2013-11-22

The Pseudo-Rigid-Body Model for Fast, Accurate, Non-Linear Elasticity

Anthony R. Hall
Brigham Young University - Provo

The Pseudo-Rigid-Body Model for Fast, Accurate, Non-Linear Elasticity

Anthony R. Hall

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science

Brigham Young University

November 2013

ABSTRACT

The Pseudo-Rigid-Body Model for Fast, Accurate, Non-Linear Elasticity

Anthony R. Hall
Department of Computer Science, BYU
Master of Science

We introduce to computer graphics the Pseudo-Rigid-Body Mechanism (PRBM) and the chain algorithm from mechanical engineering, with a unified tutorial from disparate source materials. The PRBM has been used successfully to simplify the simulation of non-linearly elastic beams, using deflections of an analogous spring and rigid-body linkage. It offers computational efficiency as well as an automatic parameterization in terms of physically measurable, intuitive inputs which fit naturally into existing animation work flows for character articulation. The chain algorithm is a technique for simulating the deflection of complicated elastic bodies in terms of straight elastic elements, which has recently been extended to incorporate PRBM beam-elements in three dimensions. We present a new, mathematically equivalent optimization of the 3D PRBM chain algorithm, from its former asymptotic complexity of $\mathcal{O}(n^2)$ in the number of elements n, to $\mathcal{O}(n)$. We also extend an existing PRBM for combined moment-force loads to 3D, where the existing 3D PRBM chain algorithm was limited to 3D PRBM elements for a moment-only load. This optimization and extension are validated by duplicating prior experimental results, but substituting the new optimization and combined-load elements. Finally, a loose road-map is provided with several key considerations for future extension of the techniques to dynamic simulations.

Keywords: simulation, rigid-body, mass-spring, non-linear elasticity

ACKNOWLEDGMENTS

Dean R. Wheeler first acquainted me with the work of Larry L. Howell, and pointed me to the PRBM as an effective method for modeling compliant mechanisms.

Murphy J. C. Randle wrote pipeline scripts for interfacing our results with existing packages for animation and rendering. He set up the lighting, texturing, cameras, rendering, and other scene elements necessary for the rendered images found in the figures.

Larry L. Howell generously offered his time and his expertise with PRBM techniques, and offered early encouragement about the relevance and interest of our research goals. He independently reviewed my thesis proposal for general soundness, and his feedback regarding the PRBM background material and our proposed extensions helped encourage us to proceed.

H. Tracy Hall re-formulated my earliest long-hand, verbose formulas for the $\mathcal{O}(n)$-optimized chain algorithm into the compact, elegant representation of section 4.1.2 as linear combinations of basis matrices. The results of chapter 5 came from implementing his formulation; my own later refinement of the earlier method appears in appendix A. As a mathematical and technical consultant, he helped push the work forward through a few key whiteboard sessions.

I would like to thank Parris Egbert and Daniel Zappala for willingly serving and giving their time as members of my thesis committee. Dr. Egbert took additional time to review and give important feedback for both the Thesis Proposal document and this Thesis.

My adviser and committee Chair, Michael Jones, provided invaluable support throughout my entire program. He offered me the freedom and trust to tackle a problem and set of techniques in which neither of us had prior expertise. He was intellectually generous, always willing to invest the time to understand and help me clarify the new concepts and techniques I uncovered in the literature. His technical insights were key to a number of important breakthroughs. He never kept me intellectually subordinate, and welcomed challenging questions and push back; my confidence as a researcher was able to flourish as a result. His help with planning and strategy was critical in defining a clear, manageable, and implementable scope for the project; without his guidance, the project would have been much more difficult to manage and ultimately finish.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

We introduce to computer graphics the use of the Pseudo-Rigid-Body model (PRBM) [Howell 2001] from mechanical engineering as a key element in methods for rapidly simulating the deformation of complicated, non-linearly elastic bodies. The PRBM has been used successfully in the design and study of compliant (elastic) mechanisms, where it has proven to be an invaluable tool for rapidly "sketching" and testing design prototypes [Chase et al. 2011, Howell 2001].

The methods are suitable for bodies that can be discretized into non-cyclic graphs (trees) of prismatic elastic beam elements with arbitrary cross-sections, where the primary modes of deformation are bending and twisting (see fig. 1.1 for examples of these and other modes of deformation). Examples would be natural trees and other vegetation, springs, stiff cords and cables, diving boards, and shark skeletons. PRBMs are not tuned for volume or area deformations such as are seen in muscle, sponges, skin, or cloth. The methods target elastic deformation, or deformations that will tend to naturally undo themselves in the absence of an external load. This is in contrast to plastic deformation, where shape and material properties may change permanently as a result of deformation.

The Pseudo-Rigid-Body Method is so called because it accurately simulates elastic beams using rigid links and torsion springs which obey Hooke's law. It therefore enables the simulation of complicated non-linear elastic behaviors using well-established and comparatively simple methods for simulating constrained rigid bodies, where traditional methods for non-linear elasticity would be mathematically and computationally complex. It provides simple formulas for tuning the mass-

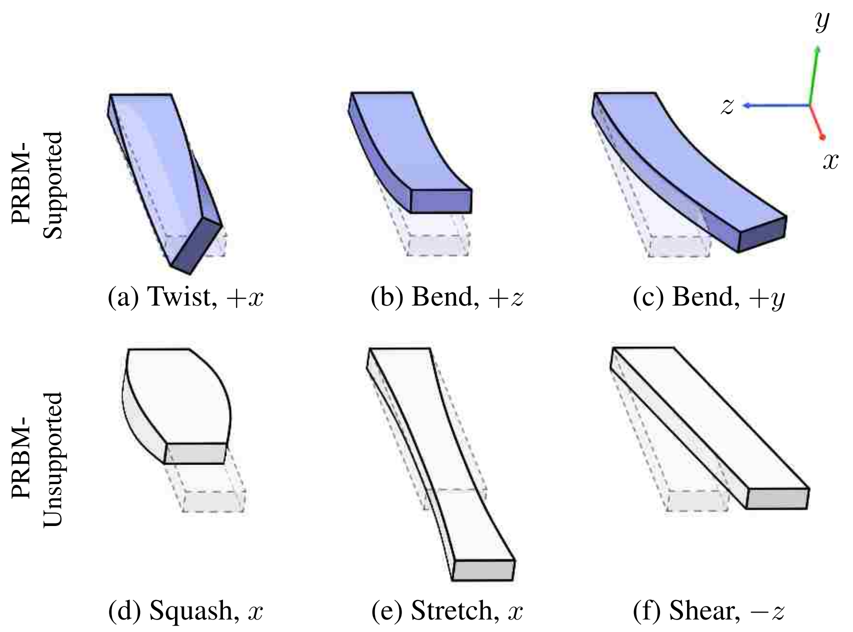


Figure 1.1: *Examples of beam deformation modes. Lavender shaded beams (top row) portray deformation modes supported by the 3D PRBM. White beams (bottom row) represent modes that are not supported by the 3D PRBM. Modes portrayed are* (a) *Twist or torsion by a postive angle about the local x axis;* (b) *Bend by a positive angle about z;* (c) *Bend by a positive angle about y;* (d) *Squash along x;* (e) *Stretch along x; and* (f) *Shear along the negative z axis.*

spring system directly from measurable physical properties such as Young's modulus, so conversion from a given input model to a model for simulation can be largely automated, given appropriate geometry and material descriptors as input. It is more versatile than classical models such as Bernoulli's beam equations, because it is accurate over large deflection angles.

1.1 Immediate contributions of this work

Since this paper focuses principally on static[1] solutions, it will have the most immediate application in film and games for setting static and start-of-shot poses for elastic bodies.

One example application is as an alternative to simulation pre-roll. Pre-roll is a technique in animation to prepare a simulated body for its correct start-of-shot pose when its rest pose does

[1]By 'static', we mean that the methods identify the correct equilibrium pose a body will assume given an initial rest pose and a set of constant loads that deflect it out of this rest pose. The rest pose represents the natural pose of the body when all net loads are zero.

not conform to the action and staging at the start of a shot—for example when a film cuts to a tree under a steady wind, or to a character's garment draped over their body under gravity rather than in the originally modelled tailoring pose. Pre-roll runs a number of frames of dynamic simulation in advance of the first shot frame, to allow the body to transition into the required starting configuration. For static start-of-shot poses such as the examples given above, a single run of our fast static method could substitute for numerous frames of pre-roll.

Another example application is deducing the rest pose of a model which was obtained under an initial load—such as a tree modeled from real-life examples under the influence of gravity—by running a static simulation on the model under equal but opposite loads.

Finally, our linear-time optimization of the chain algorithm from mechanical engineering will immediately improve run-times for existing engineering methods based on this algorithm. PRBM-based methods have already been applied with great success in facilitating and accelerating the design of machines and mechanisms based on elastic behavior, and recent efforts to incorporate PRBM elements in the chain algorithm promise to expand the possibilities even further [Pauly and Midha 2006, Chase et al. 2011]. The increased speed of our new form of the PRBM chain algorithm will assist these efforts.

1.2 Long-term contributions

We hope that introducing PRBM-based methods to computer graphics will foment further study of their application in film and games, particularly in their extension to dynamic, time-stepped simulations for fully animated results. We anticipate numerous future contributions of this field to the production of films and games.

These methods will benefit the work of animators and technical artists by providing visual accuracy while reducing turnaround times for sequence and shot work. The automatic tuning of spring stiffness values and other parameters will eliminate some trial-and-error guesswork when tuning a shot for simulation, closing the gap between what a simulation should look like given certain physical inputs, and what it does look like.

The methods will integrate well with ongoing research efforts to make simulations more directable. Directability aims to give direct, focused, and intuitive control of simulations to animators and directors, without the prohibitive workload of hand-producing every detail. This is an active area of research, with numerous significant results [Barbič et al. 2012, Coros et al. 2012, Hahn et al. 2012, Hildebrandt et al. 2012]. In the worst case, artistic direction requires a time-consuming, manual process of waiting for a simulation to finish, evaluating the results, adjusting the input parameters, and repeating till the results are good enough. Directability aims to make this process more interactive, with the simulation itself more open to artist intervention. One approach, for example, is by fitting and deducing simulation parameters to conform to constraints supplied by intuitive forms of input such as artist-defined targets or painted motion gestures. Two significant requirements for such methods are that the underlying simulation method be fast enough to give feedback at interactive rates, and that the parameter space of the simulation not be so divergent from more intuitive animation parameters that the system cannot translate quickly between the two. Our PRBM-based methods are fast, and are built around natural, intuitive inputs which are trivially converted to inner simulation parameters, so they fit these two requirements for directable methods.

