacrifice performance.

To date, all four physics engines mentioned above contain at least one parallel constraint solver. PhysX even has its own dedicated hardware.

In conclusion, the biggest strength of the gaming engines is their strong focus on performance, user-friendly and ability to produce believable results.

3.2.3 Weaknesses

The major weakness of these engines is their accuracy which sometimes, creates simulation artifacts in computer games that were noticed by players. In this subsection, I list a number of accuracy problems that usually originate from their underlying models.

Figure 3.2 illustrates a popular problem in gaming physics. In both figures 3.2a and 3.2b the box came into impact with two rectangles from above. In figure 3.2a, the box was pushed up as expect but in figure 3.2b, it was rotated clock-wise instead. Two main reasons lead to this issue: the wait-and-correct non-penetration constraint, which only allow the system to act when penetration already exists, and the way collision detection returns the penetration depth, which is defined as the shortest distance to move objects apart. Gaming physics engines usually deal with this problem by reducing the time step (in effect, increase simulation...
Figure 3.2: Two similar cases with different simulation outcomes. In both cases, the big box was dropped into two thin rectangular obstacles below. The normal impulses are generated along the penetration distance which is defined as the shortest distance to separate two bodies. In 3.2a, the outcome is as expected but in 3.2b, the direction that the box and the left obstacle achieve shortest penetration distance is not the correct normal direction.

computing power requirement) and they avoid having thin objects. This figure also makes a case for the argument that only geometric information may not be enough for collision detection to do its job. Some gaming physics developers argue that if collision detection can retain history information, for example, use information from different time steps, then the problem in figure 3.2 can be avoided. The counter argument is that collision detection should only use geometric information because the dynamic information in current time step is obsolete and will not be updated until after the problem is solved. These arguments show the current chicken-and-egg relation between collision detection and solver. This problem appears in most geometrically explicit time stepping method where the input problem, like those illustrated in figure 3.2 stays constant during a time step, or in other words, the formulation is linear. In the next chapter, I will propose a new linear time stepping formulation that can address this problem.

In figure 3.4, the non-deterministic nature of Bullet simulation is shown. As I mentioned above, many of these physics engines rely on a linear constraint solvers. These solvers try to satisfy the constraints locally, one at a time and hoping for
Figure 3.3: Screenshots at different times of the box stack scene running in dVC3d

Figure 3.4: Screenshots at different times of box stack scene in Bullet in three runs with same initial condition like 3.3 and the same model parameters. One row contains frames of a simulation and the columns correspond to time t=2,4,6,8 s.

global convergence. The problem is the order which they iterate the constraints can affect the final solution when it does not reach convergence. And if they choose a fixed order, the solution will be biased toward a certain set of constraints and over
time it will decrease the simulation stability. Most gaming physics engines overcome this problem by using random constraint order at every time step. Thus, it causes non-determinism in simulations. In the end, gaming physics engines have to make a choice between stability and determinism.

Figure 3.4 demonstrates another weakness of gaming physics engines: they cannot simulate stable stacking. It is important to note that box is the best choice of shape to simulate shape stacking because box-box collision detection is very fast and robust. Also, the maximum number of contact points needed for stable box-box contact is only 4, much less than other shapes. It means trying to simulate stacking with other shapes under gaming physics will only get worse behaviors.

Another much discussed problem of gaming physics engines is called the tunneling effect, where a fast object can unrealistically travel through a thin wall in one time step. For most shooting games, gaming engines have to treat the bullets as special objects outside of the physical simulation system because of this problem.

Gaming physics also use an inaccurate approximation of friction. The friction model, which usually called boxed friction model, breaks the dependence of friction on the normal force by assuming the normal impulse is a known constant when calculating friction force. The formulation can be written as:

\[ W_f = 0, -\mu \bar{\lambda}_n \leq \lambda_f \leq \mu \bar{\lambda}_n \] (3.1)

where \( \bar{\lambda}_n \) is an estimation of normal force \( \lambda_n \). Usually, \( \bar{\lambda}_n \) was fixed at \( \bar{\lambda}_n = Mg \). The model minimizes the work made by friction forces so it gives the system a sense of movement resistance but it does not follow the basic principle of friction where the maximum force has to be proportion to the normal force. One problematic outcome of this model is that in box stacking, the lowest box unrealistically carries the same friction force as the top one. There are a number of proposed techniques
to overcome this problem. Building a contact graph [GBF03] is a popular approach. More details can also be found in Kenny Erleben’s thesis [Erl05].

3.3 Physical simulation methods in Robotics

3.3.1 Stewart-Trinkle formulation

Stewart-Trinkle (ST) formulation is the direct result of applying time-stepping method to the mathematical model of frictional rigid body physical simulation. It was first published in [ST96] and has been studied in theories and tested in experiments extensively over the years in many publications: [PT96, TPSL97, Ste98, Ste99, Ste03, BNT07, Tri03, BTN+07, BTN09, Ber09, BNAT10]. The formulation is similar to the equations of the computer model described in chapter 2. In this section, I will focus mostly on the behavior of Stewart-Trinkle formulation in simulation and comparing it with other formulations.

3.3.1.1 Strengths

The Stewart-Trinkle formulation is the current state-of-the-art formulation in simulation accuracy. In fact, its simulation results have been compared and verified by multiple experiments.

A good example to show the method strength is the case of vibrating plates simulation:

The system above consists of two components: a kinematically controlled plate and a dynamical part interacting with the plate.

What makes this experiment interesting in simulation perspective is that the parts are moved solely by friction between them and the vibrating plate. Thus, it is an excellent test for the friction model of any simulation method. We tried the simple friction model proposed by David Baraff [Bar94] and adopted by most current gaming physics engines, but the parts did not move during the simulation.
It can be shown that every small-amplitude periodic vibration of the plate is associated with a unique velocity field. Parts on the plate’s surface can be modeled as flowing through the velocity field. Different plate motions generate different fields.

Our papers [BNT09, Ber09] and Steve Berard’s PhD thesis [Ber09] report that the simulation results of Stewart-Trinkle formulation agree with the data from the experiments. Not only that, our implementation of Stewart-Trinkle was used to find the plate’s parameters that generates a new motion for the part.

The Stewart-Trinkle formulation was also used successfully to design a part feeder [STKP04] and to collect grasping statistics [Ber09]. Theoretically, David Stewart has proved that the time-stepping Stewart-Trinkle formulation solution approaches the solution of the DCP model in [Ste98, Ste03]. In conclusion, simulation accuracy is the biggest strength of Stewart-Trinkle formulation.
Figure 3.6: The plate is kinematically controlled by the vector function \( g(t) \). There are three forces acting on the part: the force due to gravity \( \lambda_{\text{app}} \), the non-penetration constraint force \( W_n \lambda_n \), and the frictional force \( W_f \lambda_f \).

![Diagram](image)

Figure 3.7: Direct comparison of the particle’s position between our simulation results and those of Vose et al., an experiments verified, closed-form simulator, for the Centrifuge motion.

(a) X-Position of particle    (b) Y-Position of particle

3.3.1.2 Weaknesses

One of the biggest problems in implementing the Stewart-Trinkle method is that its requirements on collision detection render the method incompatible with existing collision detection libraries. Being a penetration prevention method, it requires a different active set of contacts, which usually is a superset of the one used in correction methods. In order to prevent penetration, Stewart-Trinkle method needs a list of geometric features that could be in penetration at the end of the next time step. This list must also include the corresponding distances between the
features, including contacts with positive distances, which are quickly omitted in existing collision methods. The method can be forced to work with current collision detection, by setting its built-in constraint stabilization constant to zero, but it will suffer from the same problems of penetration correction methods described in previous section.

Extending current collision detection routines to include potential contacts is not a trivial task. Moreover, a collision detection library usually consists of different components that are implemented using different methods, each handles a certain pair of shapes. A collision detection function between two shapes can be classified into three main types: custom (usually simple pairs: box-box, sphere-sphere), Separating Axis Theorem (SAT) based [Ebe01] or GilbertJohnsonKeerthi (GJK) based [GJK88, OG97, OG01]. The lack of an unified theory in collision detection makes it harder to implement a Stewart-Trinkle compatible collision detection library.

The Stewart-Trinkle method with linearized Coulomb friction model is much more expensive to solve than the formulation used by gaming physics engines. With linearized friction, the formulation is non-convex and cannot be solved by fast meth-
ods such as the iterative Gauss-Seidel that works with gaming friction model. The resulting formulation is a mixed linear complementarity problem. Thus, there are only a few available solvers to choose from. Practically, at the moment, only PATH solver [FM96] can be used to solve medium size Stewart-Trinkle formulations. The problem is PATH has quite limited license that restricts its distribution along with out Stewart-Trinkle implementations. Such problem has affected the number of outside users of dVC2d. In this thesis, I proposed a number of new solution methods, based on free and open-source libraries, that are compatible with time-stepping Stewart-Trinkle formulation. More details will be discussed in chapter 6.

### 3.3.2 Anitescu-Potra formulation

Anitescu-Potra (AP) formulation was first introduced by Mihai Anitescu and Florian Potra in a 1997 paper [AP97]. Technically, the only difference between AP and ST formulation is the omission of the stabilization factor in the right hand side of AP non-penetration constraints. Specifically, they remove the term $\Psi^\ell_n h$ from equation (3.2). However, from implementation point of view, the two methods have very different collision detection and solver requirements. And their simulation expected outcomes are also different.

Formally, the non-penetration constraint of ST can be constructed by applying time-stepping method to the mathematical model described in chapter 2:

$$0 \leq \lambda^{\ell+1}_n \perp W_T^T \nu^{\ell+1} + \Psi^\ell_n h + \frac{\partial \Psi^\ell_n}{\partial t} \geq 0 \quad (3.2)$$

where in AP, the constant part is set to zero:

$$0 \leq \lambda^{\ell+1}_n \perp W_T^T \nu^{\ell+1} + \frac{\partial \Psi^\ell_n}{\partial t} \geq 0 \quad (3.3)$$
The term $\frac{\partial \Psi^\ell}{\partial t}$ is the first order component in the Taylor series expansion and $\Psi^\ell_n$ is the current contact distance that was computed by the collision detection. Having $\Psi^\ell_n$ in the formulation give ST methods a built-in constraint stabilization factor that automatically correct penetrations during simulation due to numerical errors or missing contacts. In AP, penetration will also inevitably occur and the simulations have the freedom to choose correction methods to be applied after each time step.

One of the major differences between ST and AP is the choice of active contact list. Figure 3.9 shows that, in ST, we can, and actually should, anticipate a contact between two bodies when they are close in order to prevent penetration, while in AP, we cannot include the contact or the two bodies will unrealistically be forced to stay apart. The list of active contacts in AP should only include those represent touching ($distance = 0$) or in-penetration ($distance < 0$). In AP, we also need to apply certain correction steps to those with negative distances.

Figure 3.9: Difference between AP and ST formulation of the gap between two bodies in an active contact.

Thus, in the AP formulation, penetration is unavoidable due to its contact model choice. In ST, penetration can still occur due to numerical and linearization errors, but the method actively prevents penetration if collision detection reports correctly. However, AP formulation admits more solutions than ST. For example, figure 3.10 poses a valid input for AP, where the solution would force the rigid body
to keep its vertical coordinate, while it has no solution in ST.

Figure 3.10: A rigid body with sphere shape stays inside two rectangular obstacles. In AP, the system has solution but not in ST.

3.4 Other notable methods

3.4.1 Optimization-based formulation

From chapter 2 we can see that complementarity theory is a naturally choice to model the intermittency of rigid body simulation. One problem with complementarity theory is that there are too few available solvers. In contrast, there are many optimization libraries released every year. The use of optimization methods in simulation started with papers by Victor Milenkovic and Harald Smith [MS01] and [SM04]. In which, the authors proposed a method to reformulate the problems of physical simulation as quadratic programming (QP) problems. The non-penetration constraint can be easily express in convex QP form but the non-convex Coulomb friction model has to be modified to arrive at the final convex QP form. The main position update loop of the algorithm looks similar to many complementarity based algorithm:

This approach receives great interest from computer graphics community. Later, Kaufman et.al improve Milenkovic’s algorithm by proposing a method that requires to solve only one QP each time step in [KEP05]. Later, in [KSJP08], the
authors continue to proposed a more efficient and accurate solver that can be used to simulate a challenging scene consists of a rigid house of card as shown in figure 3.12.

The advantage of this approach is clear. It relies on more developed and robust QP solvers instead of MLCP solvers. But the disadvantage is that the method has no theoretical proof of global convergence of the trajectories provided by these time-stepping methods to discrete-time solutions of original model. It means even though the method can produce plausible results after a small number of steps, it may never converge to the solution of the original problem.

3.4.2 Prox-function formulation

In 2003, Glocker et.al [LG03], [SG05] and [SG06] proposed a method that relies on convex analysis [Roc86] to model unilateral contacts and friction using regularized and set-valued force laws. Set-valued laws can be expressed as normal
cone inclusions. By doing so, a broad scope of non-smooth interactions can be described by the same mathematical structure, which allows a unified treatment of arbitrary friction laws, for examples, classical Coulomb friction, Coulomb-Contensou friction or anisotropic friction \cite{Stu08}. The inclusions which are associated with the individual set-valued laws can be transformed into projective equations. These equations can be solved iteratively by a Jacobi or Gauss-Seidel based approach \cite{SG05} and \cite{Stu08}.

The set-valued or inclusion laws can be expressed through prox-function, which can be defined for a convex set $C \subset \mathbb{R}^n$, $n \in \mathbb{N}$ as:

$$\text{prox}_C(x) = \arg \min_{x^* \in C} \|x - x^*\|, \quad x \in \mathbb{R}^n.$$

Applying prox-function to all normal and friction constraints will arrive at a set of non-smooth, nonlinear equations can be solved by a root-finding algorithm, e.g. a fixed-point iteration scheme. The projection equations depend on a non-negative auxiliary parameter $r$ which will drive the speed of convergence. The tragedy to
pick $r$ was discussed by Martin Förg in [FGNU06] and his PhD thesis (in German) [Fr07].

Recently, Thorsten Schindler and Jeff Trinkle jointly study the difference between prox-function and the LCP-based Stewart Trinkle formulation in [TS11]. Basically, the results show that the two formulations are equivalent. Indeed, we can convert any complementarity condition to prox-function form and vice versa as shown in Saeed Ebrahimi’s thesis [Ebr07]. The fixed-point scheme solver of prox-function may be faster and can be adapted to run in parallel. But prox-function formulation may diverge with certain values of $r$ even in very simple scene consist of a particle and a plane.

### 3.5 Improvements of current methods

In this section, I proposed a number of important improvements for current methods described in this chapter. Most of the improvements are made for Stewart-Trinkle and Anitescu-Potra methods but gaming physics engines such as Bullet, Havok and PhysX should also benefit from the ideas.
3.5.1 Adapt gaming style Bullet collision detection routines to Stewart-Trinkle

Stewart-Trinkle time stepping method would not be so useful without a compatible collision detection method, so I propose an improvement to make current collision detection routines work with it.

The idea is instead of running collision detection with objects’s current position, I use their future interpolated positions instead. And the result will give us a good estimate on which contacts would be in penetration in the future. Then I update the list of contact points, which I will refer to as speculative contacts, by projecting them back to current positions. Note that the overhead cost for using the speculative contact, which only involves transforming bodies positions between two consecutive frames, is insignificant compared to the costs of the solver and collision detection.

I implemented the improvement in Bullet physics, a free, open-source collision detection and gaming physics engine. Bullet collision detection works in two modes: one-shot or incremental contact generation. In one-shot mode, Bullet will try to get all active contacts in one call while in incremental mode, it will update the list gradually during simulation.

Figure 3.14 shows the how the method described above works in the one-shot and incremental contact modes in a cylinder stacking scene. The reason I chose cylinders is because cylinder-box and cylinder-cylinder collision detection routines in Bullet collision are using a more general GJK collision detection method, while box-box is a very robust custom routine. As we can see, figure 3.14 shows a big improvement to accuracy of Stewart-Trinkle method using speculative contacts. In various other tests, Stewart-Trinkle method also benefits greatly from the use of speculative contact.
Figure 3.14: Speculative contact (S) and one-shot (O) collision detection tests in dVC3d. First row: no S, no O, second row: S only, third row: O only, fourth row: both S and O. Each column corresponds to $t=[0,0.33,0.67,1]$ s.