The efficiency and simplicity of PRBMs also suit them for incorporation in video games, where environments, sets, and interacting objects can be greatly enriched by the introduction of elastic behaviors.

Chapter 2

Related Work

Many of the works cited in this chapter implement dynamic simulations, where this paper focuses on static applications of the PRBM. Hence, we focus on comparisons that relate most to the static case.

Our methods are related to existing work for general elastic methods, Finite Element reduction, and specialized methods for strands and 1D elastic materials. Our methods are less versatile than the general methods we discuss in section 2.1, while yielding greater accuracy over some methods and great computational efficiency over others. The Finite Element reduction methods in section 2.2 are versatile and very fast, but remain less agile than our method in the face of dynamically changing model or simulation parameters, due to non-trivial pre-computation requirements. The specialized strand and 1D element methods which we consider in section 2.3 are better suited than ours to scenarios requiring numerous precise constraints or intricate self-interactions, but they either assume vanishing cross-section and thereby neglect effects such as twisting and varying directional stiffness, require a significant mathematical background to understand, or are difficult to parameterize correctly for physically accurate and predictable results.

2.1 General versus specialized elasticity methods

Many techniques for simulating elastic bodies specialize in either 3D, 2D, or 1D elastic elements. Martin et al. [Martin et al. 2010] point out that this segregation can introduce errors and artifacts, especially in hybrid or boundary cases between types. To address these concerns, they unify the representation of 3D *solids*, 2D *shells*, and 1D *rods* as composites of a fundamental *elaston* element, which accurately captures numerous volume deformation modes such as stretch, bending, shear, and twist.

As Müller and Chentanez [Müller and Chentanez 2011] point out, elastons sacrifice performance for generality and accuracy. Müller and Chentanez introduce *oriented particles*, another unified, general method which more simply encodes the same derivative information of elastons, yielding real-time elastic deformations. The simulation surface is approximated from the original fully detailed model as an aggregate of interconnected, oriented ellipsoids which are simpler to simulate than the detailed mesh. The resulting simulation is visually plausible, though it sacrifices geometric precision for details at a finer grain than what the ellipsoids capture. It also sacrifices accuracy since it is less clear how to parameterize the simulation to match physical measures of elasticity.

If we must choose only two among accuracy, generality, and performance, then PRBM-based methods choose the remaining pair, efficiency and accuracy; where Martin et al. emphasize accuracy and generality, and Müller and Chentanez opt for efficiency and generality. PRBM methods give up generality by focusing on 1D elastic elements that only capture bending and twist, a subset of the elaston's deformation modes. But in so doing, they greatly simplify the treatment of non-linear elasticity. They give accurate and predictable results since they are explicitly and simply parameterized in terms of knowable physical inputs.

2.2 Finite element reduction and coarsening

Finite element methods (FEM) offer a diversity of applications, and can be directly parameterized by physically measurable inputs. However, they can incur extravagant computational costs when simulating fine details, as their complexity increases significantly with increasing element counts.

[Kharevych et al. 2009, Nesme et al. 2009] give examples of *reduction* or *numerical coarsening* methods, with particular attention to the problem of retaining the emergent dynamic effects of fine structures, which would be lost by simply ignoring or truncating higher spatial frequencies. Nesme et al.'s innovations [Nesme et al. 2009] allow for independent motion of branching structures that would otherwise be inappropriately coupled by sharing a single reduced element.

[Barbič and Zhao 2011] simplify the subsequent simulation of reduced meshes, as well as the reduction process in the first place, by first partitioning the mesh into distinct sub-domains and reducing each sub-domain separately, mitigating the poor scaling performance of FEM. During simulation, sub-domain deformations are computed locally in real-time, and coupled to other sub-domains at rigid interfaces, which can also be handled in real-time. Branching dynamics can be preserved by careful selection of the sub-domains.

The reduction process for each of the above cited works is non-trivial. The runtime cost of pre-computation is modest when performed once, but would likely become prohibitive on a per-frame basis. The parameterization of a PRBM element, by contrast, is trivially simple from the published formulas, and is fully local, needing no reference to other elements in the elastic body. Therefore, conversion of the entire elastic body scales linearly in the number of elements. We believe, when general dynamic implementations are realized, these qualities will equip the PRBM for scenarios where online topology or material changes are desirable, such as for handling stress-driven dynamic fracture or branch splitting, or for accommodating plastic changes such as fatigue, creep, and other time- and event-driven material changes.

2.3 Specialized 1D elastic methods

[Bergou et al. 2008] employ discrete differential geometry to represent 1D elastic elements as a baseline curve with a length-parameterized scalar to represent twist at points along the curve. Their representation is physically accurate, and they show remarkable self-collision handling results involving knot-tying, tangling, and other self-interaction-related effects. Sueda et al. [Sueda et al. 2011] give results from their unique hybridization of Eulerian (space-centric) and Lagrangian (element-centric) reference frames that demonstrate unprecedented precision and stability of constraint enforcement.

We do not attempt intricate self-interaction and constraint handling in our work, and the problem is likely non-trivial for PRBM elements. Though Pauly and Midha [Pauly and Midha 2006] allude to the amenability of the chain algorithm to constraint and boundary condition enforcement, we suspect that [Bergou et al. 2008, Sueda et al. 2011] are preferable where the application demands intricate self-interactions and numerous precise constraints. However, the comparative efficiency of Bergou et al.'s method is not clear, and it also requires investment if one is not already familiar with the techniques of differential geometry, where PRBM methods principally require a grounding in basic mechanics and the mathematics of vectors and matrices.

[Selle et al. 2008] give a mass-spring method for simulating individual hairs which accounts for twist and facilitates collisions and friction handling. They assert that reduced coordinate[2] methods (such as ours) can be difficult to adapt for collision handling. However, the implementation we describe in this paper includes ongoing tracking of world-space coordinates and orientations, which when combined with knowledge of element cross-sectional geometry, should mitigate the difficulty for reasonable element densities. It may still arise for sparse element discretizations, since current PRBM sources do not provide formulas for shape and curvature between the end-points, only for the end-point locations and orientations, which may wind up spaced too far apart for good collision detection.

In general, mass-spring methods can be difficult to parameterize accurately, often requiring some guesswork and hand-tuning to arrive at the desired dynamic behavior. In contrast, the tuning of springs internal to PRBM elements is given explicitly by simple formulas based on physically measurable properties such as Young's modulus and the area moment of inertia.

[2]For a brief description of reduced coordinates, along with some of their advantages, see section 6.2.1

[Bertails 2009, Hadap 2006] each give linear-time simulation methods for open-loop (tree-like) graphs of elastic elements, with Bertails incorporating special helical elements, and Hadap introducing *oriented strands* as their fundamental element. The helix elements of Bertails, as well as a novel *clothoid* model introduced by [Casati and Bertails-Descoubes 2013], are *higher order* specializations of 1D elastic elements, which permits them to represent expanses of non-straight geometry with a single element, resulting in many fewer elements overall for some body geometries. The elements of [Casati and Bertails-Descoubes 2013] offer continuity improvements between adjacent elements over those of [Bertails 2009].

While PRBM elements can also capture curves with only a single element, they do so only at the element end-points (as mentioned previously), and under stricter assumptions. Furthermore, capturing 3D space curves, as opposed to merely planar curves, can still require numerous PRBM elements, as demonstrated by the results for varying element counts in section 5.3.3. However, the clothoids of [Casati and Bertails-Descoubes 2013] assume vanishing cross-section inertias, whereas the PRBM's parameter formulas explicitly require specification of the cross-sectional inertias. As a result, PRBMs are more general than clothoids in this respect. Indeed, the experiments of [Chase et al. 2011], which we replicate and build upon in chapter 5, were specifically designed to demonstrate *lateral torsional buckling*, an interesting 3D deformation in which the explicit treatment of imbalanced area moments of inertia plays an essential role.

Chapter 3

Background

The methods in this section primarily represent the past work of others. Though they are available in the cited works, repeating certain details here in a unified tutorial makes a contribution of its own. We have made changes to some of the notations, conventions, and details of the cited algorithms, in order to support a unified framework and presentation, as well as to support our own extensions. The results in chapter 5 confirm that we have maintained mathematical equivalence with the original models and notation.

3.1 Notation and terminology

The term 'PRBM' can be used loosely to describe any of a number of different ideas. For clarity, we will avoid this loose usage of the term, and use these more specific terms for the following cases: (*a*) *chain*: a composite body made up of simpler PRBM elements; (*b*) *PRBM element*: a simple, individual PRBM element; and (*c*) *PRBM recipe*: a specific set of formulas and methods that can be followed to construct a simple PRBM element. A PRBM element normally consists of rigid sub-elements, which we will call 'links.'

Quantities used in the algorithms generally fall under two levels of abstraction: chain-level quantities, and element-level quantities that are internal to a given element. The point or element number to which a chain-level quantity belongs is denoted with a superscript, and the point or link number of an element-level quantity with a subscript. Quantities that are spatially associated with points are indexed beginning at 0, and quantities associated with elements or links begin at 1.

A vector quantity is denoted by bold-face. An individual component of a vector quantity is denoted by subscripted parentheses around the vector, to distinguish from the un-parenthesized subscript for link indexing. For example, $(\boldsymbol{P}^i_m)_z$ denotes the z-coordinate of the m^{th} point of element i.

3.2 History and overview

3.2.1 Overview of the PRBM

A PRBM recipe dictates the specific configuration of links and springs that will accurately simulate an elastic beam, given its physical properties, length, and cross-sectional geometry. The correct choice of recipe is usually a matter of the types of loads the elastic beam will experience. For example, there is a family of similar recipes that apply when the elastic element is *cantilevered*, or fixed to a substrate at its base with the tip free to respond to a load. Among this family are separate recipes for when that load consists only of a linear force, and when it consists only of a moment (or torque). We will give in section 3.3 two of the original planar recipes that relate most closely to our work, and a 3D recipe due to [Chase et al. 2011] in section 3.3.3. Numerous additional recipes may be found in [Howell 2001].

The spring and link configuration of each PRBM recipe were tuned to minimize measures of deflection error in comparison to a rigorous, "ground truth" model of elasticity. A PRBM element configured and simulated using one recipe will not give accurate results under a different scenario. Any PRBM-based method must therefore take care that the assumptions of each PRBM element are met.

3.2.2 Overview of the chain algorithm

Our description of the chain algorithm is drawn primarily from [Pauly and Midha 2006, Chase et al. 2011, Chase 2006]. The chain algorithm is a structurally recursive technique for finding the static equilibrium pose of an elastic body under a set of loads applied at points throughout.

The elastic continuum is first discretized as an acyclic graph (the chain), of linear elements. As an acyclic graph, the chain may be strictly serial, or contain branches, with any number of *child* elements anchored to the tip of a single *parent* element. A single element is designated as the *root* element, and thus becomes the transitive parent (or *ancestor*) of all other elements. The root typically has its base anchored to a substrate. After discretization, the precise location of each applied load may be approximated to the nearest element tip.

The algorithm visits each element, starting at the chain's fixed root and proceeding towards the tip elements. As each element is visited, the structural recursion means that all its parent and ancestor elements are held fixed, with the element itself being considered fixed at its base with a static load at its tip. The static load is computed from all its child and descendent elements. The element is then deformed by that load according to its internal model of elasticity, and finally all descendent elements are rigidly transformed to match, before the algorithm proceeds to the next element.

To converge at the correct static pose, the entire algorithm must often be repeated. Two kinds of iteration are typically employed: *end iteration* and *load incrementing*. End iteration means to simply repeat the chain algorithm with the pose obtained during the previous iteration. This can be necessary because, strictly speaking, the static equations only fully apply when all elements are in their final equilibrium configuration, but this is not the case before final convergence has been reached. Fortunately, the current element configuration may serve as a good "best guess," and the static equations converge towards equilibrium under repeated application. Iteration may be stopped when elements displacements dip below a desired tolerance.

Load incrementing means to apply each external load at a fraction of its full magnitude, then increment it to the next fraction, and repeat until the total load is reached.

Load and end iterations may be combined. For example, it worked well with our implementation to nest end iterations under each load increment. Using a greater number of load increments can decrease the number of end iterations needed for each increment.

3.2.3 History of the Chain Algorithm with PRBM elements

Pauly and Midha [Pauly and Midha 2006] first proposed a framework for inclusion of PRBM elements as the beam elements in the chain algorithm. We refer the curious reader to their paper for an explanation of advantages that a PRBM chain algorithm would hold over pre-existing chain algorithms. Chase et al. [Chase et al. 2011] successfully implemented this algorithm and extended it to 3D. A visual depiction of a chain with PRBM elements is given in fig. 3.1.

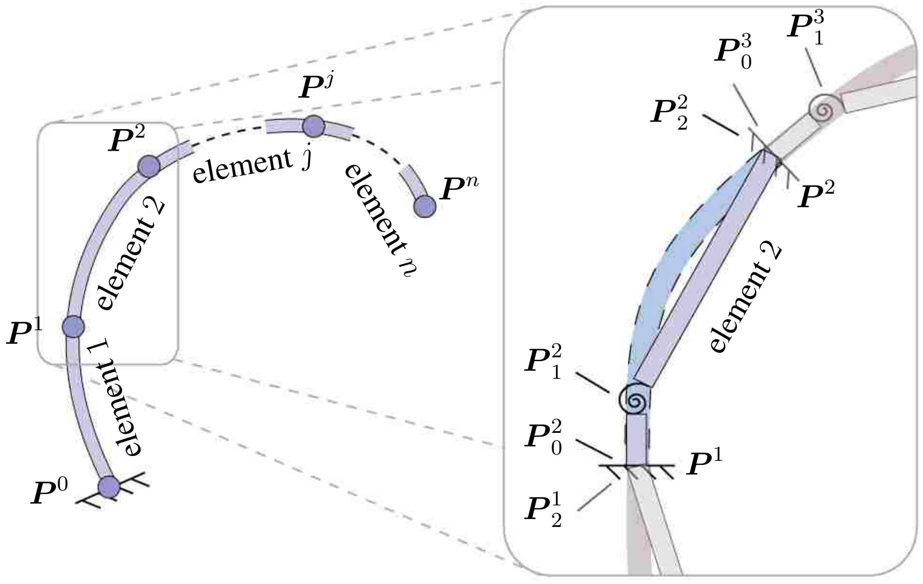


Figure 3.1: *A chain with PRBM elastic elements*

The structural recursion of the chain algorithm means the most appropriate PRBM recipes are those for a fixed-base beam with a loaded tip. But as mentioned in section 3.2.2, at the time of

Pauly and Midha's work, there was no PRBM recipe for a combined moment and force load—only for each kind of load separately. This posed a problem since a general method would need to handle both forces and moments at each element. They observed that each element may be modeled using two concrete PRBM elements separately, each constructed from one of the two recipes. The principle of superposition then allowed the correct tip *translation* to be obtained by adding the translations from each element. However, superposition did not apply for the case of tip *orientation*, so it remained unclear how to obtain the orientation from the two disparate recipes.

Chase et al. [Chase et al. 2011] introduced a simplifying assumption that with sufficient element counts in the discretization, each element can be taken to have length zero. In this way reaction forces come to act at the same point as the external applied force on the element, cancelling it out. The applied force still gives rise to static moments at the other elements throughout the chain, so in effect all forces are converted to moments. Thus no element need model the effect of a force directly, and the PRBM recipe for a moment load suffices for all elements. With the additional contribution of extending the moment-loaded PRBM recipe to 3D, Chase et al. were thus able to realize the PRBM chain algorithm first proposed by Pauly and Midha, and in three dimensions.

The original work behind [Chase et al. 2011] first appeared in Chase's master's thesis [Chase 2006]. After Chase's original work, Su [Su 2009] introduced a new PRBM recipe that simply and accurately handles a combined force-moment load. This recipe immediately suggests itself as an alternative to Chase et al.'s zero-length assumption for overcoming the combined-load problem facing Pauly and Midha's work. As of yet, however, we have not discovered any work in the literature which incorporates this new recipe within the chain algorithm. Since in this paper we will incorporate a 3D extension of Su's recipe in the chain algorithm, we will summarize Su's original recipe, with refinements by Chen et al. [Chen et al. 2011], as background in section 3.3.2.

3.3 PRBM recipes

Sections 3.3.1 and 3.3.2 give planar recipes, which are limited to beam elements that deflect entirely within the plane, with the load parallel to and lying inside the plane. They do not consider bending and twisting of the element outside of the plane. Section 3.3.3 gives Chase et al.'s [Chase et al. 2011] extension of the moment-loaded recipe to 3D, which does account for out-of-plane deflections, and which supports arbitrary 3D moments.

3.3.1 The planar PRBM for a moment load

This recipe is summarized from [Howell 2001].

A moment-loaded PRBM element is useful for obtaining two pieces of information about the elastic element it models: the location of the elastic element's tip under a load, as well as the tip's *orientation*, which is the angle of the tangent line at its tip.

The moment-loaded recipe gives two straight rigid links, initially laid end-to-end in a straight line, coupled by a torsion spring as a revolute joint. The elastic beam being modeled is assumed to be initially straight, with uniform cross-sectional geometry along the length of the element. As alluded to in section 3.2.1, the base rigid link is assumed fixed, with the tip link free to rotate about the joint in response to a moment load.

The lengths of the two rigid links are respectively determined by multiplying the element's length l with two constants, γ_1 and γ_2. Note that $\gamma_1 + \gamma_2 = 1$, so that the sum of the two link lengths is the original element length: $\gamma_1 l + \gamma_2 l = l$. The recipe gives these length coefficients as

$$\gamma_1 = 0.2654; \qquad \gamma_2 = 0.7346 \tag{3.1}$$

The recipe gives the stiffness value k for the torsional spring by the formula

$$k = \gamma_2 \kappa \frac{EJ}{l}; \qquad \kappa = 2.0643 \tag{3.2}$$

where E is Young's modulus of elasticity for the desired material, and J is the area moment of inertia for the cross-sectional geometry in the direction of bending (not to be confused with the 'mass moment of inertia' from rigid-body dynamics). Formulas of J for various geometries can be found in standard statics texts, such as [Young and Budnyas 2002]. The constant κ is called the *stiffness coefficient*. Its value in eq. (3.2) is particular to the moment-loaded case, and differs in other recipes.

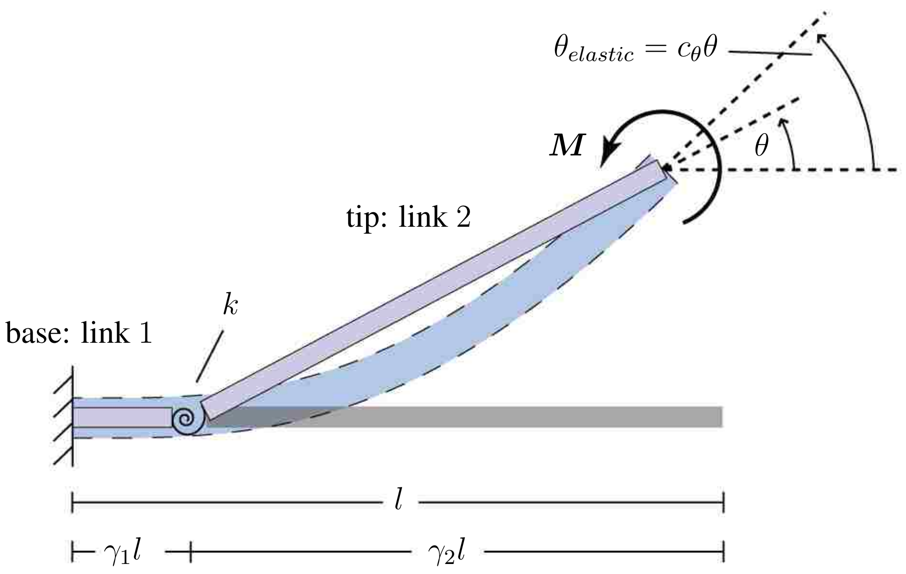


Figure 3.2: *The pure moment-loaded PRBM, comparing the rigid-body's deflected angle to the tip angle of the elastic element. The elastic element is shown with a dashed outline.*

To obtain the static deflection of the elastic element under the load M, Hooke's law $M = k\theta$ is solved for the angle θ. The second rigid link is then rotated by this angle about the spring joint. Under this rotated pose, the end tip of the second rigid link directly gives the location of the elastic element tip. Note however, as illustrated in fig. 3.2, that the tangent line of the second rigid link does not correspond to the orientation of the elastic tip. The recipe therefore supplies the *parametric angle coefficient*, c_θ, which when scaled with the angle of the rigid link's tangent line, gives the angle of the elastic tip's tangent:

$$\theta_{elastic} = c_\theta \theta; \qquad c_\theta = 1.5164 \tag{3.3}$$

Howell [Howell 2001] gives the error of $\theta_{elastic}$ as less than 0.5%, up to a maximum deflection angle of

$$\theta_{elastic}^{max} = 124.4^\circ \tag{3.4}$$

Interestingly, in the particular case of the moment-loaded recipe, it happens that $c_\theta = \gamma_2 \kappa$, offering a slight shortcut for computing k:

$$k = c_\theta \frac{EJ}{l} \tag{3.5}$$

This is not the case in other PRBM recipes.