Figure 3.14 reveals a counter-intuitive result between the run using speculative and one-shot (last row) and the one using speculative alone. It seems that using one-shot mode make the cylinder stacking simulation less stable than using incremental mode. Take a closer look, I found the explanation on the way Bullet and many other gaming physics engines reporting contacts in collision detection. Figure 3.15 shows how Bullet triangulates the cylinder’s end circle and generates contacts based on those vertices. However, Bullet, which usually focuses more on performance, limit the maximum number of contact points to 4, so one-shot collision detection could return an unbalanced set of points that could topple the cylinder when impacts occur. The cylinder only jiggle when the impact impulses are high enough, so it can remain still even with an unbalanced set of contact points such as the set showed in figure 3.15. The problem is not as severe in the incremental collision detection mode because Bullet already takes an extra care to make sure the contacts are generated
Indeed, when I switch cylinder to box shape, the simulation with one-shot collision behaves as expected as showed in figure 3.16. This example demonstrates the lack of clear, unambiguous rules for collision detection. I can only find out the upper limit in the number contact points between two bodies in Bullet by looking at its source code. And as you can imagine in this example, it could cause problems to the users later.

### 3.5.2 Modify Stewart-Trinkle friction approximation

Stewart-Trinkle formulation contains two main distinct differences from other formulations. One is the penetration prevention normal constraint which can take in potential contact pairs unlike others can only handle active (in penetration or
touching) ones. The other is the linearized Coulomb friction which most gaming physics engines do not use because of speed requirements.

I modified Stewart-Trinkle formulation by replacing its Coulomb friction model with the one from gaming physics engine [Bar94] and the convex friction model proposed by Mihai Anitescu [AH04b] and [Ani06].

For linearized friction cones, the original Stewart-Trinkle formulation can be written as a mixed LCP:

\[
\begin{align*}
M\nu^{\ell+1} &= M\nu^{\ell} + h(W_n\lambda^{\ell+1}_n + W_f\lambda^{\ell+1}_f + \lambda^{\ell\text{ app}} + \lambda^{\ell\text{ vp}}) \\
q^{\ell+1} &= q^\ell + hG\nu^{\ell+1} \\
0 &\leq \lambda^{\ell+1}_n \perp \psi_n(q^{\ell+1}) \geq 0 \\
0 &\leq \lambda^{\ell+1}_f \perp W_f\nu^{\ell+1} + E\sigma^{\ell+1} \geq 0 \\
0 &\leq \sigma^{\ell+1} \perp U\lambda^{\ell+1}_n - E^T\lambda^{\ell+1}_f \geq 0
\end{align*}
\]

In matrix form:

\[
\begin{bmatrix}
0 \\
\rho^{\ell+1}_b \\
\rho^{\ell+1}_n \\
\rho^{\ell+1}_f \\
s^{\ell+1}
\end{bmatrix}
= 
\begin{bmatrix}
-\lambda^T_b & W^T_b & 0 & 0 & 0 \\
W_b & 0 & 0 & 0 & 0 \\
W^T_n & 0 & 0 & 0 & 0 \\
W^T_f & 0 & 0 & 0 & E \\
0 & 0 & U & -E^T & 0
\end{bmatrix}
\begin{bmatrix}
\nu^{\ell+1} \\
p^{\ell+1}_b \\
p^{\ell+1}_n \\
p^{\ell+1}_f \\
\sigma^{\ell+1}
\end{bmatrix}
+ 
\begin{bmatrix}
\Psi^\ell_b + \frac{\partial \Psi^\ell}{\partial t} \\
\Psi^\ell_n + \frac{\partial \Psi^\ell}{\partial t} \\
\Psi^\ell_f + \frac{\partial \Psi^\ell}{\partial t}
\end{bmatrix}
\begin{bmatrix}
\rho^{\ell+1}_b \\
\rho^{\ell+1}_n \\
\rho^{\ell+1}_f \\
s^{\ell+1}
\end{bmatrix}
\]

\[
0 \leq 
\begin{bmatrix}
\rho^{\ell+1}_b \\
\rho^{\ell+1}_n \\
\rho^{\ell+1}_f \\
s^{\ell+1}
\end{bmatrix}
\perp 
\begin{bmatrix}
p^{\ell+1}_b \\
p^{\ell+1}_n \\
p^{\ell+1}_f \\
\sigma^{\ell+1}
\end{bmatrix}
\geq 0
\]

(3.5)
3.5.2.1 Convex friction model

The convex friction model has the form:

\[
\begin{bmatrix}
0 \\
\rho_{b}^{\ell+1} \\
\rho_{n}^{\ell+1} \\
\rho_{f}^{\ell+1} \\
s^{\ell+1}
\end{bmatrix}
= 
\begin{bmatrix}
-M & W_{b} & W_{n} & W_{f} & 0 \\
W_{b}^{T} & 0 & 0 & 0 & 0 \\
W_{n}^{T} & 0 & 0 & 0 & -U^{T} \\
W_{f}^{T} & 0 & 0 & 0 & E \\
0 & 0 & U & -E^{T} & 0
\end{bmatrix}
\begin{bmatrix}
\nu_{b}^{\ell+1} \\
\nu_{n}^{\ell+1} \\
\nu_{f}^{\ell+1} \\
\sigma^{\ell+1}
\end{bmatrix}
+ 
\begin{bmatrix}
M \nu_{b}^{\ell} + p_{\text{app}} + p_{\text{vp}} \\
\psi_{b}^{\ell} + \frac{\partial \psi_{b}^{\ell}}{\partial t} \\
\psi_{n}^{\ell} + \frac{\partial \psi_{n}^{\ell}}{\partial t} \\
\frac{\partial \psi_{f}^{\ell}}{\partial t}
\end{bmatrix}
\begin{bmatrix}
\rho_{b}^{\ell+1} \\
\rho_{n}^{\ell+1} \\
\rho_{f}^{\ell+1} \\
s^{\ell+1}
\end{bmatrix}
\]

(3.8)

\[
0 \leq \begin{bmatrix}
\rho_{b}^{\ell+1} \\
\rho_{n}^{\ell+1} \\
\rho_{f}^{\ell+1} \\
s^{\ell+1}
\end{bmatrix} \perp \begin{bmatrix}
p_{b}^{\ell+1} \\
p_{n}^{\ell+1} \\
p_{f}^{\ell+1} \\
\sigma^{\ell+1}
\end{bmatrix} \geq 0
\]

(3.9)

As you can see, the only difference is the additional term \(-U^{T}\) that change the matrix to be positive definite. Thus, it makes the system constraint matrix becomes convex. The changes also allowed the problem to be solved by convex cone solver as mentioned in [TA11], [TA10] and [AT10].

3.5.2.2 Gaming friction model

In gaming physics engines, the friction law was approximated by equation (3.1). Thus, the final matrix from of Stewart-Trinkle formulation with gaming friction can be written in MLCP from as:

\[
\begin{bmatrix}
0 \\
\rho_{b}^{\ell+1} \\
\rho_{n}^{\ell+1} \\
\rho_{f}^{\ell+1} \\
s^{\ell+1}
\end{bmatrix}
= 
\begin{bmatrix}
-M & W_{b} & W_{n} & W_{f} & 0 \\
W_{b}^{T} & 0 & 0 & 0 & 0 \\
W_{n}^{T} & 0 & 0 & 0 & 0 \\
W_{f}^{T} & 0 & 0 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
\nu_{b}^{\ell+1} \\
\nu_{n}^{\ell+1} \\
\nu_{f}^{\ell+1} \\
\sigma^{\ell+1}
\end{bmatrix}
+ 
\begin{bmatrix}
M \nu_{b}^{\ell} + p_{\text{app}} + p_{\text{vp}} \\
\psi_{b}^{\ell} + \frac{\partial \psi_{b}^{\ell}}{\partial t} \\
\psi_{n}^{\ell} + \frac{\partial \psi_{n}^{\ell}}{\partial t} \\
\frac{\partial \psi_{f}^{\ell}}{\partial t}
\end{bmatrix}
\begin{bmatrix}
\rho_{b}^{\ell+1} \\
\rho_{n}^{\ell+1} \\
\rho_{f}^{\ell+1} \\
s^{\ell+1}
\end{bmatrix}
\]

(3.10)
\[
\begin{bmatrix}
-\infty \\
0 \\
-\overline{p}_n
\end{bmatrix} \leq \begin{bmatrix}
\nu^{\ell+1} \\
\rho_n^{\ell+1} \\
\rho_{\ell+1} \\
\rho_n
\end{bmatrix} \leq \begin{bmatrix}
\infty \\
\infty \\
\infty
\end{bmatrix}
\] (3.11)

3.5.2.3 Examples

(a) Initial configuration

(b) A steady state configuration where all 9 boxes sliding down the ram.

Figure 3.17: A simple friction scene with 9 rigid bodies

I implemented Stewart-Trinkle time-stepping methods with three different friction models mentioned above in Matlab. The running time, statistics of three different models for the same scene 3.17 are as follow.

At steady states, where all 9 rigid bodies are in contact with the ramp below, the size of the main matrix and average running time for different friction models are:

- **Linearized Coulomb friction.** Main matrix size = 99, average time = 0.0111 second.

- **Convex friction.** Main matrix size = 99, average time = 0.0103 second.

- **Gaming friction.** Main matrix size = 63, average time = 0.0083 second.
The running time for 10 consecutive time steps are shown in figure 3.18.

Figure 3.18: Running time of 10 steps of different friction models measured with set up in figure 3.17.

We can observe that the gaming friction model has lowest cost both in term of memory (smaller matrix size) and computation (faster on the same machine). The Convex friction model results in the same problem size but a little faster than the non-convex linearized friction model. But the convex friction model has one serious problem. Its modification of non-penetration constraint may cause a completely physically valid configuration to be unsolvable as in figure 3.19.
Figure 3.19: Convex friction model failed when the second box is close enough to the first one. Both linearized Coulomb and gaming friction models work as expected.

Figure 3.20: A *ghosting* trail of 200 frames when simulating a triangle sliding on the ground. Convex friction model shows anomalous behavior where the triangle jumps instead of slides on the ground.

Figure 3.21: Linearized Coulomb friction model shows correct result when simulating the same scene in figure 3.20.
CHAPTER 4
Locally non-convex method

4.1 Introduction

Physical simulation of rigid bodies with Coulomb friction has come a long way since Lötstedt first developed complementarity problem formulations \cite{Lot81,Lot82}. Until now, there are various methods that are capable of handling complicated tasks from video games and virtual reality \cite{Bar94,Erl05}, to graphics and haptic application \cite{KSJP08}, to robotics, machine design and virtual prototyping \cite{ST96,Tri03,CBAT07a,AP97,AH04a,LG03}. However, they all share the same contact model that consists of a vertex and a feature to serve as the basis to prevent penetration. This simple contact model is simple easy to use and understand but it fails to handle the cases where the shape of the free space in the neighborhood of the contact point is non-convex such as the one shown in figure 4.1. Moreover, it requires complicated collision detection that is often not trivial to implement. In this chapter, I start with defining a new extended model that can accurately model locally non-convex free space contact. Then, I develop new formulations based on it. Then, the solution methods of the new formulations are presented. The new collision detection method for this new advance contact model will be discussed in the next chapter.

4.1.1 Background

The goal of a time-stepping method is to produce estimates of a dynamic system state at a discrete set of times. Given the state at the current time, the task is to formulate a time-stepping sub problem whose solution yields the state at the
next time in the set. Typical time-stepping methods for multi rigid bodies systems with unilateral contacts accomplish this by cycling between two main functions: one that computes distances between geometric features of the bodies, and one that takes this list, formulates the dynamics’ subproblem, solves it, and updates the system state. This process is repeated until the final time is reached.

The most important geometric information in most current methods to deal with non-penetration constraint is the set of geometric features (vertices and faces or edges and edges) that are in contact (distance = 0), nearly so (distance is positive, but small) or in penetration (distance is negative). I will refer to the list of such feature pairs as the active (contact) set. This set is needed to formulate the dynamics subproblem so that it can prevent inter-penetration of rigid bodies. The active set is normally obtained from a collision detection routine.

Current time-stepping methods can be classified into two main types: correction methods [Bar94, Mir96, AP97] report only pairs that are in penetration or touching and prevention methods [ST96, Tri03, BTN+07] that also report pairs that could collide in the next time step. In correction methods, constraints are formed to stop the current penetrations from getting worse. Additional correction steps can be used to reduce the depth of penetration or eliminate it all together. Even though correction methods can employ the error-correction steps, they have a major drawback: penetration is unavoidable due to its wait-and-correct behavior and to numerical errors during simulation. In general, penetration should be avoided during simulation where possible, not just because it is physically incorrect but also because there are cases where geometric information is not sufficient to recover from a penetration. In correction methods, penetration depth scales linearly with simulation time step and object speed. It makes correction methods sensitive to those runtime factors. This dependence increases the complexity of collision detection for correction methods. It has to treat high-speed objects, bullets, for example, in
a different manner than other objects to avoid *tunneling effect* i.e. objects passing through others in one time step. Reducing the simulation time step to properly handle high-speed objects is not a good option because object’s speed can vary widely while a fixed time step is crucial for realtime simulation. Smaller time steps also increase the computing power required to simulate a fixed duration.

Prevention methods [ST96, Tri03, BTN+07] have less penetration during simulation, because they anticipate penetrating pairs before collisions. These methods also have a built-in correction step to eliminate of penetrations caused by linearization and numerical errors. This type of method is less sensitive to the value of the time step and object speed because they can prevent *tunneling effect* by cautiously activating contact pairs that include the high speed objects. Prevention methods also require different active sets than correction ones. They need collision detection to include not only penetrating and in-contact pairs but also potentially colliding pairs. This difference explains the lack of prevention methods in popular physics engines as all current collision detection routines are only designed to work with corrective methods. The only implementation of this type of method known to us is our simulation package, our dVC physics engines [BNT07], which uses a simple heuristic that compares geometric distances between possible pairs and a velocity-sensitive constant to choose the active set. It works well in general, but the collision detection is also very complicated especially in three-dimensional cases.

### 4.1.2 A simple contact model for locally non-convex free space

The common contact model in current methods is the simple one between a point and a face (or edge in planar case). An active constraint associated with this simple contact model keeps the distance $\psi_{in}$ between its point and face from becoming negative. Usually for the *ith* active contact, the constraint has the form:
where $\psi_i(n)$ is a signed distance function or gap function for the $i$th contact with the property $\psi_i(n(t)) > 0$ implies separation, $\psi_i(n(t)) = 0$ implies touching, and $\psi_i(n(t)) < 0$ for interpenetration. Note that in general, there is no closed-form expression for $\psi_i(n(t))$ so usually approximation values are used instead. $\lambda_i$ is the force or impulse needed to prevent $\psi_i$ from becoming negative.

Compactly, we can write the non-penetration constraint for all contacts as

$$0 \leq \lambda_n \perp \psi_n(q, t) \geq 0$$

(4.2)

where $\psi_n$ and $\lambda_n$ are the concatenated vectors of all the signed distance functions and normal forces (impulses) respectively.

Equation (4.2) over-constrains the system at those contacts where the local free space is non-convex. It can be illustrated in two simple cases below.

Figure 4.1: Locally non-convex free space

If we write the constraints in the form (4.2) then the result must satisfy: $\psi_{1n} \geq 0$
and $\psi_{2n} \geq 0$ which means in the next time step, the particle $P$ cannot leave the cone (I) while physically, it should be allowed to move into cone (II) and (IV). In the case

![Diagram](image)

Figure 4.2: A more complicated scene resembling a robot moving through a room

of figure 4.2 formulation (4.2) is infeasible as there is no point in space that has non negative distances with all lines that support the active edges (the ones that intersect with the circle interior in the figure).

![Diagram](image)

Figure 4.3: A test case where robot need to insert a rectangular block into a closely fitted hole

In conclusion, current methods not only cannot simulate the particular cases shown in figures 4.1 and 4.2 but also rely on complicated collision detection routines to find the active set. It is not trivial to implement collision detection algorithms.
needed for penetration prevention. In fact, the current development of continuous collision detection suggests that only geometric information is not enough to find the active set. Even so, one can always find the cases where collision detection fails to identify the right active set with this simple contact model.