3.3.2 The planar 3R PRBM for moment and force loads

[Su 2009] introduced a PRBM recipe, with subsequent refinements by [Chen et al. 2011], that will accurately model an elastic beam that is fixed at the base end, with both a force and a moment applied at the other end.

Su's PRBM recipe is called a *3R* PRBM recipe, since the recipe incorporates three revolute joints in contrast to the single revolute joint (*1R*) of many other recipes. Three torsion springs serially connect four rigid links. Three stiffness coefficients and four link length ratios are used to compute the spring stiffness values and link lengths.

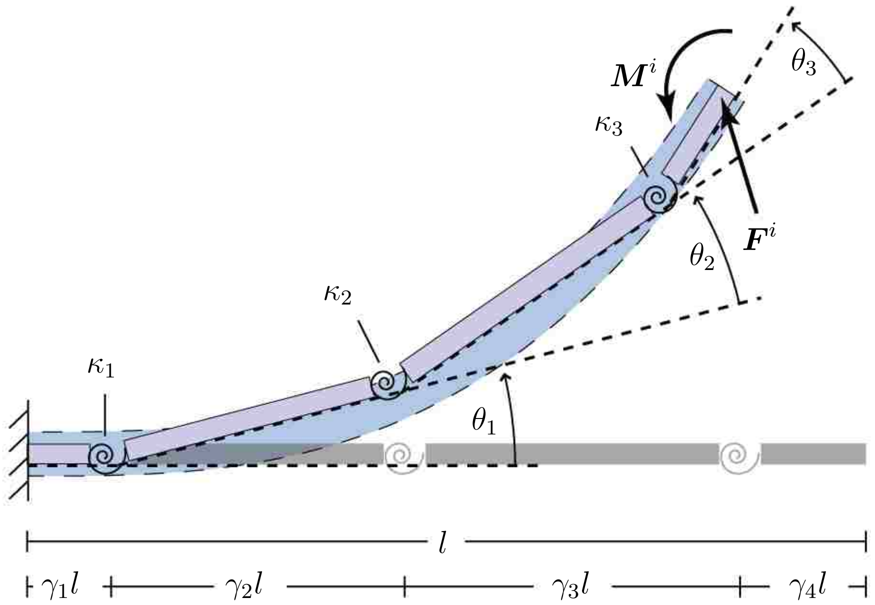


Figure 3.3: *Su's 3R PRBM*

Su gives the following formula for the stiffness value for the m^{th} spring, for $m \in \{1, 2, 3\}$, in terms of the stiffness coefficient κ_m:

$$k_m = \frac{EJ}{l}\kappa_m \tag{3.6}$$

where E is Young's modulus of elasticity and J is the area moment of inertia in the direction of bending, as in the 1R moment-loaded recipe. κ_m is given by [Chen et al. 2011] for all m:

$$\kappa_1 = 3.25 \qquad \kappa_2 = 2.84 \qquad \kappa_3 = 2.95 \tag{3.7}$$

The length coefficients for the rigid links are:

$$\begin{aligned} \gamma_1 &= 0.125\underline{25} \qquad & \gamma_2 &= 0.35\underline{025} \\ \gamma_3 &= 0.388\underline{25} \qquad & \gamma_4 &= 0.136\underline{25} \end{aligned} \tag{3.8}$$

The underlined digits in the above equation represent tweaks we have made to Chen et al.'s values [Chen et al. 2011] so that they sum to 1. Chen et al.'s original values sum only to 0.999—which we suppose to be an artifact of rounding—but we wish these to sum to precisely 1, since they represent weights of the overall element length l, and we wish the link lengths to sum to exactly l.

Unlike the 1R recipes, the 3R recipe requires no parametric angle coefficient, since the correct elastic tip orientation is given directly by the orientation of the fourth link. An important limitation is that the 3R PRBM will not give correct predictions if there is an inflection point[3] in the elastic beam.

3.3.3 The 3D 1R PRBM for a moment load

This section is drawn primarily from [Chase et al. 2011].

Similar to the moment-loaded PRBM recipe, the 3D PRBM recipe prescribes two rigid links, though the spring joint is now configured to allow rotation of the second link in 3D. This enables the beam element to experience two additional deformations, out-of-plane bending and twisting, that are not supported by the planar recipe. The 3D spring joint has three rotational degrees of freedom, each for rotating about a different basis vector from a local, orthonormal coordinate frame. This local frame represents the orientation of the second rigid link, and remains fixed to the link's base point. We choose a right-handed coordinate system, with the x-axis pointing along the length of the element, the y-axis along its height, and the z-axis along its width (see fig. 3.4). Hence, the x-spring

[3]An inflection point occurs anywhere the curvature of the beam changes sign, or in other words, where the curve changes from concave to convex or vice versa.

represents axial twisting of the elastic element, and the y- and z-springs represent the two principle axes of bending.

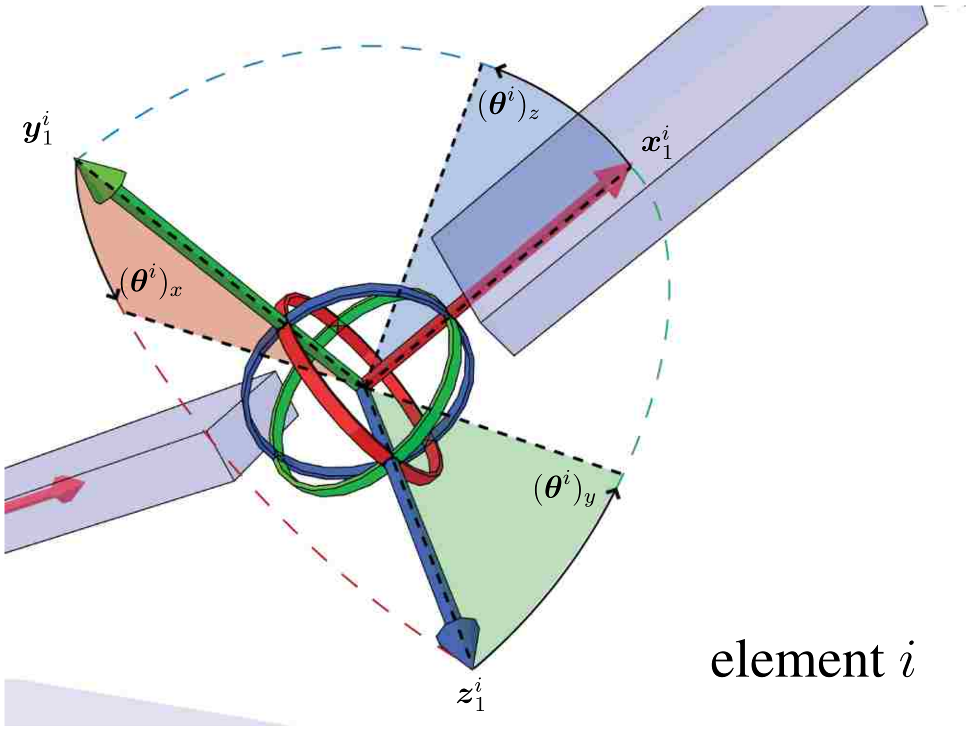

$\boldsymbol{y}_1^i$
$(\boldsymbol{\theta}^i)_z$
$\boldsymbol{x}_1^i$
$(\boldsymbol{\theta}^i)_x$
$(\boldsymbol{\theta}^i)_y$
element i
$\boldsymbol{z}_1^i$

Figure 3.4: *A 3-axis revolute spring joint from Chase's 3D PRBM. See section 3.1 for an explanation of notations.*

Besides the reference frame just mentioned, our implementation tracks two additional reference frames per element compared to [Chase et al. 2011]. In this way, the interfaces of separate elements are decoupled from each other and child elements need not be fixed initially parallel to their parents' tips, since each element now contains the orientation for its own links and elastic tip without needing to refer to frames from neighboring elements.

We place frame 0 at the base point of the first link, and frame 1, as already mentioned, at the base point of the second link. frame 2 is used to track the orientation of the tip of the elastic element being modeled (as distinct from the orientation of the second link), and is placed at the tip of the second link. The local x-, y-, and z-axis basis vector for the m^{th} local frame of element i are denoted $\boldsymbol{x}_m^i$, $\boldsymbol{y}_m^i$, and $\boldsymbol{z}_m^i$, respectively. For the recipe to give correct element deflections, the initial local y- and z-axes of all frames should coincide with the principle axes of the element's cross-sectional geometry, so that the products of inertia for the area moments are zero.

We store the current angular deflection of each torsional spring as $(\theta^i)_x$, $(\theta^i)_y$, and $(\theta^i)_z$, (see fig. 3.4). These angles directly represent how far each spring of the joint has been deflected, and give some indirect, though indeterminate, indication of the orientation of link 2 relative to frame 0. We will revisit this issue of indeterminacy below.

The stiffness values for the springs that rotate about $\boldsymbol{x}_1^i$, $\boldsymbol{y}_1^i$, and $\boldsymbol{z}_1^i$ are set using these formulas:

$$(\boldsymbol{k}^i)_x = \frac{G^i (K^i)_x}{l^i} \tag{3.9a}$$

$$(\boldsymbol{k}^i)_y = c_\theta \frac{E^i (J^i)_y}{l^i} \tag{3.9b}$$

$$(\boldsymbol{k}^i)_z = c_\theta \frac{E^i (J^i)_z}{l^i} \tag{3.9c}$$

Equations (3.9b) and (3.9c) for $(\boldsymbol{k}^i)_y$ and $(\boldsymbol{k}^i)_z$ come from the planar moment-loaded PRBM recipe, with $(J^i)_y$ and $(J^i)_z$ being the area moments of inertia for bending about the local y- and z-axes. E^i is Young's modulus of elasticity for the material of element i, l^i is the desired length of element i, and c_θ is the parametric angle coefficient from the planar recipe (eq. (3.2) in section 3.3.1). As with the planar recipe, $\gamma_2\kappa$ can be substituted for c_θ.