4.1.3 Previous works and summary of results and contributions

Kevin Egan [KE03] proposed a way to handle non-convex configurations but his method requires a trial-and-error parameter tuning process to approximate the non-convex region. Also, it is not easy to relate the formulation used in [KE03] to physical laws. The method proposed in this chapter can be shown to closely follow the underlying physics model and also accurately captures the non-convexity of the free spaces.

The method described in this chapter extends the current simple contact model to include not only the geometric features but also information about the relations between them. The main advantages of the new extended contact model proposed in this work are two fold: it solves the problem that all current simple contact models have with locally non-convex free spaces, and it reduces the dependency of the simulation results on the internal details of collision detection.

4.2 Modeling non-convex configuration space

4.2.1 Extended contact model

The new extended contact model begins with the assertion that we need at most one impulse to prevent penetration between a vertex and a convex shape. Thus, the extended contact between two bodies is defined by:

- A vertex of the first body.
- A list of features of the second body.
The physical meaning of the current contact model described in [Bar94, ST96, AP97] is that when activated, it prevents the vertex from penetrating the half space that contains the contact face. The new extended contact model extends the idea by replacing that one contact face with a convex shape (defined by a list of faces) and physically prevents the contact vertex from penetrating the shape interior.

It is worth noting that we can choose any convex subset of the second geometric body or the whole body itself if it is convex. If the second body geometry is convex, any convex hulls of a subset of its vertices belong to the body geometry also. Thus, it is easy to generate new contact information for a convex body: a simple heuristic that picks all facets with Euclidean distance to contact vertex smaller than a certain threshold to include in the extended contact should work well. If the shape of the second body is not convex, this extended contact model can still accurately model the object by decomposing the shape into a list of convex shapes, then attach one contact to each of them when needed.

It is easy to see that the extended contact model not only solves the problems with contacts that have locally non-convex free spaces, it also gives the users more freedom in forming the active sets. Usually, it is not obvious how to determine which facet will make contact with a vertex in next time step to form the active set. The extended contact model gives the collision detection the flexibility to handle
this uncertainty better: it can be conservative by picking a big convex shape when it is hard to guess the correct set or even just one facet if it is obvious (see figure 4.5). Normally, for current methods, the only way to tackle the uncertainty in set selection problem is to reduce the time step or to use a continuous collision detection algorithm which is expensive and could lead to exhaustive exponential search to find the right order for all the events. In reality, all current common physics engines rely on their collision detection ability to pick the right active set. Picking the wrong one would result in unrealistic behaviors. Extra contacts in the active set lead to unwanted obstacles during simulation while missing contacts lead to deep penetrations. The extended contact model reduces collision detection complexity because it does not need the unique contact facet corresponds with the contact vertex but a list of possible facets that contain it. With this extended contact model, collision detection is well defined and easier to implement. Actually, one can heuristically pick the set of facets to form the contact then calculate the geometric distances between the vertex and all the facets in the list then feed that information to the dynamic step. The correct active facet and contact forces will be obtained along with solving the dynamics subproblem.

4.2.2 Mathematical implementations

In order to handle the new extended contact model correctly, we need to keep the vertex from penetrating the convex shape in the next time step. This task can be separated into two main sub-tasks: accurately model the non-convex free space corresponds to the contact shape geometry, find the correct facet in the shape that the vertex collides with, then find the correct normal (force) impulse to prevent penetration. We separate the task into two because they are actually different. In following parts, we will provide mathematical equations for each part, we start with a simple contact (with only two edges or faces) then extend the formulation to a
Figure 4.5: There are cases where user can use current information to choose the set of edges to minimize the chance of penetration. (a): the vertex $P$ of a polygon moves relatively slowly and is close to the obstacle so only edge 1 is needed. (b): the vertex moves fast and the object below is rotating so edge 1, 2 and 3 need to be included. (c): Speeds of the vertex and object’s spinning are very high, so we may need to consider adding ALL edges.

4.2.2.1 Simple contact

We have argued that for the case of locally non-convex configuration spaces, treating all non-penetration constraints conjunctively will lead to error. For the case shown in figure 4.1, we should not constrain both of $\psi_{1n}$ and $\psi_{2n}$ but only one of them to be nonnegative or equivalently:

$$\max(\psi_{1n}, \psi_{2n}) \geq 0$$

We leverage the existing Linear Complementarity framework to encode equation (4.3).

Lemma 4.2.1 Given $a, b \in \mathbb{R}$, $b = \max(a, 0) \iff 0 \leq b - a \perp b \geq 0$

Proof $\Rightarrow$ direction: from $b = \max(a, 0)$, we have $0 \leq b - a \perp b \geq 0$

- case $a \leq 0$: $\max(a, 0) = 0 \Rightarrow b = 0 \Rightarrow 0 \leq b - a \perp b \geq 0$. 
\begin{itemize}
  \item case \( a > 0 \): \( \max(a, 0) = a \Rightarrow b = a \Rightarrow 0 \leq b - a \perp b \geq 0 \).

  \text{\( \iff \) direction: from \( 0 \leq b - a \perp b \geq 0 \), we have \( b = \max(a, 0) \).}

  \item case \( b - a = 0, b \geq 0 \): obviously \( b = \max(a, 0) = a \)

  \item case \( b = 0, b - a \geq 0 \): because \( a \leq 0 \) so \( \max(a, 0) = 0 = b \)
\end{itemize}

**Lemma 4.2.2**  Given \( a, b \in \mathbb{R} \), \( b = |\min(a, 0)| \iff 0 \leq b + a \perp b \geq 0 \)

**Proof**  We call \( b = |\min(a, 0)| \) and \( 0 \leq b + a \perp b \geq 0 \) as equation (i) and (ii) respectively.

  \text{\( \implies \) direction: (i) holds, need to prove (ii)}

  \begin{itemize}
    \item case \( a \leq 0 \): \( |\min(a, 0)| = -a = b \Rightarrow b + a = 0 \Rightarrow (ii) \) holds.
    \item case \( a > 0 \): \( |\min(a, 0)| = 0 = b \Rightarrow (a + b)b = 0 \Rightarrow (ii) \) holds.
  \end{itemize}

  \text{\( \iff \) direction: (ii) holds, need to prove (i)}

  \begin{itemize}
    \item case \( b + a = 0, b \geq 0 \Rightarrow a \leq 0 \Rightarrow |\min(a, 0)| = -a = b \)
    \item case \( b = 0, b + a > 0 \Rightarrow a > 0 \Rightarrow |\min(a, 0)| = 0 = b \)
  \end{itemize}

Using these two lemmas, we can transform the constraint (4.3) into linear complementarity conditions as follow:

\[
\max(\psi_{1n}, \psi_{2n}) = \psi_{2n} + \max(\psi_{1n} - \psi_{2n}, 0)
\]

(4.4)

Then define the variable \( c \) as follows:

\[
c = \max(\psi_{1n} - \psi_{2n}, 0)
\]

(4.5)
Using lemma 4.2.1, equation (4.5) can be written as a linear complementarity condition:

\[ 0 \leq c - (\psi_{1n} - \psi_{2n}) \perp c \geq 0 \]  

(4.6)

Then, the correct non-penetration constraint at the contact along edge 1 is:

\[ 0 \leq c + \psi_{2n} \perp \lambda_{1n} \geq 0 \]  

(4.7)

This constraint basically means that when the term \( c + \psi_{2n} \), which is equivalent to \( \max(\psi_{1n}, \psi_{2n}) \), becomes negative, the normal force (impulse) \( \lambda_{1n} \) along edge 1 will be positive to prevent the penetration.

Similarly, non-penetration constraint along edge 2 is:

\[ 0 \leq c + \psi_{2n} \perp \lambda_{2n} \geq 0 \]  

(4.8)

Here, \( \lambda_{1n} \) and \( \lambda_{2n} \) are normal forces (or impulses) to maintain condition (4.3). The above non-penetration constraint correctly prevents penetration, but it allows impulses to be generated on both edges, which is not physically correct. There should only be at most one impulse along the edge that the vertex will collide with in the next time step. So we need a constraint that allows no more than one of \( \lambda_{1n} \) and \( \lambda_{2n} \) to be positive.

**Lemma 4.2.3** Given \( a, b \in \mathbb{R}, a = 0, b \leq 0 \Leftrightarrow (\max(a, b) \geq 0) \land (\max(a, b) + |\min(a, 0)| = 0) \)

**Proof** Because \( \max(a, b) \geq 0 \) and \( |\min(a, 0)| \geq 0 \) then the only way to have \( \max(a, b) + |\min(a, 0)| = 0 \) is both reach equality. From \( |\min(a, 0)| = 0 \) we can have \( a \geq 0 \) so to keep \( \max(a, b) = 0 \) then \( a = 0, b \leq 0 \). The reverse direction is
obvious.

Using lemma 4.2.3, we can formulate a non-penetration constraint along edge 1 as:

\[
0 \leq c - (\psi_{1n} - \psi_{2n}) \perp c \geq 0 \\
0 \leq d_1 + \psi_{1n} \perp d_1 \geq 0 \\
0 \leq c + \psi_{2n} + d_1 \perp \lambda_{1n} \geq 0 \\
c + \psi_{2n} \geq 0
\] (4.9)

Equations (4.9) only allows the normal force (impulse) along edge 1 to be nonnegative when \( \psi_{2n} \leq 0 \) and \( \psi_{1n} = 0 \). Note that \( \psi_{2n} \leq 0 \) and \( \psi_{1n} = 0 \) physically means particle \( P \) is touching edge 1 in figure 4.1.

Similarly, a non-penetration constraint along the second edge is:

\[
0 \leq d_2 + \psi_{2n} \perp d_2 \geq 0 \\
0 \leq c + \psi_{2n} + d_2 \perp \lambda_{2n} \geq 0
\] (4.10)
4.2.2.2 General contact case

In general, the contact has the form \((P, [1, 2, \ldots n])\). We can extend simple case formulations \((4.9)\) and \((4.10)\) as follow:

\[
0 \leq c_2 - \psi_{2n} + \psi_{1n} \perp c_2 \geq 0
\]

\[
0 \leq c_3 - \psi_{3n} + c_2 + \psi_{1n} \perp c_3 \geq 0
\]

\[
0 \leq c_n - \psi_{nn} + c_{n-1} + \ldots + c_2 + \psi_{1n} \perp c_n \geq 0
\]

\[
0 \leq d_1 + \psi_{1n} \perp d_1 \geq 0
\]

\[
0 \leq d_n + \psi_{nn} \perp d_n \geq 0
\]

\[
0 \leq d_1 + c_2 + \ldots + c_n + \psi_{1n} \perp \lambda_{1n} \geq 0
\]

\[
0 \leq d_n + c_2 + \ldots + c_n + \psi_{nn} \perp \lambda_{nn} \geq 0
\]

\[
c_2 + \ldots + c_n + \psi_{1n} \geq 0
\]

where \(c_2, \ldots, c_n, d_1, \ldots, d_n\) are new variables, \(\psi_{1n}, \psi_{2n}, \ldots, \psi_{nn}\) are distances between the vertex of this contact and the edges.

A problem arises here, in \((4.11)\), the last inequality is not in the form of a linear complementarity condition. We will discuss about how to solve this problem in the next section.

In comparison, the same type of constraints based on most of current methods has the form:

\[
0 \leq \psi_{in} \perp \lambda_{in} \geq 0, \ i = 1 \ldots n
\]
That means for a contact with \( n \) edges on a convex object, there will be \( 3n - 1 \) linear complementarity conditions. Which is almost three times as many as used by the current methods. However, the advantage is that we can take bigger time steps.

This new formulation relies on the solver to choose the face on which the actual contact will occur. That makes the problems harder to solve than say, Stewart-Trinkle formulation. Also, in reality, most of the time we can detect the correct face of the contact. This observation leads to a much simpler formulation.

### 4.2.2.3 General contact case with predefined contact face

This formulation make use of the assumption that we can easily select the normal direction of contact force for a contact in many cases. The contact now has the form \((P, [1, 2, \ldots n], u)\) where \( u \) is the index of chosen face. Then, the formulation simplifies to:

\[
0 \leq c_2 - \psi_{2n} + \psi_{1n} \perp c_2 \geq 0 \\
0 \leq c_3 - \psi_{3n} + c_2 + \psi_{1n} \perp c_3 \geq 0 \\
\vdots \\
0 \leq c_n - \psi_{nn} + c_{n-1} + \cdots + c_2 + \psi_{1n} \perp c_n \geq 0 \\
0 \leq c_n + c_{n-1} + \cdots + c_2 + \psi_{1n} \perp \lambda_{un} \geq 0
\]

(4.13)

The last equation is equivalent to:

\[
0 \leq \max(\psi_{1n}, \psi_{2n}, \ldots, \psi_{nn}) \perp \lambda_{un} \geq 0
\]

(4.14)

The physical intuition here is that when the maximum predicted distance of the contact vertex to the possible contact faces progresses to negative value, a normal force perpendicular the chosen face \( u \) will kick in to prevent the penetration.
This formulation is much cheaper to solve with the main matrix size is only a third compared to the general case. Indeed, it accurately models the non-convexity of the contact geometry and can be solved easily using PATH.

4.3 Mathematical properties and solution methods

4.3.1 Solutions method for extended contact

It is natural to formulate equations (4.11) as a LPCC (Linear Program with Complementarity Constraints) form [HMP+08]:

\[
\begin{align*}
\text{minimize} & \quad c' x + d' y \\
\text{subject to} & \quad A x + B y \geq f \\
& \quad 0 \leq y \perp q + N x + M y \geq 0 \\
& \quad x \geq 0 \\
& \quad x \in \mathbb{R}^n, y \in \mathbb{R}^m, f \in \mathbb{R}^k
\end{align*}
\]

An LPCC is a harder problem than a normal LCP so the mentioned solver can only be used for small problems (less than 500 variables).

Another possible approach to solve the resulting problem from extended contact without predefined acting normal is to reformulate it as a LCP. To do this, I reformulated:

\[ c_2 + \cdots + c_n + \psi_{1n} \geq 0 \]

as a complementarity condition:

\[ 0 \leq c_2 + \cdots + c_n + \psi_{1n} - \alpha \cdot e \perp e \geq 0 \]

where \( \alpha \) is a non-negative number and \( e \) is a dummy variable.
Then the resulting problems can be solved using a standard Linear Complementarity solver (e.g. PATH [FM96]). The problem with this method is that the term $-\alpha$ will show up in the main matrix’s diagonal of this LCP problem, which could cause numerical problems. Nonetheless, PATH has been able to successfully solve all the problems we have tried.

In practice, the extended contact with predefined acting normal is more desirable because it is much cheaper and produce almost identical results. Theoretically, the main difference between the two models is that in this model, the solver will have to find the contact normal, while in the other, we have to guess. In the next chapter, we will see how collision detection can choose the right acting normal for an extended contact.

### 4.3.2 Solution method for extended contact with predefined acting normal

The main matrix of equation $4.13$ has the form:

$$
\begin{bmatrix}
-M & W_b & W_n & W_f & 0 & 0 \\
W_b^T & 0 & 0 & 0 & 0 & 0 \\
W_n^T & 0 & 0 & 0 & E_1 & 0 \\
W_f^T & 0 & 0 & 0 & 0 & E \\
W_a^T & 0 & 0 & 0 & E_2 & 0 \\
0 & 0 & U & -E^T & 0 & 0
\end{bmatrix}
$$

(4.15)

With $W_a^T$ is formed by stacking all auxiliary contact wrenches together. Normally, every extended contact that has more than one distances will have an auxiliary contact wrench. Specifically, an extended contact $P_i [e_1, e_2, \cdots, e_d]$ has auxiliary
contact wrench $W_{ia}^T = \begin{bmatrix} W_{1n}^T - W_{2n}^T \\ \vdots \\ W_{1n}^T - W_{dn}^T \end{bmatrix}$. For the same extended contact, $E_{i1}$ is the lower triangular matrix with all non-zeros equal to one and all elements in $E_{i2}$ are equal to one.

Thus, the new extended contact formulation has a final matrix form quite similar to the original Stewart-Trinkle formulation in (2.75). But the term $W_a^T$ breaks the co-positive property of the matrix.

In practice, the resulting LCP can be solved easily using PATH and the new solvers I developed in chapter 6. The size of the LCP matrix associated with this contact model is actually smaller than original Stewart-Trinkle because it only uses one friction condition for a group of features instead of one for friction condition for each feature as in Stewart-Trinkle formulation.

### 4.4 Numerical results

#### 4.4.1 Box-ramp

We simulate a box falling then sliding down a cracked ramp. Note that the crack is small so physically, the box should be able to slide to the end of the ramp.

![Figure 4.6: New method, time step $h = 0.008$](image)

With the old contact model, the incorrect model of locally non-convex free
space introduced by the crack caused the block to tumble. This example is motivated by the fact that Computer-Aided Design (CAD) software sometimes produces flawed models.

4.4.2 Peg in hole

We simulate a box falling under gravity into a hole with a minimal clearance as in figures 4.8, 4.9 (box width = 9.99, hole width = 10). Physically, the box should be able to fall through until making contact with the hole’s floor.

All examples are solved using PATH \cite{FM96} using the method mentioned in previous section.
4.5 Conclusion and future works

In this chapter, I proposed a new extended contact model that can represent non-convex free space exactly unlike the single vertex-single feature contact model. The problem with traditional contact model is that it implicitly combines all active non-penetration constraints together using AND operations. Along with the linearization effect of the point-based contact model, the non-penetration system restricts the free space to be smaller than ground truth. The extended contact model relies on the topological relations of the features associate with the non-convex free space to group them together and then use complementarity theory, the mathematical background of current physical simulation constraints, to accurately select the free region outside the shape. Fortunately, the cost of the extended contact is small. In fact, the LCP matrix size of the extended contact model is always equal or lesser than the size of LCP matrix of the current state-of-the-art Stewart-Trinkle contact model.

The extended contact model requires different contact information than the single vertex-single feature model. In the next chapter, we will discuss the new collision detection that works with the new contact model.
In the future, I would like to find a method of selecting the first feature in the group to ensure solvability of the final LCP. In practice, I have yet to find a case where PATH and the new solvers described in chapter 6 failed to solve the problem.
CHAPTER 5
Collision detection in physical simulation

5.1 Introduction

In chapter 2, I have introduced the process of deriving from physical to mathematical and then to computer models of frictional rigid body system. Collision detection was introduced in the process to enumerate a list of active contacts which are then used to approximately model the rigid nature of bodies. However, collision detection deserves a much more important role in practice. In the literature, it is rare to see a work that addresses both collision detection and dynamics as equally important to the quality of simulation. In practice, as least in game physics engines, there are a number of fast and stable published methods to solve the mathematical problems from the dynamics step. The main difference between expensive commercial engines and the open-source ones is the quality of the collision detection. Currently, collision detection is built and grown through experiments, there is no clear definition now how to build the active set in many physics engines. But the importance of collision detection to the quality of physical simulation is universally accepted. In fact, collision detection and dynamics are the two main components of a physical simulation system.

Figure 5.1: Diagram of physical simulation.

Figure 5.1 shows a typical role of collision detection in physical simulation. It
takes in system positions and computes contact information. Dynamics uses this contact information to form the time stepping sub-problems, solves it to get position updates for the next time step. Then, collision detection gets the new positions and the cycle restarts.

Figure 5.2 presents a more detailed and practical diagram. It lists all the important information collision detection and dynamics modules own or share and their important roles.

---

Figure 5.2: Bullet Physics engine [Cou06] collision detection and dynamics. Each square corresponds to a small steps in Forward Dynamics and Collision Detection modules. Both modules operate on various parts of the system of rigid bodies states. There are some parts are shared between the two.

In this chapter, I start with the discussions of the problems with current collision detection and why we need a better one. Then I show how the advanced contact model described in chapter 4 helps develop a better collision detection method. At the end, numerical results are presented along with the comparisons of the old and new collision detection methods.
5.2 Problem with current contact model

As mentioned in previous chapter, current physical simulation non-penetration constraint consists of a pair of a single vertex on one body and a feature, which can be another vertex, an edge or a face, on another body. Additional information may include the outward facing contact normal and the contact distance along that normal. The non-penetration constraint then acts to prevent the projected distance along the contact normal to become negative. Normally, in one time step, there is only one set of contacts which we usually refer to as the active set exists. Choosing the right active set at a given time step is crucial to the quality of simulation. Finding minimal active set that include the correct contacts is the main role of collision detection. The most important heuristic for picking a contact is based on the close proximity of the pairs. Essentially, collision detection will have to activate a contact or at least need to consider it when the contact distance is less than a certain threshold $\epsilon \geq 0$.

![Diagram showing the particle is inside the disk centers at a vertex of the obstacle with $\epsilon$ radius.](image)

Figure 5.3: The particle is inside the disk centers at a vertex of the obstacle with $\epsilon$ radius.

In figure 5.3, the particle is in the circle centers at the vertex and radius equals to $\epsilon$, so the collision detection needs to decide whether to create a contact constraint. The reason behinds the close proximity heuristic is that we need to create a set of potential contact(s) when the two bodies are so close that they might be in penetration if we ignore them. Therefore, we have to create at least one
contact to prevent future violations. At most we can activate two contacts between the particle and the two edges. Now, we will look at four different scenarios with different outcomes.

If the collision detection decides that we only need a single potential contact between the particle and the left edge, then essentially, the non-penetration constraint associated with this contact will cut the free space of the particle to the half limited by the left edge as shown in figure 5.4. Obviously, it is not the right result.

The case of picking the right edge also results in the wrong free space as shown in figure 5.5.

If the collision detection decides that both contacts need to be active, the effective free space of the particle is even more limited to just the opposite cone of
Figure 5.6: Effective region of the obstacle (shaded) if both (particle,left edge) and (particle,right edge) contacts are active.

Figure 5.7: Effective region of the obstacle (shaded) if only the special (particle,vertex) contact is active.

Even if the collision detection decides against using any existing edge and construct the new contact from the particle and the vertex as shown in figure 5.7, the effective free space is still not correct.

Therefore, even in a very simple 2D scene involving only a particle and a two edges corner, the current contact model fails to represent the space exactly in all the cases. Now you might wonder if the contact model is that ineffective, how can current physics engines, which use the contact model, produced so many impressive results such as the one shown in figure 5.8?

The answer is that current collision detection adapts to the contact model limitation by predicting the future position and working around many corner cases
to make sure the returned active contact set is correct. Figure 5.9 shows a simple example on how to look at the particle’s current velocity to *guess* the valid contact. Usually, to make a good guess, the collision detection needs more than just geometric information of the system. Ultimately, to make sure the guesses are absolutely correct, the collision detection needs to know the future position of the bodies, which is the job of the dynamics solver. However, the solver in turn requires the current contact information returned from collision detection to set up the mathematical equations before it can solve for future position. Hence, there is a chicken-and-egg relation between collision detection and dynamics solvers. The dependence on additional information makes collision detection become complicated and deviates far from just distance computation that we had hoped for at the beginning. Current collision detection also does not scale well to new shapes as it has to deal with all the corner cases specific to the shapes. Many conditional checks in collision detection make it unfriendly to parallel computing and in stream processing and
general purpose GPUs.

![Figure 5.9: Second-order information can help estimate future position.](image)

In the following section, I will discuss the basic principles of current popular collision detection methods.

### 5.3 Collision detection methods

#### 5.3.1 Collision detection: The Method of Separating Axes (SAT)

SAT underlying basis is the Hahn-Banach separation theorem. Originally, it was used to check for intersections of convex shapes. Given an axis, it is easy to check if the projections of two shapes on that axis overlap. For checking separation, the algorithm can be described as:

**Theorem 5.3.1** *If two convex objects are not penetrating, there exists an axis for which the projections of the objects will not overlap.*

For example, figure 5.10 shows one separating axis between two bodies $C_1$ and $C_2$. Note that there is usually more than one separating axis. In contrast, figure 5.11 indicates that intersecting convex polygons have no separating axis. Algorithm 1 can be used to check for intersection of two convex polygons.

You may notice that algorithm 1 has not mentioned how to collect the list of all possible separating axes. In fact, it is impossible to collect all of them because
there are infinite number separating axes in separation cases. However, if we miss one certain direction, we may end up with a false-negative intersection.

In 2D, it is quite clear that checking all the edges’s normal of the two bodies should be enough as mentioned in [Ebe01]. Formally, given two 2D polygons $C_1$ and $C_2$ with $f_1$ and $f_2$ faces, respectively, then the maximum number of check is $f_1 + f_2$. In 3D, the set includes the normal vectors to the faces of the polyhe-
Algorithm 1: SAT algorithm to check if two convex polygons intersected.

**Input:** $C_1, C_2$: Two convex polygons

**Input:** $AxesList$: List of **ALL** possible separating axes

**Output:** True (for intersect) or False

1. **forall** the axis $a_i$ in $AxesList$ do
2. \[ p_1 = \text{project } C_1 \text{ into } a_i; \]
3. \[ p_2 = \text{project } C_2 \text{ into } a_i; \]
4. **if** $p_1$ and $p_2$ overlapped **then**
5. \[ \text{return } \text{True}; \]
6. **end**
7. **end**
8. **return** False;

dral and vectors generated by a cross product of two edges, one from each polyhedron. Thus, the maximum number of check is $f_1 + f_2 + f_1f_2$ which could be too large for many real life shapes with hundreds of faces. In reality, there are a number of optimizations that helps to reduce the size of the list. For example, in [Eri07], the practical numbers of axes with different shapes are listed below:

<table>
<thead>
<tr>
<th>3D objects</th>
<th>$C_1, f_1$ faces</th>
<th>$C_2, f_2$ faces</th>
<th>Cross edges</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Segment-Triangle</td>
<td>0</td>
<td>1</td>
<td>$1 \times 3$</td>
<td>4</td>
</tr>
<tr>
<td>Segment-OBB</td>
<td>0</td>
<td>3</td>
<td>$1 \times 3$</td>
<td>6</td>
</tr>
<tr>
<td>AABB-AABB</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>OBB-OBB</td>
<td>3</td>
<td>3</td>
<td>$3 \times 3$</td>
<td>15</td>
</tr>
<tr>
<td>Triangle-Triangle</td>
<td>1</td>
<td>1</td>
<td>$3 \times 3$</td>
<td>11</td>
</tr>
<tr>
<td>Triangle-OBB</td>
<td>1</td>
<td>3</td>
<td>$3 \times 3$</td>
<td>13</td>
</tr>
</tbody>
</table>

where AABB is the acronym for Axis Aligned Bounding Box and OBB for Object-oriented Bounding Box. Both are popular bounding shapes for broad phase collision detection, which usually the first phase in collision detection to quickly check if two bodies can be collided.

Algorithm [\[ can only be used for interference detection. It does not return any contact information, which is the main concern of collision detection. It turns out
that we can extend SAT to include contacts generation. However, it is complicated and often required special handling of corner cases. Erin Catto in his Game Developer Conference 2007 [Cat07] mentioned a number of fuzzy cases for SAT contact generation.

Figure 5.12: Hard (fuzzy 1, 2 and 3) and very hard (extreme fuzzy 1 and 2) cases for SAT contact generation (Erin Catto, Game Developer Conference 2007 [Cat07]).

Note that all the problems in figure 5.12 and most problems with SAT contact generation are in connection with the inherent flaw of the contact model. With that, collision detection is not a purely geometry check but it has to consider certain dynamical coherence factors.

SAT is fairly parallel friendly. Given the separating axes list, we only need to repeatedly examine current axis. But it may not run well on GPU as too many
checks for early exits in the code may lead to large number of branch divergence.

We can also improve SAT performance by exploiting temporal coherence in simulation. In practice, the separating axis does not change much in two consecutive frames. Even if it changes, it will only move to neighbor feature of the shapes.

5.3.2 Collision detection: The Gilbert-Johnson-Keerthi and Expanding Polytope Algorithm method (GJK/EPA)

The Gilbert-Johnson-Keerthi distance algorithm (GJK) is an iterative method for computing the distance between convex objects. The original idea developed by E. G. Gilbert, D. W. Johnson, and S. S. Keerthi for convex polyhedral in 3D [GJK88] was published in a Robotics journal in 1988. The algorithm was later extended to handle convex objects in general by Gilbert and Foo in [GF90]. An enhancement of the algorithm was also developed to compute the estimates of penetration distances when the polyhedral are intersecting [Cam97]. Gino Van den Bergen in 1999 improved the algorithm performance and accuracy in [VdB99] by using separating axis for early exit and better handling of the ill-conditioned cases.

The algorithm’s principle idea is based on the observation that two bodies are not colliding if their Minkowski Difference does not contain the origin. Thus, we can begin by generating the simplex that is closest to the origin. Then, the distance is calculated by first searching for the closest point on the Minkowski Difference to the origin, its magnitude is the (separation) distance. Note that it will only work when the two shapes are not overlapping because the origin will be inside their Minkowski Difference otherwise. Finding the penetration distance when two shapes are in penetration is harder. Cameron in [Cam97] demonstrated a step that called hill-climbing to estimate the penetration depth. The estimation is fairly accurate when the penetration is not deep. When there is deep penetration, the hill climbing problem becomes non-convex and the iterations will diverge.
Even with shallow penetration, the GJK algorithm only reports at most one contact point which is not enough for stable simulation. In [Eri05] and [Ber04], an method called Expanding Polytope Algorithm (EPA) was introduced in replacement to Cameron’s hill-climbing to report multiple contacts and to better handle deep penetrations. With GJK/EPA, we can now report multiple contact points in one call. However, it is an expensive operation so many collision detection libraries rely on frame coherence to build the contact manifold incrementally. In figure 5.14 a contact manifold between two bodies can be defined as the set of all contacts between them. It is essential for retaining contact information between different time steps.

\[
\begin{align*}
    t &= 0 \\
    t &= 1 \\
    t &= 2
\end{align*}
\]

Figure 5.13: Three frames show how we can build a good contact manifold (two contact points) between a box and the ground.(images taken from Erin Catto GDC2007 talk [Cat07]).

Given a contact with shallow penetrations, GJK/EPA can compute the contact set reliably even if it is considered to be an expensive operation. If we completely correct all the penetrations so that in the next time step, the two bodies will just be resting on each other. At that instance, GJK/EPA may not be able to generate the stable set of contacts anymore because the method switches to GJK only mode. Therefore, in practice, it is advised to leave a small penetrations to allow GJK/EPA stay in EPA mode.
Figure 5.14: In SIGGRAPH ASIA 2009 course, Jumpei Tsuda \cite{Tsu09} advised to leave a small penetrations for stable simulation.

GJK alone is good candidate to run in parallel but with EPA, the whole algorithm becomes more serialized.

5.3.3 Custom collision detection routines and other collision detection methods.

Both SAT and GJK/EPA algorithm mentioned previously are capable of generating contacts for many types of convex shapes. However, there are certain shapes which are used so frequently that they deserve their own customized collision detection. The most frequently used shape in collision detection is the 3D box. The current widely use 3D box-box collision detection algorithm was first implemented as a part of Open Dynamics Engine \cite{Smi01}. Because of the open-source nature of ODE, its box-box collision is widely used by even other physics engines. Over the time, it becomes a highly robust and optimized routine. Many other physics engines, Bullet Physics \cite{Con06}, for example, has adapted ODE’s box-box collision routine as its own. Bullet collision detection matrix shows a broad picture on the popularity of custom collision detection. By default, the entire matrix is filled with the following algorithms.

In table 5.15, "convex" represents convex polyhedron, cylinder, cone and capsule and other GJK compatible primitives. In the table, "gjk" stands for GJK/EPA
Custom collision detection routines improve performance and quality of contact generation because it takes advantage of extra knowledge of the shape’s geometries. However, it is not possible to fill the whole collision matrix with custom routines because of the large amount of work and testing associated with one routine. Furthermore, because they do not follow any particular theory, it is hard for them to reuse other implementations. One of the biggest problems with custom collision detection is they are usually not suitable to be run in parallel.