G^i from eq. (3.9a) is the shear modulus for element i's material. This can be found in standard tables, or if the material's Poisson ratio is available, may be computed by the formula $E^i/2\,(1+\nu^i)$ [Chase 2006]. $(K^i)_x$ is a torsion constant dependent on the cross-sectional geometry of the element, and can be found for many common geometries in resources such as [Young and Budnyas 2002, Wikipedia 2013b].

3.4 Algorithms

See section 3.1 for a reminder of the terms and notations used in this section.

For more cohesive class interfaces in an object-oriented implementation, we find it convenient to make a clear separation of concerns between internal element deformations and the accumulations of static loads across the chain. This facilitated our runtime and error experiments involving varied combinations of different forms of the chain algorithm with different constituent element recipes.

As a result, our presentation of the chain algorithm differs somewhat from that of [Chase et al. 2011], since certain portions involving element deformations and transformations are extracted out as element object methods. The extracted methods are called **updateElement** and **transformElement**. These methods vary depending on the particular PRBM recipe, and are given with their respective recipes in section 3.3.3 and section 4.2, apart from the generic chain algorithm in section 3.4.1.

The responsibility of **updateElement** is to compute the deflection of the element's internal links and local coordinate frames given the static load computed by the chain algorithm, and return a transformation matrix representing how all its descendant elements should be transformed. The responsibility of **transformElement** is to take that rigid transformation matrix and apply it uniformly to all the points, links, coordinate frames, etc., of the passed-in element.

3.4.1 The chain calculations

For simplicity of presentation here, we make the assumption that the chain is strictly serial and non-branching, so we may give the algorithm in terms of simple mathematical summations. The extension of these summations to branching chains is straightforward, however, and will be given in chapter 4.

Pseudo-code for the chain algorithm with generic elements is given in algorithm 3.1. We use the index symbol i for the element i being operated on by the outer loop, and j for a descendant element j of element i in the inner loops.

The overall job of the chain algorithm at each step of the outer loop is to compute the *total load* at element i, update its deflection according to that load, then rigidly transform all its descendants to match its new tip location and orientation.

The total load at the tip of an element is the sum of its own *applied load* and its *internal load*. The applied load is the net result of any loads at its tip due to the environment. Its internal load is the sum of the static loads it experiences because of applied loads at its descendants. Separate inner loops are involved in both computing the total load at each element and transforming the element's descendants.

The force component $\boldsymbol{F}^i_\Sigma$ of the total load at element i is given by

$$\boldsymbol{F}^i_\Sigma = \sum_{j=i}^{n} \boldsymbol{F}^j \tag{3.10}$$

This single formula conveniently includes both the applied force and the internal force. Pseudo-code for this computation is found on line 2, as well as line 7 inside the inner loop, of algorithm 3.1.

The total moment $\boldsymbol{M}^i_\Sigma$ at element i has two separable terms, $\boldsymbol{\mu}^i_\Sigma$ and $\boldsymbol{\rho}^i_\Sigma$, so

$$\boldsymbol{M}^i_\Sigma = \boldsymbol{\mu}^i_\Sigma + \boldsymbol{\rho}^i_\Sigma \tag{3.11}$$

$\boldsymbol{\mu}^i_\Sigma$ is the sum of the moments which result from the *moments* $\boldsymbol{M}^j$ applied to each of element i's descendant nodes, and $\boldsymbol{\rho}^i_\Sigma$ is the sum of the static moments which result from the *forces* $\boldsymbol{F}^j$ applied to each of its descendant elements.

For each element j which is a descendant of element i (that is, for all $j > i$, under our non-branching graph assumption), the moment that results at the tip of element i due to the *moment*

Algorithm 3.1 The $\mathcal{O}(n^2)$ form of the generic, element-agnostic chain algorithm, adapted from [Chase et al. 2011, Pauly and Midha 2006]

```
for $i = 1$ to $n$ do

    $\boldsymbol{F}^i_\Sigma \leftarrow \mathbf{0}$
    $\boldsymbol{\mu}^i_\Sigma \leftarrow \mathbf{0}$
    $\boldsymbol{\rho}^i_\Sigma \leftarrow \mathbf{0}$
        // Init. the terms of the total load at element $i$

    for $j = i$ to $n$ do
        $\boldsymbol{F}^i_\Sigma \leftarrow \boldsymbol{F}^i_\Sigma + \boldsymbol{F}^j$
        $\boldsymbol{\mu}^i_\Sigma \leftarrow \boldsymbol{\mu}^i_\Sigma + \boldsymbol{M}^j$
        $\boldsymbol{\rho}^i_\Sigma \leftarrow \boldsymbol{\rho}^i_\Sigma + \left(\boldsymbol{P}^j - \boldsymbol{P}^i\right) \times \boldsymbol{F}^j$
    end for
        // Compute eqs. (3.10), (3.12) and (3.13), the terms of the total load

    $\boldsymbol{M}^i_\Sigma \leftarrow \boldsymbol{\mu}^i_\Sigma + \boldsymbol{\rho}^i_\Sigma$
        // Combine the terms of the total moment
    $\mathrm{R}^i \leftarrow$ updateElement(element $i, \boldsymbol{F}^i_\Sigma, \boldsymbol{M}^i_\Sigma$)
        // Bend element $i$ and get its elastic tip transformation

    for $j = i + 1$ to $n$ do
        transformElement(element $j, \mathrm{R}^i$)
    end for
        // Apply the elastic tip transformation to all descendants

end for
```

$\boldsymbol{M}^j$ applied at element j is simply $\boldsymbol{M}^j$. Thus, the accumulated value $\boldsymbol{\mu}^i_\Sigma$ of these moments is

$$\boldsymbol{\mu}^i_\Sigma = \sum_{j=i}^{n} \boldsymbol{M}^j \tag{3.12}$$

Similar to eq. (3.10), the range of summation in this equation includes the applied moment $\boldsymbol{M}^i$ as well as the terms of the internal moment, so the term $\boldsymbol{\mu}^i_\Sigma$ of the total moment is covered by a single equation. Code for this summation appears on lines 3 and 8 of algorithm 3.1.

For each element j which is a descendant of element i, the moment that results at the tip of element i due to the *force* $\boldsymbol{F}^j$ applied at the tip of element j is computed using the cross-product form of the standard torque equation, with the lever arm as the vector from the tip of element i to the tip of element j. This lever arm is computed with the expression $\boldsymbol{P}^j - \boldsymbol{P}^i$, so the moment is $(\boldsymbol{P}^j - \boldsymbol{P}^i) \times \boldsymbol{F}^j$. Thus, the accumulated value $\boldsymbol{\rho}^i_\Sigma$ of these moments is

$$\boldsymbol{\rho}^i_\Sigma = \sum_{j=i}^{n} (\boldsymbol{P}^j - \boldsymbol{P}^i) \times \boldsymbol{F}^j \tag{3.13}$$

Just like eqs. (3.10) and (3.12), this summation range includes element i itself. Though technically the torque for the index i is not part of the internal moment, it comes out to zero since $\boldsymbol{P}^i - \boldsymbol{P}^i = \boldsymbol{0}$, so including it causes no error and in fact simplifies implementation a bit. The code for this last summation appears on lines 4 and 9 of algorithm 3.1.

Figure 3.5 visually depicts the computation of eq. (3.13) for $\boldsymbol{\rho}^1_\Sigma$ and $\boldsymbol{\rho}^2_\Sigma$ in the left and right panels, respectively. These two computations occur during two separate steps of the chain algorithm's outer loop.

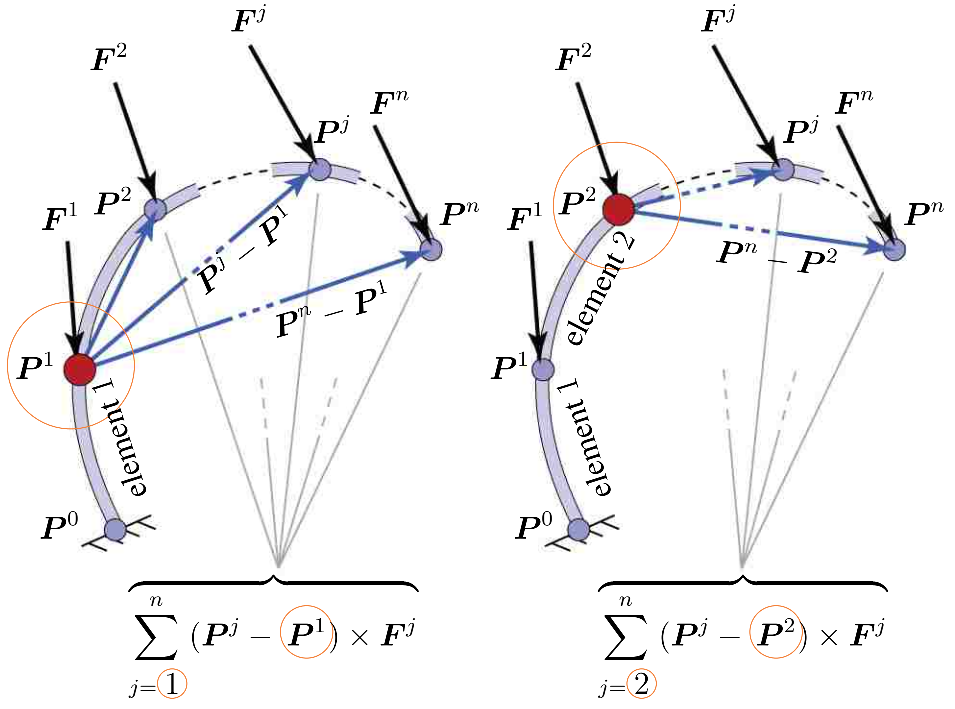


Figure 3.5: *The process of computing and summing each term of the total moment at element* 1 *(left) and element* 2 *(right), respectively, which arises from forces applied at their respective descendent nodes.*

The left panel of fig. 3.5 depicts traversal of the inner loop with element 1 as the parent element (meaning the outer loop is at element 1). The right panel is similar, with element 2 as the parent element. In each, the parent element is circled and filled in red. The lever arm from the parent to each of its descendants is depicted with a blue arrow, and the force applied at each descendant with a black arrow. The inner loop for computing the moment sum begins at the parent element and traverses towards the leaf elements, computing each torque and adding as it goes.

Finally, the two terms $\boldsymbol{\mu}^i_\Sigma$ and $\boldsymbol{\rho}^i_\Sigma$ of the total moment $\boldsymbol{M}^i_\Sigma$ for element i are combined at line 12 of algorithm 3.1. The element is then deflected according to the total load $\boldsymbol{F}^i_\Sigma$ and $\boldsymbol{M}^i_\Sigma$ at line 14 by calling **updateElement**, which returns the transformation matrix R^i_Σ. R^i_Σ is then applied to each descendent element j (though not to element i itself) in an inner loop on lines 16-18, so that they are rigidly transformed to match element i's new tip location and orientation.

3.4.2 updateElement for the 3D 1R PRBM

[Chase et al. 2011] gives a unified presentation of individual element deflections and chain calculations under a single algorithm. We differ in our presentation by factoring out the element deflection steps from the chain algorithm as a new method we call **updateElement**. Decoupling **updateElement** from the chain algorithm with a clean interface permits us to define **updateElement** polymorphically, with different versions for the deflection computations required by different PRBM recipes. The chain algorithm may then be adjusted trivially for different PRBM types simply by calling the appropriate version of **updateElement**.

updateElement interfaces with the chain algorithm by receiving a total load as a function argument alongside the element in question, and by returning to the chain algorithm a transformation matrix. The returned matrix is formed by tracking and composing all the transformations that the element tip experiences during computation of the deflection. Thus, when **updateElement** returns this cumulative transformation matrix, the chain algorithm may then correctly update all the element's child elements.

In this section, we present the version of **updateElement** for Chase et al.'s 3D 1R PRBM. The algorithm works on each of the three local element axes in turn. For the current axis being operated on, it computes an angle of rotation from the static spring equation and the projection of the total moment $\boldsymbol{M}_{\Sigma}^{i}$ onto that axis. The element tip point is rotated around that axis by the computed angle, with the base point of the second link as the center of rotation. The other two axes are then rotated similarly. The algorithm then moves on to operating on each of the next two axes in the same way.

The matrix R^{i} representing the overall tip transformation, which will be returned to the chain algorithm, is initialized to the identity matrix I on line 1 of algorithm 3.2.

Lines 3-6 are concerned with preparing a separate world-space rigid rotation matrix $(\mathrm{R}_1)_{u}^{\mathrm{link}}$, to be used in rotating the second rigid link of the element about the current axis $\boldsymbol{u}$. The angle of

Algorithm 3.2 updateElement for an element i from Chase et al.'s 3D PRBM recipe

```
$\mathrm{R}^i \leftarrow \mathrm{I}$

for each local axis $\boldsymbol{u} \in \{\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}\}$ do

    $(\boldsymbol{\theta}^i)_u \leftarrow \left(\boldsymbol{M}^i_\Sigma \cdot \boldsymbol{u}^i_1\right) / (\boldsymbol{k}^i)_u$
    $(\Delta\boldsymbol{\theta}_1)^{\text{link}}_u \leftarrow (\boldsymbol{\theta}^i)_u - (\boldsymbol{\theta}^i)^{\text{old}}_u$
    $(\mathrm{R}_1)^{\text{link}}_u \leftarrow \text{rotationMatrix}\begin{pmatrix} \text{axis} = \boldsymbol{u}^i_1 \\ \text{angle} = (\Delta\boldsymbol{\theta}_1)_u \\ \text{origin} = \boldsymbol{P}^i_1 \end{pmatrix}$
        // Rotation for second rigid link

    if $\boldsymbol{u}$ is $\boldsymbol{x}$ then
        $(\mathrm{R}_2)^{\text{tip}}_u \leftarrow \mathrm{I}$
            // No additional elastic tip rotation for $\boldsymbol{x}$
    else
        $(\Delta\boldsymbol{\theta}_2)^{\text{tip}}_u \leftarrow c_\theta(\Delta\boldsymbol{\theta}_1)^{\text{link}}_u - (\Delta\boldsymbol{\theta}_1)^{\text{link}}_u$
        $(\mathrm{R}_2)^{\text{tip}}_u \leftarrow \text{rotationMatrix}\begin{pmatrix} \text{axis} = \boldsymbol{u}^i_1 \\ \text{angle} = (\Delta\boldsymbol{\theta}_2)^{\text{tip}}_u \\ \text{origin} = \boldsymbol{P}^i_2 \end{pmatrix}$
            // Additional rotation of elastic tip for $\boldsymbol{y}$ and $\boldsymbol{z}$
    end if

    $\boldsymbol{P}^i_2 \leftarrow (\mathrm{R}_1)^{\text{link}}_u \boldsymbol{P}^i_2$
        // Rotate rigid tip point of second link
    for $\boldsymbol{v}$ as each of the other two, non-$\boldsymbol{u}$ axes, do
        $\boldsymbol{v}^i_1 \leftarrow (\mathrm{R}_1)^{\text{link}}_u \boldsymbol{v}^i_1$
            // Rotate local frame for second link
        $\boldsymbol{v}^i_2 \leftarrow (\mathrm{R}_1)^{\text{link}}_u (\mathrm{R}_2)^{\text{tip}}_u \boldsymbol{v}^i_2$
            // Rotate local frame for elastic tip orientation
    end for

    $\mathrm{R}^i \leftarrow (\mathrm{R}_1)^{\text{link}}_u (\mathrm{R}_2)^{\text{tip}}_u \mathrm{R}^i$
        // Accumulate elastic tip transformations
end for

return $\mathrm{R}^i$
```

rotation $(\boldsymbol{\theta}^i)_u$ is computed in line 3, by solving Hooke's spring equation with the stiffness value $(\boldsymbol{k}^i)_u$ of the spring for the current axis, which was computed in eq. (3.9); and the projected moment $\boldsymbol{M}^i_\Sigma \cdot \boldsymbol{u}^i_1$, which is the dot product of the total moment and the current spring axis vector. We will rotate only by the difference between the currently computed angle $(\boldsymbol{\theta}^i)_u$ and its value $(\boldsymbol{\theta}^i)_u^{\text{old}}$ from the previous chain iteration, so we compute this on line 4 and call it $(\Delta\boldsymbol{\theta}_1)_u^{\text{link}}$. Finally, on line 5, $(\mathrm{R}_1)_u^{\text{link}}$ is set as a rotation matrix for rotation about the spring vector $\boldsymbol{u}^i_1$ for the current axis, by the difference angle $(\Delta\boldsymbol{\theta}_1)_u^{\text{link}}$, with the location $\boldsymbol{P}^i_1$ of the spring joint as the center of rotation.

Lines 7-14 compute an additional world-space matrix, $(\mathrm{R}_2)_u^{\text{tip}}$, for orienting the elastic tip of the element and for conveying additional information about how descendant elements should be transformed. When $\boldsymbol{x}$ is the current axis, no additional rotation is required beyond what $(\mathrm{R}_1)_x^{\text{link}}$ already provides, so in that case the "additional" rotation $(\mathrm{R}_2)_x^{\text{tip}}$ is simply set to the identity matrix I on line 8.

Some additional transformation is required at the elastic element tip when $\boldsymbol{y}$ or $\boldsymbol{z}$ are the current operating axes, however, since the action of the springs for those axes is taken from the original planar moment-loaded PRBM recipe, which requires scaling the rigid rotation of the second link by the parametric angle coefficient c_θ to obtain the proper elastic tip orientation.

Thus, when the current axis $\boldsymbol{u}$ represents $\boldsymbol{y}$ or $\boldsymbol{z}$, an additional angle of rotation $(\Delta\boldsymbol{\theta}_2)_u^{\text{tip}}$ is computed on line 11. Since $(\mathrm{R}_1)_u^{\text{link}}$ already provides the amount of rotation that comes before scaling by c_θ, we don't use the full scaled difference angle $c_\theta(\Delta\boldsymbol{\theta}_1)_u^{\text{link}}$, but only its difference with the unscaled difference angle $(\Delta\boldsymbol{\theta}_1)_u^{\text{link}}$. This is then used on line 12 as the angle of rotation for the additional rotation matrix $(\mathrm{R}_2)_u^{\text{tip}}$, for rotation around the current spring axis vector $\boldsymbol{u}^i_1$, with the element's tip point $\boldsymbol{P}^i_2$ as the center of rotation.

The rigid rotation is applied in lines 15-19 by multiplying $(\mathrm{R}_1)_u^{\text{link}}$ with the other two local axis vectors of frame 1 and with the element's tip point $\boldsymbol{P}^i_2$. On line 21, the elastic tip orientation at frame 2 is updated by both the rigid transformation $(\mathrm{R}_1)_u^{\text{link}}$ and the additional elastic tip transformation $(\mathrm{R}_2)_u^{\text{tip}}$.

The accumulated descendant transformation R^i is updated on line 24 before moving on to the next axis of operation. It is finally returned to the chain algorithm on line 26, after all three local axes have been operated on.

The order in which we apply the rigid link transformation $(\mathrm{R}_1)_u^{\text{link}}$ and the elastic tip transformation $(\mathrm{R}_2)_u^{\text{tip}}$ on lines 20 and 23 is important. The order of transformation is read right to left, so we are first applying $(\mathrm{R}_2)_u^{\text{tip}}$ then $(\mathrm{R}_1)_u^{\text{link}}$. This is important since $(\mathrm{R}_2)_u^{\text{tip}}$ was formed in lines 7-14 for rotation around the old location of the element tip $\boldsymbol{P}_2^i$, *before* line 15 where the element tip $\boldsymbol{P}_2^i$ receives the actual transformation. So the effect of this order of transformations, when it is eventually applied by the chain algorithm to descendant elements, is to first hold still the tip of the parent element i and rotate the descendants around it, then to rotate the descendants together with the tip around the parent element i's spring joint at $\boldsymbol{P}_1^i$. If the transformations were applied in the opposite order, the descendants would first be rotated together with the parent's tip by $(\mathrm{R}_1)_u^{\text{link}}$ around the spring pivot point $\boldsymbol{P}_1^i$, but then would be transformed by $(\mathrm{R}_2)_u^{\text{tip}}$ around the location where the parent tip point *used* to be, so it would now pivot around a point other than the current location of the parent tip. This would incorrectly disconnect descendants from the parent's tip, introducing gaps between chain elements.

Note that any order of the axes of operation for the loop at line 2 could have been chosen, for example $\{\boldsymbol{z}, \boldsymbol{y}, \boldsymbol{x}\}$ or $\{\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{z}\}$. This would have led to a different overall order of rotations, and since rotations of this sort are non-commutative, would have given a different result. Thus, this element update procedure does not give a uniquely determined deflection for a given load. Chase et al. [Chase et al. 2011] note, however, that since we assume that elements have very short length, the rotation error will be small. In practice, these errors converge to zero under iteration.