Besides SAT and GJK/EPA, there are arguably less popular collision detection methods. The LinCanny closest features algorithm [Lin93] is a feature-based collision detection algorithm that computes the distance between disjoint polyhedra. It has traditionally been considered one of the fastest methods before being outperformed by improved GJK/EPA and SAT. One important implementation, Cohen’s I-Collide collision detection package [CLMP95] uses LinCanny method to perform the low-level collision checks. The algorithm tracks and caches the closest
features between a pair of convex polyhedra. Once these features are known, the closest points between them, and therefore between the polyhedra, can be determined. The biggest problem with Lin-Canny is that when given a penetrating pair of shapes, its termination criteria will never be satisfied. Thus, it enters an infinite loop. This behavior can be avoided by forcing a maximum number of iterations but it is slow and numerically sensitive. Besides, even when it reaches termination, no penetration depth can be returned. Moreover, implementing Lin-Canny requires tweaking six numerical tolerances which can greatly affect the results.

Later, Brian Mirtich proposed numerous improvements over Lin-Canny in his Voronoi-Clip, or V-Clip, method [Mir98]. V-Clip can handle non-penetration case and at return, it reports the witness features that are in penetration. It can help to determine the penetration depths and generate contacts. V-Clip is significantly more robust than its ancestor, Lin-Canny method, because no tolerances tweaking is required. Also, it is simpler to implement than Lin-Canny. V-Clip was used mainly in Brian Mirtich thesis [Mir96] as a part of his impulse based physics engine. Today, modern GJK/EPA and SAT are still better choices in collision detection due to their simplicity, performance and the ability to generate reliable contacts.

The last notable collision detection method I would like to discuss in this chapter is Proximity query using Swept Sphere volume (PQP) [GLM96] and [LGLM00]. PQP builds a Bounding Volume Hierarchy (BVH) of the shapes using swept spheres to help quickly identify the closest features during query. A heavily modified of PQP was used in early version of dVC2d [BNT07]. Most of the modifications were to generate penetration distances and output multiple contacts.
5.4 Collision detection for non-convex contact model

5.4.1 Basic rules for the new collision detection

In previous sections, I have elucidated many problems with the state-of-the-art physical simulation involving collision detections. In particular, the failure to accurately model the configuration space forces the collision detection to make guesses about the future state in order to generate valid contacts. In chapter 4, a new advanced contact that can model non-convex free space around a shape was introduced. In this section, we will develop a novel collision detection method for this contact model. Because the new contact model can fully represent the free space, the new collision detection should only need geometric information and should not have to guess the future states of the bodies. Determining which contact will indeed be active in the next step is now the job of the new solver. It is a good direction because the solver has all the needed information (important geometric proximities from the collision and all velocities, acceleration and external forces of all the bodies) and control (its job is to find a new position for all bodies) to solve the problems.

\[
\max(d_1, d_2) \geq 0
\]

Figure 5.16: The new contact proposed in chapter 4 can accurately model non-convex polyhedral free space.

The two main problems with current collision detection mentioned in previous section are: collision detection is required to make guesses on the bodies future
position and the implementations are too complicated and not parallel friendly. The new collision detection method I propose should learn from and fix these problems. In order to do so, there are two rules or guidelines that it has to follow:

- **Function of current state.** The new collision detection is a pure function of the system state at the current time. The only type of inputs that can affect the outcome of the simulation is the position. Other state variables such as velocity, acceleration or external forces may be used to compute certain thresholds for further optimization. This rule forces collision detection to be a pure proximity query module.

- **Simple computation.** The new collision detection should only consist of simple functions that can easily be run in the parallel and modern stream processing GPUs. With this rule, it will be able to take advantage of recent developments in computer hardware technologies.

### 5.4.2 Extended contact representation

The output of any collision detection routine has to include a list of contacts, so I start by reintroducing the new contact representation. Traditionally, a contact holds information between two features of two different bodies. The new advanced contact can accurately model non-convex region because it groups related distance constraints together. Thus, we need an extended contact representation that was described in [4.2.1](#).

For clarity, we focus on 2D case in this section. Most ideas presented here are applicable to 3D and all other 3D specific details will be discussed in later section.

In 2D, a contact representation of the new non-convex contact model contains:

- A vertex $p$ of the first body.

- A group of edges $e_1, \ldots, e_n$ of the second body.
• An assumption that the normal impulse, if it exists, will only act along the normal of \( e_1 \).

The extended contact does not include the normally problematic vertex-vertex case. All the constrained distances are between a vertex and an edge. Under extended contact model, the vertex-vertex case has become redundant because we can now model the surrounding space exactly using vertex-edge pairs.

The third statement indicates that we focus on the new contact model with predefined acting normal direction as mentioned in section 4.2.2.3. The effect of having a predefined acting normal to the contact model and a good heuristic to choose one in the list are discussed in later section.

Before going into implementation details of the new collision detection, we examine some specific examples of the extended contact representation.

![Figure 5.17: Example of extended contact: Particle vs Triangle.](image)

In figure 5.17 a particle \( P \) is close enough to two edges \( e_1 \) (negative distance) and \( e_2 \). The only extended contact that the collision detection algorithm needs to return in this case is \( (P, [e_2, e_1]) \). \( e_2 \) is chosen to be the acting edge because it has the smallest positive distance to \( P \). In comparison, the active contact list of Stewart-Trinkle formulation in this case is \([ (P, e_1), (P, e_2) ] \).
Figure 5.18: Example of extended contact: Triangle vs Rectangle.

The active extended contact list in figure 5.18 is \(((P_2, [e_2, e_1]), (P_3, [e_3]), (P_1, [e_3, e_4]))\).

Normal collision detection for Stewart-Trinkle should return a list of 5 contacts. Note that, the epsilon circle, other vertices and edges were left out for brevity. For extended contact \((P_1, [e_3, e_4])\), we can pick \(e_4\) as the acting edge because the distances between \(P_1\) to both \(e_3\) and \(e_4\) are the same (the segment, not the line or ray containing the edge).

Figure 5.19: Example of extended contact: Triangle vs Quadrilateral with penetration.

Figure 5.19 shows a very interesting case. Based on the position of the \(\epsilon\)-circle around \(P_1\), the only active contact is \((P_1, [e_1, e_2])\) with two negative distances. We also notice that the contact \((P_1, [e_1])\) will have exactly the same effect but half the cost (only one complementarity condition instead of two).

Now considering the case of figure 5.20, if we follow the implied heuristic
that only includes the pair with absolute value of distance less than $\epsilon$, there is no contact returned. Thus, we will need to handle the in penetration case differently. Fortunately, it can be detected easily (when all distances < 0) so it can easily be checked for in implementation. Once already in penetration, a possible heuristic is to pick the edge with the smallest absolute distance or the largest signed distance as the acting edge. We can also pick the acting edge based on relative velocity of the two.

Figure 5.20: Example of extended contact: Triangle vs Quadrilateral with deep penetration.

5.5 Implementation of new collision detection

There are certain assumptions I want to make. First, we will focus on the narrow-phase, which is used to generate contacts data between two bodies. In practice, given a scene of $n$ bodies, we usually employ a separate broad-phase algorithm to avoid $O(n^2)$ body-body checks. There are a number of broad-phase approaches described and implemented in [Cou06], but they are not the focus of this section so I will not mention them here. We also assume that all bodies are polyhedral. Extension to implicit shapes is possible because the method itself operates on distances and we should be able to construct the same set of distances for implicit shapes. However, I will leave it as future work.

Function \texttt{CollideBodies} is the main entry point of the collision detection. It
Function CollideBodies(body₁, body₂)

Data: Contacts: a global list of active contacts.
input : Two collidable bodies: body₁ and body₂.
output: Active contact list in Contacts.

1 // Collide all vertices of body₁ to edges of body₂
2 foreach vertex v₁ of body₁ do
3    CollideVertexBody(v₁, body₂)
4 end
5 // Collide all vertices of body₂ to edges of body₁
6 foreach vertex v₂ of body₂ do
7    CollideVertexBody(v₂, body₁)
8 end

check for collision of all the vertices of the first body to the second body and all the
vertices of the second one to the first one through function CollideVertexBody.
Note that the collision detection defines a number of global variables: Contacts is
the active contact list and \( \epsilon^+ \) as the upper bound of the active distances. The role
of \( \epsilon^+ \) is quite clear: any pairs of vertex-edge (segment) with distances greater than
\( \epsilon^+ \) will not be included in the active list. A negative number \( \epsilon^- \) can be used as the
lower bound for the signed distances to reduce the number of edges in the set of
active contact.

Function distanceVertexSegment(v, segment)

input : A vertex \( P \) and an edge \( e \) defined by two points \( P₀, P₁ \).
output: Euclidean distance between \( v \) and the line segment \( P₀, P₁ \).

1 \( v = P₁ - P₀; \)
2 \( w = P - P₀; \)
3 if \( (c₁ = w \cdot v) \leq 0 \) then
4    return distance \( (P, P₀); \)
5 end
6 if \( (c₂ = v \cdot v) \leq c₁ \) then
7    return distance \( (P, P₁); \)
8 end
9 return |distanceVertexLine \( (P, e)]]; |

Function CollideVertexBody \((v, b)\) generates at most one extended contact
Function distance($P_0$,$P_1$)

input : Two vertex $P_0 = (P_0.x, P_0.y)$ and $P_1 = (P_1.x, P_1.y)$.
output: Euclidean distance between $P_0$ and $P_1$.

1 return $\sqrt{(P_0.x - P_1.x)^2 + (P_0.y - P_1.y)^2}$;

Function distanceVertexLine($v$,$e$)

input : A vertex $P = (P.x, P.y)$ and an edge $e$.
output: Signed distance between $v$ and the line contains $e$.

1 Assume $n = [n_x, n_y]$ is the unit normal of $e$ and $c$ is the third implicit parameter of the line that contains $e$;
2 return $n_xP.x + n_yP.y + c$;

between vertex $v$ and all edges of body $b$. It is the most important function of the collision detection. Therefore we will walk through the function to understand its important steps. CollideVertexBody has three auxiliary functions: distanceVertexLine, distanceVertexSegment and distance. Function distance simply computes Euclidean distance between two vertices. Function distanceVertexLine relies on the implicit equation to compute the vertex’s signed distance to the line.

Function distanceVertexSegment logic is based on figure 5.21 and [Ebe08a]. The implementation delays the most expensive operation, the division, until needed.

Lines 5 to 10 compute distances from vertex $v$ to all the infinite lines that support the edges of body $b$. It computes the maximum distance and stores in the maxDistance variable. Line 11 compares maxDistance to 0. The result of maxDist < 0 indicates that all the distances are negative, which means the vertex $v$ is inside body $i$. Hence, we apply our penetration correction heuristic by creating a contact between $v$ and the closest edge of $b$. For non-penetration case, we start by calculating Euclidean distances between $v$ and all the edges (segments) of $b$ by calling distanceVertexSegment. Note that these Euclidean distances are used to find the edges in the active contact because the distance to an infinite line is not a good indication of close proximity (figure 5.22).
Function CollideVertexBody(v, b)

Data: Contacts active contact list, $\epsilon^-, \epsilon^+ =$distances lower, upper bound.

input : A body $b$ and a vertex $v$ of a different body.

output: Add at most one extended contact to Contacts.

// initialization
listEdges $\rightarrow$ all of $b$ edges;
set all lineDistances to 0;
maxDistanceIndex $\leftarrow$ 0
maxDistance $\leftarrow$ listEdges[0];

foreach edge $e_i$ in listEdges do
  lineDistances[i] $=$ distanceVertexLine($v, e_i$);  
  if maxDistance $<$ lineDistances[i] then
    maxDistance $\leftarrow$ lineDistances[i] maxDistanceIndex $\leftarrow$ i;
  end
end

if maxDistance $<$ 0 then // Penetration
  add $(v, [listEdges(maxDistanceIndex)])$ to Contacts;
  return;
else // No penetration
  set all segmentDistances to 0;
  activeIndices $\leftarrow []$;
  foreach edge $e_i$ in listEdges do
    segmentDistances[i] $=$ distanceVertexSegment($v, e_i$);  
    end
  activeIndices $\leftarrow \{i \mid (lineDistances[i] > \epsilon^-) \land (segmentDistances[i] < \epsilon^+)\}$;

  // activeIndices now holds the active contact list
  if activeIndices is empty then
    return; // No contact created
  end
else
  actingIndex $\leftarrow$ index that minimizes segmentDistances over the list of positive lineDistances;
  Create and add extended contact from vertex $v$, activeIndices edges, and acting edge actingIndex to Contacts;
  return;
end
end
$w_0 = P - P_0 \text{ and } \theta_0 \in [-180^\circ, 180^\circ]$  

$w_0 \cdot v \leq 0$

$\Leftrightarrow |\theta_0| \geq 90^\circ$ is obtuse

$\Leftrightarrow d(P, S) = d(P, P_0)$

$w_1 = P - P_1 \text{ and } \theta_1 \in [-180^\circ, 180^\circ]$  

$w_1 \cdot v \geq 0 \Leftrightarrow w_0 \cdot v \geq v \cdot v$

$\Leftrightarrow |\theta_1| \leq 90^\circ$ is acute

$\Leftrightarrow d(P, S) = d(P, P_1)$

Figure 5.21: Two cases when calculating distances of vertex $P$ to the segment defined by $P_0, P_1$. $d$ is the distance between two vertices.

Figure 5.22: Distance from $P$ to edge $e$ is 0 but the vertex is not close to the rectangle.

Once we have the list of distances to the segments, we can use it to find the list of edges that should be included in the contact. Two simple proximity checks for active edges can be used. First, they have to be close geometrically to the vertex, verified by the condition $segmentDistances[i] < \epsilon^+$. For now, $\epsilon^+$ is one globally defined positive constant but in implementation, having a body’s level $\epsilon^+$ is desirable. Moreover, making $\epsilon^+$ depend on velocity is a possible heuristic. It is important to note that this $\epsilon^+$ check is purely for performance. In fact, the two runs, one with practically infinite $\epsilon^+$ and one with reasonable $\epsilon^+ = 0.1$, have the same outcomes as shown in figure 5.23. They are identical, even though in 5.23a PATH
uses more computation time to solve in average as the contacts between all vertices and edges of the fixed triangle and the rectangle are always active. In practice, we should always use a reasonable $\epsilon^+$ instead of infinite because the size of the input problem would grow too large. The second check $lineDistances[i] > \epsilon^-$ is also for performance but it is not as important as the first check because it already filters out any edge $i$ with $lineDistances[i] < -\epsilon^+$.

The list of edges that pass two previous checks will be included in the extended contact with vertex $v$. Obviously, if there is no such edge, we can assume that vertex $v$ and body $b$ are too far apart that we they will not interact in the next time step. If the list is not empty, then we still have one more task to complete: finding the acting edge. Now assume we have a way to pick the acting edge, we can construct the extended contact from $v$, the active list and the acting edge. That concludes function $CollideVertexBody$.

Below is the short of summary $CollideVertexBody$:

<table>
<thead>
<tr>
<th>Function</th>
<th>$CollideVertexBody(v,b)$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data:</strong></td>
<td>$Contacts$ active contact list, $\epsilon^-,\epsilon^+$ = distances lower, upper bound.</td>
</tr>
<tr>
<td><strong>input:</strong></td>
<td>A body $b$ and a vertex $v$ of a different body.</td>
</tr>
<tr>
<td><strong>output:</strong></td>
<td>Add at most one extended contact to $Contacts$.</td>
</tr>
<tr>
<td>1</td>
<td>Compute distances of $v$ to all the lines support body $b$’s edges;</td>
</tr>
<tr>
<td>2</td>
<td><strong>if</strong> max distances less than 0 <strong>then</strong></td>
</tr>
<tr>
<td>3</td>
<td>Use in penetration heuristic to create a contact then return;</td>
</tr>
<tr>
<td>4</td>
<td><strong>end</strong></td>
</tr>
<tr>
<td>5</td>
<td><strong>else</strong></td>
</tr>
<tr>
<td>6</td>
<td>Compute distances from $v$ to all the edges (the actual segments);</td>
</tr>
<tr>
<td>7</td>
<td>Filter out edges with distances greater than $\epsilon^+$;</td>
</tr>
<tr>
<td>8</td>
<td>We can have an extra filter step that remove edges with line distances less than $\epsilon^-$;</td>
</tr>
<tr>
<td>9</td>
<td>Find acting edge as the one with minimum segment distances in those with positive line distances.</td>
</tr>
<tr>
<td>10</td>
<td><strong>end</strong></td>
</tr>
</tbody>
</table>
5.5.1 Acting edge in extended contact

By using extended contact, we have grouped a list of edges together and prevent the contact’s vertex from penetration the group by using one normal impulse (or force). In previous contact models, one pair of vertex-edge has one contact. Thus, the choice of the reaction impulse normal for a contact is always its only edge’s normal. However, extended contacts may contain more than one edge so it is not immediately clear how to choose the contact’s normal. Next, we will consider the choices of acting edge in three different contact modes: penetration, touching and not touching.

In penetration mode, the new collision detection algorithm will return an the one with only one edge. So the choice of acting edge is obvious.

When in touching mode, our physical intuition tells us that the right choice for acting edge is the one which is currently touching the vertex. That basically means we should choose the edge whose distance (segment) to the vertex equal to 0. Failing to do so will result in unexpected behavior as shown in figure 5.24. In figure 5.24a, when the box starts to touch the ramp, there is only one edge in the extended contact, thus it behaves as expected. However, when it gets to the end of the ramp, there are two edges in the list and it picks the horizontal edge as the acting one. Hence, the normal impulse shoots it back to the ramp. In figure 5.24b the heuristic to minimize the segment distance will find the correct acting edge which has the distance equal to 0.

When the bodies are not in touching mode, it is not that clear which edge should be acting anymore. However, figure 5.25 shows that the best possible pick using our physical intuition for the acting edge in this case is $e_2$. Algorithmically, $e_2$ has the lowest segment distance to $P$. Thus, our heuristic for choosing acting edge works in the non-touching case also.
However, picking acting edge is in non-touching mode is in fact a guessing game. We can pick the best possible candidate but we do not have access to the future positions to be sure it will be correct. Come back to our example in figure 5.25, let see how our heuristic affect the simulation with different future position of $P$.

In figure 5.26, our heuristic chooses $e_2$ as the acting edge. But if $P'$ is the future position of $P$, the correct acting edge should be $e_1$ because $P$ penetrates $e_1$ first. With $e_1$ as acting edge, the penetration-free position of $P$ in simulation is $P_1$, however, in our heuristic, it will be $P_2$. The same applies for the case where the future position of $P$ is $P''$. Thus, correct position is $P_3$, our heuristic position ends up at $P_4$. As we can see in the picture, the error is quite small. Also, the chance for $P$ to hit $e_1$ or $e_3$ is relatively low. If $P$ penetrates the body through $e_2$ (our heuristic pick) or does not penetrate at all (the choice of acting does not matter) then the heuristic produces the expected simulation result.

In conclusion, the proposed heuristic to pick acting edge that minimizes segment distance to the vertex in extended contact works well in most of the cases.

### 5.5.2 Negative bound $\epsilon^-$ in extended contact

The negative bound $\epsilon^-$ was used in a check to filter out edges to be included in an extended contact in function $\text{CollideVertexBody}$. We already mentioned that the check $\text{lineDistances}[i] > \epsilon^-$ will be redundant after we checked for $\text{segmentDistances}[i] < \epsilon^+$ if $\epsilon^+ > |\epsilon^-|$. We can show that by using the fact that $\text{segmentDistances}[i] \geq |\text{lineDistances}[i]|, \forall i$. Thus, we need to choose $\epsilon^-$ in the range $[0, \epsilon^+]$.

Figure 5.27 shows the reason we may want to have $\epsilon^- < \epsilon^+$. In the figure, $P$ is close to both of $e_1$ and $e_2$, but the distance to $e_1$ is positive and to $e_2$ is negative. We definitely want to include $e_1$ in our extended contact because the chance $P$ interacts
with $e_1$ is high. However, in order to have $P$ interact with $e_2$, assuming the box is fixed, it will need to move to the right, then down, then move once more to the left all in one time step. If our time step is not too big, the chance of all three to happen is very low. Even if the box is rotating, the chance for $P$ to touch $e_1$ still appears to be higher. That is the reason we may choose $\epsilon^- < \epsilon^+$ to filter out more edges to improve performance.

On the other extreme, if the all three movements of $P$ in one time step are so unlikely, should we set $\epsilon^-$ to 0? Figure 5.28 shows the ghosting positions of two bodies in 200 frames, the bodies are displayed every 20 frames with increasing alpha value. We can easily see that the two results are different. In figure 5.28b, the omission of negative distance edge causes a slight push right after the lower right corner of the box touches the upper vertex of the static triangle. The reason for the push lies on the transition of the extended contact from one having to two edges. The abrupt introduction of an zero distance edge causes the push. As the result of that, it is advisable to have $\epsilon^-$ strictly $< 0$.

In conclusion, $\epsilon^-$ should be chosen in $(-\epsilon^+, 0)$ range. In my experience, having $\epsilon^-$ to be around 0.05 — 0.1 is usually enough.

5.6 Extension of new collision detection in 3D

Both the previous and the new extended contact models are point-based. In other words, they maintain non-penetration by creating constraints over the signed distances from a set of points along certain normals. In previous contact models, one such distance creates one contact constraint. In the new extended contact model, they are grouped together if they are from the same bodies and share the contact vertex. So given a set of distances, the step that converts these distances to contact data in 2D and 3D should be similar.
In 2D, non-penetration constraint between two bodies can be maintained by creating contacts from the first and second body vertices to the second and first edges, respectively. However, this reasoning does not work in the case shown in figure 5.29 where obviously even there is no vertex penetrating any bodies, penetration still happens. However, this is the problem with point-based contact models. The only way to handle this case is to use other contact models such as volume-based that are usually prohibitively expensive for practical usages. Moreover, even using point-based contact, if the initial state is valid and the time step is reasonably small so that after one time step, no pairs of bodies can move from touching or separating to the invalid configuration as shown in figure 5.29, this problem will not happen.

The same simple contact constraints system can be extended from 2D to 3D by replacing edges with faces. Therefore, we need a constraint system of distances between vertex of one body to the faces of the other for all the pairs of bodies in the system. However, vertex-face constraints cannot prevent edge-edge penetrations as illustrated in figure 5.30.

In 2D, edge-edge intersection is prevented by vertex-edge contact constraint. In 3D, figure 5.30 shows that vertex-face contact constraints cannot prevent edge-edge intersections. Thus, 3D edge-edge contact constraints are needed. We can generate 3D edge-edge contacts from the result of segment-segment distance computations.

Figure 5.31 shows three different results of 3D segment-segment distance computations. The new vertices will then be added to the respective body’s vertex list and treated the same as original ones. Thus, in 3D, the new collision detection that works with the advanced contact model has one more step to find new edge-edge case vertices before the step of vertex-face distance computations similar to the vertex-edge distance computation of the 2D version.

An efficient and simple implementation of 3D segment-segment is described in [Ebe08b]. The basic idea is to turn the distance computation problem into minimiza-
tion of a continuous differentiable quadratic function over a certain range. Then, we only need to check at most three positions: the two boundaries and the local minimum where the function gradient is zero.

5.7 Conclusion and future works

In this chapter, I have described the foundations and implementation details of a new collision detection method for the advanced contact model proposed in chapter 4. The new collision detection consists of very simple, embarrassingly parallel functions over the current states of the simulation system. Thus, they can be efficiently run on modern Stream Processing and General Purpose GPU. Unlike current collision detection methods, the new method is simple to implement.

The new collision detection expected outcome is well defined for all polygonal inputs. Given a pair of polygonal shapes, the expected result is unambiguous and only depends on a set of geometric based thresholds that are mostly used for efficiency. It does not have to guess the bodies’s future positions in order to generate the active contact set like normal collision detection. The future states are only useful in choosing the acting feature of the advanced contact. And the proposed heuristic should work in most of the cases. Even when it picked the wrong feature, the simulation error is small.

The collision detection method proposed in this chapter shares the same input with graphics rendering pipeline (vertices and faces). So when running on a GPU, it will not require any extra data structures and memory bandwidth. In the future, I hope to implement a 3D version of the method in GPU’s programming language like CUDA or OpenCL.

The advanced contact model and this new collision detection method are not complete. They mostly open a new ground for a different approach to contact mod-
eling. So many works are still needed in the future. One crucial item in the future work is to extend this framework to handle implicit shapes like sphere, cylinder or cones. As mentioned earlier, the method is based on distances. Therefore, we should be able to extend it to implicit shapes. Another important future work is to implement a 3D version of the collision detection. Due to time constraint, I only implemented a 2D version of the method in Matlab.
Figure 5.23: 200 frames of two simulations with two different $\epsilon^+$. Initially, both the box and the triangle are dropped from above. The triangle, which is lower, hits the ground first and then collide with the falling box. From there, the two shapes are separated and move to two different walls.
(a) Acting edge is always the first edge found. In this case, the acting edge was the horizontal one. That explains why the box jumped up when getting close to the right vertex of the triangular obstacle.

(b) Acting edge is chosen to minimize (segment) distance to the vertex. The box can slide down naturally as expected.

Figure 5.24: 150 frames of two simulations with two different heuristic of choosing acting edge.

Figure 5.25: Distances from vertex $P$ to three edges $e_1, e_2, e_3$ in non-touching mode of extended contact.
Figure 5.26: Possible future position in simulation of $P$ with different acting edge.

Figure 5.27: A particle is moving close to the box’s vertex.

Figure 5.28: 200 frames of two simulations with two different $\epsilon^-$. You can see that in the case where $\epsilon^- = 0$, the slight push moves the box a little farther to the left than the case with $\epsilon^- \neq 0$. 
Figure 5.29: The two shapes are intersected even when there is no vertex of them inside the other shape.

Figure 5.30: Edge-Edge intersection in 3D.
Figure 5.31: Cases of different number of new vertices generated to handle 3D edge-edge contacts.
CHAPTER 6
Solution methods for physical simulation problems

6.1 Introduction

In physical simulation, after running collision detection to compute the active contacts set and formulating the time stepping problem \(2.3.3\), the result is a mixed complementarity problem (MLCP) \((2.74), (3.5)\). There are a number of methods to find the exact or approximate solutions of the MLCP. Most gaming physics engines use iterative type solvers to achieve real time performance. For example, Bullet Physics engine and Box2D use Sequential Impulse solver [Cat05] which basically is a more specialized version of Gauss-Seidel method for LCP [Mur88]. Sequential Impulse solver can only be applied to convex LCPs. Therefore, it cannot be used to solve the MLCP from Stewart-Trinkle method with linearized Coulomb friction model. Mihai Anitescu in [AH04b] proposed a convex friction model and the problem final MLCP can be solved efficiently using convex quadratic programming solvers.

Another class of solvers attempt to compute the approximate solution of the original non-convex problem by alternating between finding the solutions of two convex sub-problems [MS01, SM04, KEP05, KSJP08]. Many properties of these methods have been discussed throughout the thesis. In this section, I will solely focus on solution methods for the non-convex MLCP from Stewart-Trinkle time stepping method and the new formulation with advanced contact model described in chapter 4. To refresh our memory, the mixed complementarity problem of physical simulation can be written as:
\[ M\nu^{\ell+1} = M\nu^{\ell} + W_n\nu^{\ell+1} + W_f\nu^{\ell+1} + p_{app}^{\ell} + p_{\nu}^{\ell} \quad (6.1) \]

\[ 0 \leq p_n^{\ell+1} \perp W_n^{T}\nu^{\ell+1} + \frac{\Psi_n^{\ell}}{h} + \frac{\partial \Psi_n^{\ell}}{\partial t} \geq 0 \quad (6.2) \]

\[ 0 \leq p_f^{\ell+1} \perp W_f^{T}\nu^{\ell+1} + E\sigma^{\ell+1} \geq 0 \quad (6.3) \]

\[ 0 \leq \sigma^{\ell+1} \perp U_p^{\ell+1} - E^T p_f^{\ell+1} \geq 0 \quad (6.4) \]

Currently, there are two known solvers for the MLCP. They are Lemke [CPS92] and Path library [FM96]. In practice, Lemke, a pivotal based method, can only be used on small simulation problems. The PATH solver applies techniques similar to those used in Newton’s methods for smooth systems to a non-smooth reformulation of the MLCP. The algorithm consists of a sequence of major iterations, each consisting of an approximation or linearization step (sometimes called successive LCP), the construction of a path to the Newton point (the solution to the approximation), and a possible search of this path. When the Newton point does not exist or the path cannot be entirely constructed, a step along the partially computed path is taken before the problem is re-linearized. A non-monotone watchdog strategy is employed in applying the path search; this helps to avoid convergence to local minima of the norm function for the underlying non-smooth equation and keep the number of function evaluations required as small as possible. It was actively developed for more than 15 years and currently is the state-of-the-art in solving complementarity problems.

In this chapter, I describe two new approaches to solve the physical simulation MLCP. The first one is based on Levenberg-Marquardt, which was a standard technique for nonlinear least-squares problems but recent studies showed that it can be applied to MLCP [Kan04] and [Pet06]. It can be thought of as a combination
of the steepest descent and the Gauss-Newton method but the method has fast local convergence unlike fixed point approaches. The other method was inspired by [MS01, SM04, KEP05, KSJP08]. It oscillates between solving two smaller convex quadratic programming problems to find the solution for the original MLCP. Instead of developing its own solver, the method makes use of many available convex quadratic programming solvers.

6.2 Levenberg-Marquardt based solution methods

Many previous algorithms to find the solution of the complementarity problem are based on a reformulation as a non-smooth nonlinear system of equations. With the development of non-smooth Newton methods [PQ93], [Qi93] and [QS93], finding such reformulations has attracted considerable attention. An early approach by Pang [Pan90] is based on the minimum function, which belongs to the class of so called NCP functions. Combined with a line search strategy, this approach, which is based on the B-derivative of the reformulation operator, achieves global convergence under certain relatively strong regularity conditions. Another approach using other NCP functions such as the popularly used Fischer-Burmeister function [Fis92] leads to more desirable reformulations where the natural merit function is continuously differentiable. Under further conditions, this property facilitates the use of line search strategies for obtaining global convergence. This approach has been extensively studied, improved and extended (more references in [Pet06]). It brings semi-smooth Newton methods to a high level of reliability and efficiency.

Interestingly, reformulations using smooth NCP functions [SQ99] can take advantage of standard Newton but they only lead to fast local convergence with non-degenerate condition in addition to usual non-singularity and Lipschitz continuity conditions of usual Newton methods [Pet06]. In contrast, non-smooth NCP func-
tions may require a more restricted non-smooth Newton method but can lead to fast local convergence even in the case the problems under consideration are degenerate.

In this section, I report my experiences with two different Levenberg-Marquardt based solvers of the reformulations of the MLCP using Fischer-Burmeister NCP function. The first solver simply employs the Fischer-Burmeister NCP-function to convert the original MLCP into a non-smooth system of nonlinear equations. Then, a Levenberg-Marquardt based method [Fis95] along with simple trust-region scheme is used to solve the resulting system. My implementation is based on Yuval Tassa’s Matlab function [Tas08].

The second approach uses another NCP function along with Fischer-Burmeister. Even it leads to an over-constrained system of nonlinear equations, it avoids certain drawback with the first approach. The method also has a very desirable property to achieve global convergence which is its natural merit function is continuously differentiable.

6.2.1 Simple semi-smooth solver

The method starts with reformulation of the MLCP as a system of nonlinear equations $G(x) = 0$ through a NCP-function.

**Definition** A function $\phi : \mathbb{R}^2 \to \mathbb{R}$ having the property that

$$\phi(a, b) = 0 \iff a \geq 0, b \geq 0, ab = 0$$

(6.5)

is called an NCP-function.
Let \( F : \mathbb{R}^n \to \mathbb{R}^n \) be given, \( \phi : \mathbb{R}^2 \to \mathbb{R} \) be any NCP-function and we define \( \Phi : \mathbb{R}^n \to \mathbb{R}^n \) as follows:

\[
\Phi(x) = \begin{pmatrix}
\phi(x_1, F_1(X)) \\
\vdots \\
\phi(x_n, F_n(X))
\end{pmatrix}
\]  

(6.6)

then \( x^* \) solves the MLCP(\( F \)) if and only if \( x^* \) solves \( \Phi(x) = 0 \).