3.4.3 transformElement for the 3D 1R PRBM

Defining a polymorphic **transformElement** method with different versions for each PRBM recipe is another essential part of decoupling PRBM deflection computations from the chain algorithm.

Thus, this section provides a version of **transformElement** for Chase et al.'s 3D 1R PRBM [Chase et al. 2011], in algorithm 3.3. **transformElement** transforms the entire element rigidly with no internal deflection or deformation using a transformation matrix that would have been returned previously from **updateElement**.

Algorithm 3.3 **transformElement** for an element j from Chase et al.'s 3D PRBM recipe

```
1 for $m = 0$ to $2$ do
2     $\boldsymbol{P}_m^j \leftarrow \mathrm{R}^i \boldsymbol{P}_m^j$
3     for each local axis $\boldsymbol{u} \in \{\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}\}$ do
4         $\boldsymbol{u}_m^j \leftarrow \mathrm{R}^i \boldsymbol{u}_m^j$
5     end for
6 end for
```

Care should be taken when transforming point locations and local basis vectors, since the transformation matrix R^i passed in from the chain algorithm includes translations as well as rotations[4]. The points $\boldsymbol{P}_m^i$ should be affected by these translations, since they represent locations in space, whereas the basis vectors $\boldsymbol{u}_m^i$ should not, since they only represent directions. In our own implementation, we accomplished this distinction by using 4x4 transformation matrices and 4-dimensional vectors with homogenous coordinates. We gave a 1 for the w-coordinate in points, and a 0 for the w-coordinate in basis vectors.

[4]Translations will show up in any matrix that rotates around points other than the origin, as these matrices do, since the effect of such a rotation is to first translate the center of rotation to the origin, apply the rotation, then translate this "temporary origin" back to the center of rotation.

Chapter 4

Methods

This chapter explains our extensions to the 3D PRBM chain algorithm of [Chase et al. 2011]. Pseudo-code for our $\mathcal{O}(n)$ optimization of the chain algorithm is listed near the beginning of this chapter as algorithm 4.1 for convenient reference, and is derived in section 4.1. Section 4.2 extends the combined-load 3R PRBM recipe of [Su 2009, Chen et al. 2011] to a 3D recipe that supports out-of-plane bending and twisting. An explanation of the terms and notations used in the sections and algorithms can be found in section 3.1.

Thanks to our previous refactoring of the chain algorithm into the chain calculations of algorithm 3.1 in section 3.4.1 and the polymorphic element methods **updateElement** and **transformElement**, constructing a chain algorithm with a different type of element is simply a matter of substituting **updateElement** and **transformElement** from a different PRBM recipe. Thus, a new 3D 3R PRBM chain algorithm can be created by combining the new 3D 3R recipe from section 4.2 and its associated **updateElement** (algorithm 4.2) and **transformElement** (algorithm 4.3) methods with either the unoptimized chain algorithm in algorithm 3.1 or the optimized algorithm in algorithm 4.1.

Algorithm 4.1 The optimized chain algorithm with $\mathcal{O}(n)$ runtime complexity

```
$\boldsymbol{F}^n_\Sigma \leftarrow \boldsymbol{F}^n$
$\boldsymbol{\mu}^n_\Sigma \leftarrow \boldsymbol{M}^n$
**for** each index pair $\alpha, \beta$ in a 4x4 matrix **do**
    $\boldsymbol{\sigma}^n_{\alpha\beta} \leftarrow [\boldsymbol{F}^n]^T_\times \mathrm{E}_{\alpha\beta} \boldsymbol{P}^n$
**end for**
    *// Initialize accumulated values at "leaf" elements*

**for** each element $i = n-1$ **to** $1$ **do**
    $\boldsymbol{F}^i_\Sigma \leftarrow \boldsymbol{F}^{i+1}_\Sigma + \boldsymbol{F}^i$
    $\boldsymbol{\mu}^i_\Sigma \leftarrow \boldsymbol{\mu}^{i+1}_\Sigma + \boldsymbol{M}^i$
    **for** each index pair $\alpha, \beta$ **do**
        $\boldsymbol{\sigma}^i_{\alpha\beta} \leftarrow \boldsymbol{\sigma}^{i+1}_{\alpha\beta} + [\boldsymbol{F}^i]^T_\times \mathrm{E}_{\alpha\beta} \boldsymbol{P}^i$
    **end for**
**end for**
    *// Pre-compute accumulated values at "non-leaf" elements*

$\mathrm{R}^1_\Sigma \leftarrow \mathrm{I}$
    *// First accumulated transformation is no transformation*

**for** each element $i = 1$ **to** $n$ **do**
    **transformElement**(element $i, \mathrm{R}^i_\Sigma$)
        *// Apply the accumulated transformation to this element*

    $\boldsymbol{a}^i_\Sigma \leftarrow \mathbf{0}$
    **for** each axis pair $\alpha, \beta$ **do**
        $\boldsymbol{a}^i_\Sigma \leftarrow \boldsymbol{a}^i_\Sigma + r^i_{\alpha\beta} \boldsymbol{\sigma}^i_{\alpha\beta}$
    **end for**
        *// Compute $\boldsymbol{a}^i_\Sigma$, eq.* (4.8)

    $\boldsymbol{b}^i_\Sigma \leftarrow \left(\mathrm{R}^i_\Sigma \boldsymbol{P}^i\right) \times \boldsymbol{F}^i_\Sigma$
        *// Compute $\boldsymbol{b}^i_\Sigma$, eq.* (4.5)

    $\boldsymbol{M}^i_\Sigma \leftarrow \boldsymbol{a}^i_\Sigma - \boldsymbol{b}^i_\Sigma + \boldsymbol{\mu}^i_\Sigma$
        *// The final total moment at element $i$*

    $\mathrm{R}^i \leftarrow$ **updateElement**(element $i, \boldsymbol{F}^i_\Sigma, \boldsymbol{M}^i_\Sigma$)
    **if** $i \neq n$ **then**
        $\mathrm{R}^{i+1}_\Sigma \leftarrow \mathrm{R}^i \mathrm{R}^i_\Sigma$
    **end if**
        *// Accumulate the transformation for the next*
        *// element $i+1$ (unless there is no next element)*
**end for**
```

4.1 Linear-time optimization of the chain algorithm

We convert the $\mathcal{O}(n^2)$ chain algorithm shown in algorithm 3.1 to the $\mathcal{O}(n)$ form of algorithm 4.1 by computing eq. (3.10), eq. (3.12), and eq. (3.13) for all i in linear time. This is done by converting the inner loops of the $\mathcal{O}(n^2)$ algorithm to a series of fast linear-time pre-computations that can be updated in constant time during each step of the outer loop, while yielding the same results. The terms $\boldsymbol{F}^i_\Sigma$ from eq. (3.10) and $\boldsymbol{\mu}^i_\Sigma$ from eq. (3.12) are simpler to convert than $\boldsymbol{\rho}^i_\Sigma$ from eq. (3.13). The conversions of $\boldsymbol{F}^i_\Sigma$ and $\boldsymbol{\mu}^i_\Sigma$ are detailed in section 4.1.1, and that of $\boldsymbol{\rho}^i_\Sigma$ in section 4.1.2.

4.1.1 The simple bit: $\boldsymbol{F}^i_\Sigma$ and $\boldsymbol{\mu}^i_\Sigma$

We begin with the straightforward portion of the optimization which will introduce the form of the other portions.

Recall eq. (3.10) for the total force $\boldsymbol{F}^i_\Sigma$ at the tip of element i, and the term $\boldsymbol{\mu}^i_\Sigma$ from eqs. (3.11) and (3.12), which represents the portion of the internal moment at element i that results from moments applied to element i and its descendent elements:

$$\boldsymbol{F}^i_\Sigma = \sum_{j=i}^{n} \boldsymbol{F}^j; \qquad \boldsymbol{\mu}^i_\Sigma = \sum_{j=i}^{n} \boldsymbol{M}^j$$

In the static kinematic context of the chain algorithm, applied loads never vary from one step of the outer iteration to the next, so the value of $\boldsymbol{F}^j$ for a given j is constant regardless of the value of i. Thus, for any given i, it is true that $\sum_{j=i}^{n} \boldsymbol{F}^j = \boldsymbol{F}^i + \sum_{j=i+1}^{n} \boldsymbol{F}^j$. Using this recursive formulation, we can pre-compute $\boldsymbol{F}^i_\Sigma$ for all i in a single linear-time scan from n to 1. As we visit each element, we add its applied force to the sum we had stored at the previous element, and store the new sum at the current element. We show the initialization of this computation on line 1 of algorithm 4.1, and the pre-computation for all i on line 8, inside the pre-computation loop. This scan need only be done once, ahead of time, before entering the chain algorithm's main *outer loop*, which appears on lines 17-35 of algorithm 4.1.

The outer loop is so called because it serves the same general function as the outer loop from the old $\mathcal{O}(n^2)$ algorithm, though the optimized version no longer has a corresponding *inner loop*.

After entering the outer loop, each step of the loop simply looks up the value of $\boldsymbol{F}^i_\Sigma$ when it is needed. The old formulation of the chain algorithm was wasteful because it did not store and make use of previously computed values in this way—it repeated all but one of the same addition operations from one step of the outer loop to the next.

$\boldsymbol{\mu}^i_\Sigma$ can be pre-computed for all i using this same idea. This is accomplished on lines 2 and 9 of algorithm 4.1.

Each of these recursive, cumulative sums is an example of a *vector scan* (to borrow the terminology of [Harris et al. 2007]), which is the same as a *vector reduction*, with the difference that intermediate results are kept stored, in order, rather than discarded. In the case of $\boldsymbol{F}^i_\Sigma$ and $\boldsymbol{\mu}^i_\Sigma$, we are performing a *suffix scan* over a vector containing all of the applied forces in order, with addition as the scan operator.

Interestingly, while vector scans can be computed in $\mathcal{O}(n)$ time as a simple serial algorithm, they have also been studied as a fundamental operator in parallel computing, and very efficient implementations have even been realized for the GPU [Harris et al. 2007].