In the case of physical simulation, \( F \) is continuous and differentiable. If the NCP function \( \phi_M = f(|a-b|) - f(a) - f(b) \) where \( f : \mathbb{R} \to \mathbb{R} \) is any strictly increasing function with \( f(0) = 0 \) is used, then \( \Phi(x) = 0 \) is smooth and any Newton-type method can be used without modification \cite{Man76}. But Newton method does not work well when strict complementarity is violated at the solution. That is the reason why modern approaches prefer semi-smooth reformulations. The most famous semi-smooth NCP-function was first proposed in a paper by Fisher \cite{Fis92}, where it was also attributed to Burmeister:

\[
\phi_{FB} = \sqrt{a^2 + b^2} - a - b
\]  

(6.7)

The Levenberg-Marquardt method is a variant of semi-smooth Newton’s method for solving \( G(x) = 0 \). The algorithm is described as follows:

<table>
<thead>
<tr>
<th>Algorithm 2: Non-smooth Levenberg-Marquardt Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Choose ( x^0 \in \mathbb{R}^n ). Set ( k = 0 ).</td>
</tr>
<tr>
<td>2. If ( x^k ) satisfies termination conditions: STOP.</td>
</tr>
<tr>
<td>3. Choose ( H_k \in ) the ( B )-subdifferential of ( G(x_k) ) and find a solution ( d^k \in \mathbb{R}^n ) of the linear system:</td>
</tr>
</tbody>
</table>
| \[
(H^T_k H_k + \nu_k I) d = -H^T_k G(x_k),
\]  

(6.8)

where \( \nu_k \geq 0 \) is a chosen regularization parameter. |
| 4. Set \( x^{k+1} = x^k + d, k \rightarrow k + 1 \), and go to step 1. |
The algorithm looks similar to semi-smooth Newton’s method with the only notable difference in equation (6.8) where in Newton-based, it is $H_k d = -G(x^k)$. Solving this system has several advantages. First, its matrix is symmetric and positive definite if $\nu_k > 0$. Second, if the iterates are not sufficiently close to the (regular) solution, then $H_k$ can be a singular matrix, resulting into an ill-conditioned linear system. This problem can be avoided by an appropriate choice of $\nu_k$ which is discussed in details in \cite{Pet06}.

It can be shown that algorithm 2 has very strong local convergence property under mild regularity conditions. Interested readers can find more information in Stephania Petra’s PhD thesis \cite{Pet06}.

To achieve a global solution, it is important to provide a measure for monitoring the convergence of the iterates in our method of choice toward a desired solution. Merit functions provide such a measure and are formally defined as follows:

**Definition** A function $\Psi : \mathbb{R}^n \to \mathbb{R}$ is called a merit function for the mixed complementarity problem $MCP(F, l, u)$ if it has the following properties:

- $\Psi(x) \geq 0$, $\forall x \in \mathbb{R}^n$.
- $\Psi(x) = 0 \iff x$ solves $MCP(F, l, u)$.

The above definition admits a large class of functions; to be useful in practice a merit function should satisfy further properties, such as being continuously differentiable. In such a case it is easy to enforce the global convergence of algorithms by using the gradient of the merit function.

With proposed algorithm 2, we can choose the natural merit function as:

$$\Psi(x) = \frac{1}{2} \Phi(x)^T \Phi(x) = \frac{1}{2} \|\Phi(x)\|^2.$$ (6.9)
The natural merit function is continuously differentiable (despite the fact that the underlying equation involved non-smooth function $\phi_{FB}$), a very desirable property to enforce global convergence of the algorithms by using the gradient of the merit function. This is a major advantage of using Fischer-Burmeister NCP-function over others.

I have implemented algorithm 2 together with simple trust region method (algorithm 3.28 in [Pet06]) in the newest version of dVC2d Matlab. It works quite well in practice. The most important component of algorithm 2 is the sparse linear system solver for equation (6.8). The list of notable sparse linear system solver libraries can be found in the following section. The study on the effects of using different linear solvers on the algorithm is left to future work.

### 6.2.2 Over-constrained semi-smooth solver

![Figure 6.1: The value of $\phi_{FB}(a, b)$ function is not a good indication of the complementarity gap $\max(a, 0) \max(b, 0)$ in the first orthant.](image)

The Fischer-Burmeister function $\phi_{FB}(a, b)$ is very effective in obtaining feasi-
bility, since it becomes relatively large if $a$ or $b$ is significantly negative. However, it has some difficulties in reducing the complementarity gap, since $\sqrt{a^2 + b^2} - a - b$ is quite flat at on the positive orthant (see figure 6.1). For example, if $a = 1000, b = 0.1$, then complementarity gap, the product $ab = 100$, which is a large number while $\phi_{FB}(a, b) = \sqrt{1000^2 + 0.1^2} - 1000 - 0.1 = -0.099995$ is very small. In this section, I describe an approach proposed by Petra et.al which overcomes these difficulties but retains the advantages of the Fischer-Burmeister function. To achieve that, the nonlinear systems reformulation described in previous section by was modified by appending additional equations that incorporate the product terms:

$$\Phi(x) = \begin{pmatrix} \vdots \\ \lambda_1\phi_{FB}(x_i, F_i(x)) \\ \vdots \\ \lambda_2\phi_+(x_i, F_i(x)) \\ \vdots \end{pmatrix}$$

(6.10)

with $\phi_+(a, b) = \max(a, 0) \max(b, 0)$ and $\lambda_1, \lambda_2 \in \mathbb{R} \setminus \{0\}$ are the weight constants. In practice, I set the values of them to default values that are recommended by Petra and Kanzow where $\lambda_1 = .9, \lambda_2 = .1$. The additional function $\phi_+(a, b) = 0$ in all other three orthants where $\phi_{FB}$ is efficient and grows very large with complementarity gap to help the overall system reaches global convergence faster. Besides keeping the same desired properties of just using Fischer-Burmeister NCP-function, this method has the following advantages:

- Faster reduction of the complementarity gap $x^T F(x)$.
- A Newton-type search direction can be taken in full at each iteration because no gradient steps are necessary in order to get global convergence.
• Numerically more robust.

However, this method leads to a least-squares reformulation of the nonlinear complementarity problem. To solve the arising over-constrained system of equations, Petra in [Pet06] proposes a semi-smooth Levenberg-Marquardt method equipped with line search. The global and local fast convergence results (under mild assumptions) are similar to some existing equation-based methods.

This solver is also implemented as a part of dVC2d Matlab. The solver is based on LMMCP library [PK06], which based upon Petra’s work in her thesis.

In both solvers, majority of the computational time is spent on computing the search direction, which actually is solving a sparse system of linear equations. In addition to that, the ability to avoid singularity and degeneracy of linear system solvers are crucial to both algorithms. Because of that reason, in the next section, I list a number of notable sparse linear system solvers. Each of them has different set of capabilities and optimizes for certain type of problems. Experimenting with them could lead to improvement in robustness and performance of both solvers with physical simulation problems. For now, dVC2d implementations only use default Matlab function backslash ($A\backslash b$).

### 6.2.3 Notable implementations of sparse linear system solver

Direct solvers for sparse matrices involve much more complicated algorithms than for dense matrices. The main complication is due to the need for efficient handling of the fill-in of the factors. A typical sparse solver consists of four steps as opposed to two in the dense case:

1. **Ordering step**: Reorders the rows and columns such that the factors suffer little fill-in, or that the matrix has special structure such as block triangular form.
2. **Symbolic factorization step**: determines the nonzero structures of the factors and create suitable data structures for the factors.

3. **Numerical factorization step**: computes the factors \((L, U)\) for LU-decomposition or \(L\) for Cholesky decomposition.

4. **Solve step**: performs forward and back substitution using the factors.

There is a vast variety of algorithms associated with each step. SuperLU [DEG+99] is a general purpose library for the direct solution of large, sparse and nonsymmetric systems of linear equations on high performance machines. The library routines perform an LU decomposition with partial pivoting and triangular system solves through forward and back substitution. SuperLU also comes with a shared memory parallel machine version: SuperLU_MT [DGL99] and a distributed memory version: SuperLU_DIST [DGL99]. Paridiso [SBR06] is also a very popular implementation of large sparse linear solver based on LU-decomposition. It is actively maintained and developed by Olaf Schenk of University of Basel. MUMPS [ADKL01, AGLP06], a MultiFrontal Massively Parallel sparse direct Solver, MA57 [Duf04] and Cholmod [DH09] are excellent Cholesky decomposition based solvers for sparse symmetric linear systems. OpenNL [BCLar] is the newest addition to the long list of sparse linear solvers. An attractive feature of OpenNL is that it can be run on modern GPUs.

### 6.3 Convex optimization based approach

This solution method solves two convex quadratic programming problems repeatedly with different constants each time. If the solutions of two problems converge, we have also found the solution of original non-convex MLCP. Even when they do not converge after a number of iterations, the result is still relevant for most
### Serial platforms

<table>
<thead>
<tr>
<th>Code</th>
<th>Technique</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHOLMOD</td>
<td>Left-looking</td>
<td>SPD</td>
</tr>
<tr>
<td>MA57</td>
<td>Multifrontal</td>
<td>Sym</td>
</tr>
<tr>
<td>MA41</td>
<td>Multifrontal</td>
<td>Sym-pat</td>
</tr>
<tr>
<td>MA42</td>
<td>Frontal</td>
<td>Unsym</td>
</tr>
<tr>
<td>MA67</td>
<td>Multifrontal</td>
<td>Sym</td>
</tr>
<tr>
<td>MA48</td>
<td>Right-looking</td>
<td>Unsym</td>
</tr>
<tr>
<td>Obio</td>
<td>Left/right/Multifr.</td>
<td>sym, out-core</td>
</tr>
<tr>
<td>SPARSE</td>
<td>Right-looking</td>
<td>Unsym</td>
</tr>
<tr>
<td>SPARSPAK</td>
<td>Left-looking</td>
<td>SPD, Unsym, QR</td>
</tr>
<tr>
<td>SPOOLES</td>
<td>Left-looking</td>
<td>Sym, Sym-pat, QR</td>
</tr>
<tr>
<td>SuperLU1</td>
<td>Left-looking</td>
<td>SPD</td>
</tr>
<tr>
<td>SuperLU</td>
<td>Left-looking</td>
<td>Unsym</td>
</tr>
<tr>
<td>UMFPACK</td>
<td>Multifrontal</td>
<td>Unsym</td>
</tr>
</tbody>
</table>

### Shared memory parallel machines

<table>
<thead>
<tr>
<th>Code</th>
<th>Technique</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCSLIB-EXT</td>
<td>Multifrontal</td>
<td>Sym, Unsym, QR</td>
</tr>
<tr>
<td>Cholesky</td>
<td>Left-looking</td>
<td>SPD</td>
</tr>
<tr>
<td>DMF</td>
<td>Multifrontal</td>
<td>Sym</td>
</tr>
<tr>
<td>MA41</td>
<td>Multifrontal</td>
<td>Sym-pat</td>
</tr>
<tr>
<td>MA49</td>
<td>Multifrontal</td>
<td>QR</td>
</tr>
<tr>
<td>PanelLLT</td>
<td>Left-looking</td>
<td>SPD</td>
</tr>
<tr>
<td>PARASPAR</td>
<td>Right-looking</td>
<td>Unsym</td>
</tr>
<tr>
<td>PARDISO</td>
<td>Left-right looking</td>
<td>Sym-pat</td>
</tr>
<tr>
<td>SPOOLES</td>
<td>Left-looking</td>
<td>Sym, Sym-pat</td>
</tr>
<tr>
<td>SuiteSparseQR</td>
<td>Multifrontal</td>
<td>Rank-revealing QR</td>
</tr>
<tr>
<td>SuperLU_MG</td>
<td>Left-looking</td>
<td>Unsym</td>
</tr>
<tr>
<td>TAUCS</td>
<td>Left/Multifr.</td>
<td>Sym, Unsym, out-core</td>
</tr>
<tr>
<td>WSMP</td>
<td>Multifrontal</td>
<td>SPD, Unsym</td>
</tr>
</tbody>
</table>

### Distributed memory parallel machines

<table>
<thead>
<tr>
<th>Code</th>
<th>Technique</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMF</td>
<td>Multifrontal</td>
<td>Sym</td>
</tr>
<tr>
<td>DSCPACK</td>
<td>Multifrontal</td>
<td>SPD</td>
</tr>
<tr>
<td>MUMPS</td>
<td>Multifrontal</td>
<td>Sym, Sym-pat</td>
</tr>
<tr>
<td>PaStiX</td>
<td>Left-right looking</td>
<td>SPD</td>
</tr>
<tr>
<td>PSPASES</td>
<td>Multifrontal</td>
<td>SPD</td>
</tr>
<tr>
<td>SPOOLES</td>
<td>Left-looking</td>
<td>Sym, Sym-pat, QR</td>
</tr>
<tr>
<td>SuperLU_DIST</td>
<td>Right-looking</td>
<td>Unsym</td>
</tr>
<tr>
<td>S+</td>
<td>Right-looking</td>
<td>Unsym</td>
</tr>
<tr>
<td>WSMP</td>
<td>Multifrontal</td>
<td>SPD, Unsym</td>
</tr>
</tbody>
</table>

Figure 6.2: List of softwares to solve sparse linear systems using direct methods. Screen captured from Xiaoye Sherry Li’s survey [Li10].

Gaming and graphics applications. In [KEP05] and [KSJP08], the authors reported convergence in many cases after a small number of steps. There are many convex quadratic programming solvers freely available for academic use. For example, Object-oriented software for quadratic programming (OOQP) [GW03] is a C++ library based on a primal-dual interior-point method, for solving convex quadratic
programming problems. BPMPD [Ms10] is a very efficient implementation of a primal-dual interior point algorithm for convex quadratic programming with advanced warm-start, linear system presolved and sparsity heuristic. It is always one of the fastest solvers in every convex QP benchmark.

First, we need to define the two convex sub-problems.

6.3.1 Subproblem I: fixed friction, find normal force

For this subproblem, we will assume that the frictional forces are known. Therefore we can drop equations (6.3) and (6.4) which are used to compute friction forces and solve the resulting mixed LCP composed of equations for the normal forces $\lambda_n$ and corresponding velocity $\nu_1^{\ell+1}$. The MLCP($\lambda^i_t, \lambda^i_o$):

$$0 = -M\nu_1^{\ell+1} + M\nu^{\ell} + h(W_n\lambda_n^{i+1} + W_t\lambda_t^{i} + W_o\lambda_o^{i} + \lambda_{\text{app}} + \lambda_{\text{vp}}) \quad (6.11)$$

$$0 \leq \lambda_n^{i+1} \perp W_n^T\nu_1^{\ell+1} + \Psi_n^i \geq 0 \quad (6.12)$$

This problem can be reformulated as convex quadratic program as follow:

$$\begin{align*}
\text{minimize} & \quad \frac{1}{2} \nu_1^{\ell+1T}M\nu_1^{\ell+1} + c^T\nu_1^{\ell+1} \\
\text{subject to} & \quad W^T\nu_1^{\ell+1} \leq b
\end{align*} \quad (6.13)$$

With $c = -(M\nu^{\ell} + h(W_t\lambda_t^{i} + W_o\lambda_o^{i} + \lambda_{\text{app}} + \lambda_{\text{vp}}))$, $W = -hW_n$ and $b = \Psi_n^i$

6.3.2 Subproblem II: fixed normal, find friction force

For this subproblem the non-penetration force result of Problem I, $\lambda_n^{i+1}$, is used to solve for the frictional forces in the original formulation. However, we will only consider the friction forces associated with the non-zero $\lambda_n^{i+1}$, thereby avoiding the degenerate Jacobian. For brevity, this problem is shown with the same variables, but in reality the unknowns, $\lambda_t^{i+1}, \lambda_o^{i+1}, \sigma^{\ell+1}$ will have a dimension equal to the
number of non-zeros in $\lambda_{n}^{i+1}$. In this problem, equation (11) will never yield a zero row in the Jacobian, since $\lambda_{n}^{i+1} > 0$. The MLCP($\lambda_{n}^{i+1}$):

$$0 = -M\nu_{2}^{\ell+1} + M\nu^{\ell} + h(W_n\lambda_{n}^{i+1} + W_t\lambda_{t}^{i+1} + W_o\lambda_{o}^{i+1} + \lambda_{app} + \lambda_{vp})$$ (6.14)

$$0 \leq \lambda^{\ell+1} \perp W_t^T\nu_{2}^{\ell+1} + E\sigma^{\ell+1} \geq 0$$ (6.15)

$$0 \leq \sigma^{\ell+1} \perp U\lambda^{\ell+1} - E^T\lambda^{\ell+1} \geq 0$$ (6.16)

This MLCP can also be reformulated as a convex quadratic programming problem.

6.3.3 The algorithm

After solving the two subproblems the values of $\nu_{1}^{\ell+1}$ and $\nu_{2}^{\ell+1}$ are most likely not going to be equal, which means we did not find a solution to the original NCP. However, we can iteratively solve the two problems, hoping to eventually converge:

**Algorithm 3:** Convex optimization-based solution method.

1. $\lambda_i^0 = 0; \lambda_o^0 = 0; \nu_{1}^{\ell+1} = \epsilon + 1; \nu_{2}^{\ell+1} = 0$;
2. $i = 1$;
3. while $|\nu_{1}^{\ell+1} - \nu_{2}^{\ell+1}| > \epsilon$ do
4.   $\nu_{1}^{\ell+1}, \lambda_i^{i} \rightarrow$ Solve MLCP($\lambda_{t}^{i-1}, \lambda_{o}^{i-1}$);
5.   $\nu_{2}^{\ell+1}, \lambda_i^{i}, \lambda_o^{i} \rightarrow$ Solve MLCP($\lambda_{n}^{i}$) with positive components of $\lambda_{n}^{i}$;
6.  $i \rightarrow i + 1$;
7. end

6.3.4 Convergence and solution of original problem

We can prove that if both $\nu_{1}^{\ell+1}$ and $\nu_{2}^{\ell+1}$ converges, then $\nu^{\ell+1}$ is a part of original problem’s solution.

Assume that $(\nu^{\ell+1}, \lambda_{1n})$ solves MLCP($\lambda_{1t}, \lambda_{1o}$) and $(\nu^{\ell+1}, \lambda_{2t}, \lambda_{2o}, \sigma_{2})$ solves
MLCP(λ₁n) we can easily verify that (ν₊₁, .LargeMathSymbols λ₁n,  LargeMathSymbols λ₂t,  LargeMathSymbols λ₂o,  LargeMathSymbols ̂σ₂) solves original problem, where operator ̂x means padding x with zero at positions where the first friction forces were zero.

**Lemma 6.3.1** If ̂ν₊₁ is the solution of the original MLCP, then ν₊₁ ≤ ̂ν₊₁ ≤ ν₊₁.

I leave the proof of this lemma for future work. Given it can be proven, we can then find the maximum error of the algorithm after a certain number of iterations.

### 6.4 Numerical results

In dVC2d Matlab, I have implemented two Levenberg-Marquardt based solvers.

#### 6.4.1 Example one

![Figure 6.3: dVC2d simulation results (320 time steps) of a scene contains a box and a triangle using non-convex formulation and collision detection.](image-url)
In this example, dVC2d Matlab simulates a frictional scene containing a triangle and a box using non-convex formulation and collision detection. Many simulation information of the first 320 time steps were reported in this example.

![Figure 6.4: Example one] The running time of different solvers of the simulation in figure [6.3](#)

<table>
<thead>
<tr>
<th>Type</th>
<th>PATH</th>
<th>Normal Levenberg-Marquardt</th>
<th>Over-constrained Levenberg-Marquardt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>0.0039</td>
<td>0.0031</td>
<td>0.0059</td>
</tr>
<tr>
<td>Max</td>
<td>0.0113</td>
<td>0.0114</td>
<td>0.0264</td>
</tr>
<tr>
<td>Average</td>
<td>0.0058</td>
<td>0.0050</td>
<td>0.0120</td>
</tr>
</tbody>
</table>

Table 6.1: Min, max and average running time of three solvers: PATH, normal and over-constrained Levenberg-Marquardt.

In term of running time, normal Levenberg-Marquardt outperforms both PATH and the over-constrained version. The main reason is that at every step, the over-constrained version requires the solution of a least square problem of twice the size of the squared system of linear equation in the case of the original version.

The value of the merit function is a crucial indication of MLCP solvers’s accuracy. It is easy to see that over-constrained version produces much more accurate
Figure 6.5: [Example one] The problem size at different time steps of the simulation in figure 6.3.

and stable results. PATH accuracy was set at around $10^{-4}$ and was not reported here.

### 6.4.2 Example two

In the second example, a scene with 4 boxes were dropped into a ramp below. This time, Stewart-Trinkle formulation was used. As the size of the problem grows bigger, over-constrained Levenberg-Marquardt solver’s running time was punished even more. It has to solve a harder (least square compares to system of linear equations) and bigger problem. But again, over-constrained solver, along with PATH, produces much more robust and accurate results compare to the original one. The plot of the values of merit function over each time step clearly shows that.
6.4.3 Example three

In this example, the inputs are simulation problems recorded using dVC3d and Blender 3D tool of a large box stacking problem in figure 6.11. The size of the problems grows fast during simulation as shown in figure 6.12. And finally, the running time plot 6.13 shows that the Fischer-Burmeister solver outperforms the PATH solver about 30 times in average. The Fischer-Burmeister solver is also more stable while PATH solver’s solution time varies widely during the simulation. Moreover, even the complementarity gap plot shows in figure 6.14 that the Fischer-Burmeister solver is only less accurate than the PATH solver but its average is very small (less than floating point double precision epsilon).

6.5 Conclusion and future works

In this chapter, two different approaches to solve the MLCP resulted from physical simulation are presented. The first approach was based on the development
Figure 6.7: dVC2d simulation results (320 time steps) of a scene containing 4 boxes and a ramp using Stewart-Trinkle formulation.

of convex quadratic programming. It was inspired by many previous works. I believe this type of methods could be attractive to computer graphics and game developers due to its speed advantage and the ability to generate visual believable simulations with small number of iterations.

The second class of solution methods presented in this chapter is based upon Levenberg-Marquardt, a modern and more robust variation of Newton’s method. We discussed two different MLCP solvers of this class. The first one is a basic implementation of original Levenberg-Marquardt with a simple trust region scheme. The second solver is built on top an over-constrained least square Levenberg-Marquardt with very strong theoretical backing. Based on the numerical results, we can see that both methods can be valuable to different physical simulation requirements. The simple implementation often outperforms PATH in terms of speed even it is written entirely in Matlab while PATH is a more optimized Mex file. The over-constrained
Figure 6.8: [Example two] The running time of different solvers of the simulation in figure 6.7.

solver is usually slower but consistently produces very accurate results.

Note that the two methods described above are only served as an proof of concept that this type of solution methods are suitable for physical simulation problems. There are many possible improvements for both implementations. First, PATH has been developed for fifteen years therefore its linear system solvers are very good at handling degeneracy. A better linear system solver will definitely improve both performance and accuracy of both new solvers. Second, Levenberg-Marquardt and Newton-based methods are extensively studied in literatures. There are many researches on how to improve the solvers given different problems. We can detect the types of resulting MLCP by using our physical interpretations of the simulation problems then pick proper optimizations to use. Last but not least, the most expensive operation of the solvers is solving a system of linear equations. But there are many parallel and distributed libraries for this task. Therefore, we can build parallel solvers to scale up with size of the simulation problems if needed.
Figure 6.9: [Example two] The problem size at different time steps of the simulation in figure 6.7.

Figure 6.10: [Example two] The values of merit function of normal and over-constrained Levenberg-Marquardt solvers of the simulation in figure 6.7.
Figure 6.11: A box stacking scene with large number of boxes simulated using dVC3d inside Blender 3D.

Figure 6.12: [Example three] The size of the main matrix in during simulation of the scene in figure 6.11.
Figure 6.13: [Example three] The running time of the two solvers in 121 time steps of the scene in figure 6.11.

Figure 6.14: [Example three] The complementarity gap of the two solvers in 121 time steps of the scene in figure 6.11.
CHAPTER 7

Accurate multi rigid bodies physics engine implementations

7.1 Introduction

In this chapter, I will introduce three of my most recent physics engine implementations. They are dVC3d C++ library and tools, dVC3d Matlab and dVC2d Matlab. Each of them has different set of objectives and use cases. dVC3d C++ library is a full-features physics engine that can be used for complex 3D simulations. It is also integrated with various external tools to improve the usability and exposure. But dVC3d does not implement the recent methods described in chapters 4, 5 and 6. dVC3d Matlab implementation can be used for 3D physics engine prototyping and researches. It has a 3D rendering system and implements all functionalities needed for a physics engine in both native Matlab code or as Matlab external interfaces (mex files and .NET assemblies). I used dVC3d Matlab mostly to study the detailed structures of the complementarity problems originated from physical simulation. Last, dVC2d Matlab was my most recent implementation. It was the proof of concept of the new advanced contact model and collision detection. dVC2d hosts the implementations of the new Levenberg-Marquardt based solution methods proposed in chapter 6. It can be used as a reference for future 3D implementations. All the implementations are freely available in my homepage.

7.2 dVC3d library

dVC3d is a cross-platforms, object-oriented, C++ accurate physics engine. dVC3d implements Bullet Physics engine application interface. There are several reasons for me to drop dVC2d C++, the direct ancestor, interface and switch
to Bullet:

- Bullet physics engine has a large and active group of users, including many robotics researchers. Many robotics experiment platforms are using Bullet as their physical simulation engine, to name a few: Carnegie Mellon University’s OpenRAVE, a Planning Environment for Autonomous Robotics [DK08] and the commercial Virtual Robots Experimental Platform [Fre09].

- Bullet application interface was designed and maintained by a group of professional developers. They spent considerable amount of time on software engineering to make it user-friendly and flexible.

- dVC3d already relies on Bullet for collision detection. Bullet’s flexible design allows me to make no compromise to implement dVC3d as a Bullet extension.

- Bullet is integrated with Blender [Fou10], one of the most popular open-source 3D content creation tool. It opens the door for dVC3d users to create, design, setup and run their simulation inside Blender and Python programming language.

dVC3d has been under development for about two years and it is still a work in progress. I have released the first version of dVC3d 1.0 in September 2010. dVC3d is the most mature 3D implementation of accurate physical simulation methods. There are many possible future features that should benefit dVC3d users greatly. To name a few: new sparse Levenberg-Marquardt based solvers, implementation of advanced contact model and collision detection, parallel linear system solvers and tighter interpolation with Blender to allow users to take advantage of dVC3d simulation through Blender and Python.
7.3 dVC3d Matlab version

Along with dVC3d C++ version, I have developed a Matlab interface for the physics engine. dVC3d Matlab serves as a quick prototype platform to test new methods in physical simulation.

Figure 7.2 shows a simulation involves an external geometry (the plane) and an user created shape (the box). dVC3d Matlab has built-in interpolation with dVC3d and Bullet physics engine through Microsoft .NET interface. In the future, I may consider switching to Java for better cross platform support.

7.3.1 Simple example of dVC3d Matlab

Below is the source code of a simple demo of dVC3d Matlab:

```matlab
1  % create the world
2  g.world = dvcBTWorld();  
3  % setup body and drawable   
4  % drawable                   
```
Figure 7.2: dVC3d Matlab.

```
5  b1 = dvcOGLDrawable('models/cube.obj');
6  % body1
7  body1 = dvcBody();
8  body1.geom = b1;
9  body1.q = [0 1 0 0 0 0 1];
10 % body2
11  body2 = dvcBody();
12  body2.geom = b1; % Share geom
13  body2.setQ( [3 1 0], [0 1 0], 45); % Using helper function
14  body2.color = [0.4 0.4 0] ;
15 % add them to the world
16  g_world.addBody(body1);
17  g_world.addBody(body2);
```
Line 2 creates a default instance of `dvcBTWorld` class which is an interface to dVC3d C++ Stewart-Trinkle time stepper implementation. We can replace it with any Matlab or C++ implementations if they expose similar interface. Line 4 loads a model file and assign to a `dvcOGLDrawable` variable which can be used to hold the geometry of a shape. Lines 7-14 create two rigid body `dvcBody` in variables `body1`, `body2` with different position and orientation but the same geometry is loaded in line 4. Line 16 and 17 add `body1`, `body2` to the physical world `dvcBTWorld`. Line 19 and 20 create and later add the ground to the active world. Finally, the simulation is created in line 22 through the function `setupApp`.

Figure 7.3: A screenshot of simple dVC3d Matlab simulation of two boxes sitting on the ground.
7.4 dVC2D Matlab

dVC2d Matlab is implemented recently as a proof of concept for the new extended contact model and collision detection proposed in chapters 4 and 5. I implemented a number of different friction models: linearized Coulomb, gaming physics engine and convex friction model in dVC2d. dVC2d also includes two implementations of Levenberg-Marquardt based solvers in chapter 6 along with PATH [FM96].

Below is a short example of how to use dVC2d along with the simulation result:

```matlab
1 1 % Configuration
2 sim = defaultSim();
3 sim.enableFriction = 1; % enable friction
4 sim.timeStepper = 'BT'; % Non-convex method
5 sim.solver = 2; % New Levenberg-Marquardt solver
6 sim.epsilon = .2; % collision epsilon
7 sim.negativeEpsilon = 0.1; % absolute value of lower bound
8 sim.boundEpsilon = 1; % epsilon for walls
9 sim.boundingSphereEpsilon = .5; % epsilon for bounding spheres
10 % Rendering options
11 sim.maxSteps = 200;
12 sim.ghosting = 1; % ghosting effects
13 sim.ghostkey = 20; % save position of all bodies every 20 frames
14 % Adding body
15 % a moving box
16 box1.x = [-1 -1 1 1]+1; % world coords of all the vertices
17 box1.y = [ 2 3.8 3.8 2]';
18 box1.color = [.31 .31 .31];
19 box1.rot = 45; % initial rotation
20 box1.mass = 1;
21 % an triangle shape obstacle
```
tris1.x = [-1 0 1]'+.3; % local coords of all the vertices
tris1.y = [-3 -2 -3]';
tris1.color = [.24 .24 .24];
tris1.mass = 0; % static object

sim.body = {tris1 box1}; % list of bodies
dvc = initSim(sim); % initialize and run simulation

Figure 7.4: Simulation result of the simple Matlab example above.
CHAPTER 8
Discussions and conclusions

In conclusion, this thesis’s goals are to provide better understanding and improves the theories and implementations of computer physical simulation. At the beginning, I introduce the layers of abstraction of physical simulation, starting with physical model, then to mathematical and finally computer model. Each new layer introduces a number of approximations that the users of physical simulation need to understand in order to use the tool correctly and efficiently. These abstraction layers also provide a big picture of physical simulation that could help to explain certain decisions. However, the main focus of this thesis is not only on understanding but also developing new theories and implementations of physical simulation:

- A new advanced contact that can accurately model the free spaces where previous contact model cannot. Importantly, the cost of the new accuracy is very small. In fact, the size of the final MLCP matrix under the advanced contact is equal or even smaller than the matrix under previous model.

- A novel, un-ambiguous and simple collision detection. Current collision detection techniques are unclear and often very complicated because they have to handle many edge cases. The new collision detection method proposed in this thesis solves most of these problems. Because of its simplicity, it is also suitable to run in parallel in modern hardware.

- New approach for solution methods. A number of new methods to solve the MLCP resulted from the formulations with original and new contact model are presented. At present, we can only rely on PATH to solve the problems. Experiments show that the new solution methods could outperform PATH in
practice. Also, there are many possible optimizations to improve the speed and accuracy of the solvers. Most importantly, these solvers are built on top many available optimization and solver libraries to reduce the time and complexity of any implementations.

In the future, there are many ways to extend the works of this thesis. One possible direction is to implement a completely new 3D physics engine based on the new advanced contact model, collision detection and solvers. All of these components should be ready to run in parallel. Once the goal is achieved, it will open many new possibilities in the field of robotics, machine design and even gaming and computer graphics. In collision detection, more works are needed to find better heuristics for medium-phase and choosing the collision parameters to improve its performance. This thesis also opens many future directions for new solution methods. In particular, developing a better linear system solver that can detect and make use of certain physical interpretations and carefully handle degeneracy cases of the problems should greatly improve both performance and accuracy of these solvers.
REFERENCES


[Ebe08a] David Eberly. Distance between point and line, ray, or line segment. WWW page, 1999-2008.