4.1.2 The not-so-simple bit: $\boldsymbol{\rho}^i_\Sigma$

Optimizing the second term from eq. (3.11), $\boldsymbol{\rho}^i_\Sigma$, is less straightforward. Consider again the equation for this term,

$$\boldsymbol{\rho}^i_\Sigma = \sum_{j=i}^{n} \left(\boldsymbol{P}^j - \boldsymbol{P}^i\right) \times \boldsymbol{F}^j \tag{4.1}$$

Just as for $\boldsymbol{\mu}^i_\Sigma$, the $\mathcal{O}(n^2)$ implementation of the chain algorithm computed this $\mathcal{O}(n)$ expression n times—once for each $i \in [1, n]$—giving rise to its $\mathcal{O}(n^2)$ complexity. Our next step in re-factoring the old chain algorithm hinges on a particular way of breaking apart and re-writing this equation in terms of stored, pre-computed values, with an update step that can be performed in constant time during each step of the new chain algorithm's outer loop. Suffix vector scans will also play a role in this optimization, though some additional manipulations are necessary.

Equation (4.1), as originally presented in the old chain calculations of section 3.4.1, must be interpreted according to the algorithm's current state—that is, it only applies in the context of the current step i. At each step i, the points $\left\{\boldsymbol{P}^i, \ldots, \boldsymbol{P}^n\right\}$ will have been transformed during the previous step $i-1$. This means that the locations represented by the points $\left\{\boldsymbol{P}^i, \ldots, \boldsymbol{P}^n\right\}$ *all* vary in i, not just $\boldsymbol{P}^i$ itself—in other words, if the set of point locations A equals $\left\{\boldsymbol{P}^{i+1}, \ldots, \boldsymbol{P}^n\right\}$ during step i of the outer loop, and the set B equals $\left\{\boldsymbol{P}^{i+1}, \ldots, \boldsymbol{P}^n\right\}$ during step $i+1$ of the outer loop, then $A \neq B$, even though the sets cover the same indices. Thus the actual locations $\left\{\boldsymbol{P}^i, \ldots, \boldsymbol{P}^n\right\}$ in the summation expression of eq. (4.1) will depend on the range of summation, so it is not appropriate to directly compare or relate a summation having one range to a summation having another. For a contrasting example, in optimizing the computation of $\boldsymbol{\mu}^i_\Sigma$, we could rely on the fact that the value of $\boldsymbol{M}^j$ never varied with i, so it was correct to observe that $\sum_{j=i}^n \boldsymbol{M}^j = \boldsymbol{M}^i + \sum_{j=i+1}^n \boldsymbol{M}^j$. However, A similar relationship would not be true for the summations $\sum_{j=i}^n (\boldsymbol{P}^j - \boldsymbol{P}^i) \times \boldsymbol{F}^j$ and $\sum_{j=i+1}^n (\boldsymbol{P}^j - \boldsymbol{P}^{i+1}) \times \boldsymbol{F}^j$, since even when j from the first is equal to j from the second, $\boldsymbol{P}^j$ from the first is not equal to $\boldsymbol{P}^j$ from the second.

Just like the other optimizations, our optimization of eq. (4.1) depends on relating summations of different ranges, so we first wish to re-write eq. (4.1) with the points in $\left\{\boldsymbol{P}^i, \ldots, \boldsymbol{P}^n\right\}$ as constants in i, so that in and of themselves they only represent the chain's initial configuration, and not their latest updated positions, regardless of the range of summation. This is done by making their updated transformations explicit, by introducing a transformation matrix R^i_Σ and left-multiplying it with the now-constant points, then replacing the points in the original eq. (4.1) with this matrix-point product. R^i_Σ represents the cumulative transformation that all the points $\left\{\boldsymbol{P}^i, \ldots, \boldsymbol{P}^n\right\}$ in the summation range have undergone as a result of steps 1 through $i-1$ of the old chain algorithm's outer loop. Re-writing eq. (4.1) in this way now frees all the quantities except

R_{Σ}^{i} from the context of the algorithm's outer loop over i:

$$\boldsymbol{\rho}_{\Sigma}^{i} = \sum_{j=i}^{n} \left(\mathrm{R}_{\Sigma}^{i}\boldsymbol{P}^{j} - \mathrm{R}_{\Sigma}^{i}\boldsymbol{P}^{i}\right) \times \boldsymbol{F}^{j} \tag{4.2}$$

From eq. (4.2) on, the symbols $\left\{\boldsymbol{P}^{1}, \ldots, \boldsymbol{P}^{n}\right\}$ will be interpreted to represent constant locations through the duration of the algorithm, and the matrix R_{Σ}^{i} will explicitly represent their most current transformation.

To summarize, this new eq. (4.2) offers the property that even in two different contexts, that is, for different values of i, if j from the first context is equal to j from the second, then $\boldsymbol{P}^{j}$ in the first context is guaranteed to equal $\boldsymbol{P}^{j}$ from the second. So, it finally becomes appropriate to manipulate and compare summations of different ranges, so long as appropriate care is taken with R_{Σ}^{i} which depends on i.

R_{Σ}^{i} is the result of successive application of individual transformation matrices, each resulting from a single call to **updateElement** during a separate previous step of the old outer loop. Denoting these matrices in order as $\{\mathrm{R}^{1}, \ldots, \mathrm{R}^{i-1}\}$, and recalling that applying a transformation matrix means multiplying on the left,

$$\mathrm{R}_{\Sigma}^{i} = \mathrm{R}^{i-1}\mathrm{R}^{i-2} \cdots \mathrm{R}^{1} \tag{4.3}$$

Note that R_{Σ}^{i} can be computed with a single matrix multiplication if we already have R^{i-1} and $\mathrm{R}_{\Sigma}^{i-1}$, since by this recursive formulation, $\mathrm{R}_{\Sigma}^{i} = \mathrm{R}^{i-1}\mathrm{R}_{\Sigma}^{i-1}$.

Equation (4.2) can be factored and distributed as shown, using the fact that both matrix multiplication and application of the cross product distribute over column vector addition:

$$\begin{aligned}\boldsymbol{\rho}_{\Sigma}^{i} &= \sum_{j=i}^{n} \left(\mathrm{R}_{\Sigma}^{i}\boldsymbol{P}^{j}\right) \times \boldsymbol{F}^{j} - \sum_{j=i}^{n} \left(\mathrm{R}_{\Sigma}^{i}\boldsymbol{P}^{i}\right) \times \boldsymbol{F}^{j} \\ &= \sum_{j=i}^{n} \left(\mathrm{R}_{\Sigma}^{i}\boldsymbol{P}^{j}\right) \times \boldsymbol{F}^{j} - \left(\mathrm{R}_{\Sigma}^{i}\boldsymbol{P}^{i}\right) \times \left(\sum_{j=i}^{n} \boldsymbol{F}^{j}\right)\end{aligned}$$

The factor $\mathrm{R}_{\Sigma}^{i}\boldsymbol{P}^{i}$ may be pulled out of the second summation because it is constant over the index j of summation, and because the cross product with $\boldsymbol{F}^{j}$ obeys the distributive property. Note, however, that though R_{Σ}^{i} is constant over the index of the first summation, we may *not* pull it out of that summation in a similar way, because its matrix multiplication with $\boldsymbol{P}^{j}$ does not obey the associative principle with the subsequent cross product, i.e. $\left(\mathrm{R}_{\Sigma}^{i}\boldsymbol{P}^{j}\right) \times \boldsymbol{F}^{j} \neq \mathrm{R}_{\Sigma}^{i}\left(\boldsymbol{P}^{j} \times \boldsymbol{F}^{j}\right)$. Thus, the matrix multiplication must occur before the cross product, and its result is not constant in j, and clearly neither is $\boldsymbol{F}^{j}$, so at no point in the first summation term is there a simple separable factor that is constant in j.

It will now be convenient to name these two terms:

$$\boldsymbol{a}_{\Sigma}^{i} = \sum_{j=i}^{n} \left(\mathrm{R}_{\Sigma}^{i}\boldsymbol{P}^{j}\right) \times \boldsymbol{F}^{j} \tag{4.4a}$$

$$\boldsymbol{b}_{\Sigma}^{i} = \left(\mathrm{R}_{\Sigma}^{i}\boldsymbol{P}^{i}\right) \times \left(\sum_{j=i}^{n} \boldsymbol{F}^{j}\right) \tag{4.4b}$$

so $\boldsymbol{\rho}_{\Sigma}^{i} = \boldsymbol{a}_{\Sigma}^{i} - \boldsymbol{b}_{\Sigma}^{i}$.

The summation factor of the cross-product in eq. (4.4b) for $\boldsymbol{b}_{\Sigma}^{i}$ is simply $\boldsymbol{F}_{\Sigma}^{i}$ which will already have been pre-computed for all i. Thus, during any given step i of the new algorithm, $\boldsymbol{b}_{\Sigma}^{i}$ may be computed in constant time using R_{Σ}^{i} from the previous algorithm step, by looking up $\boldsymbol{F}_{\Sigma}^{i}$, and computing

$$\boldsymbol{b}_{\Sigma}^{i} = \left(\mathrm{R}_{\Sigma}^{i}\boldsymbol{P}^{i}\right) \times \boldsymbol{F}_{\Sigma}^{i} \tag{4.5}$$

This computation is performed in the pseudo-code on line 25 of algorithm 4.1.

Equation (4.4a) for $\boldsymbol{a}_{\Sigma}^{i}$ may also be computed in constant time, with the right set of pre-computations. The first manipulation of eq. (4.4a) we perform is to re-write the cross product expression as a matrix multiplication. We form the cross-product matrix[5] from the cross-product operand $\boldsymbol{F}^{j}$ rather than than from $\mathrm{R}_{\Sigma}^{i}\boldsymbol{P}^{j}$, because we are preparing a pre-computation that must be independent of the i in R_{Σ}^{i}. $\left(\mathrm{R}_{\Sigma}^{i}\boldsymbol{P}^{j}\right) \times \boldsymbol{F}^{j}$ thus becomes $[\boldsymbol{F}^{j}]_{\times}^{T}\left(\mathrm{R}_{\Sigma}^{i}\boldsymbol{P}^{j}\right)$, so:

$$\boldsymbol{a}_{\Sigma}^{i} = \sum_{j=i}^{n} [\boldsymbol{F}^{j}]_{\times}^{T} \mathrm{R}_{\Sigma}^{i}\boldsymbol{P}^{j} \tag{4.6}$$

Next, we recall that matrices form a vector space over matrix addition and multiplication by a scalar; that is, any matrix may be written as a linear combination of a set of spanning basis matrices. For the sixteen basis matrices of our 4x4 transformations, we choose the natural bases $\mathrm{E}_{\alpha\beta}$ for all index pairs α, β, where the α, β^{th} entry of $\mathrm{E}_{\alpha\beta}$ is 1, and all other entries are 0. Now, denoting the α, β^{th} entry of R_{Σ}^{i} as $r_{\alpha\beta}^{i}$, we may re-write R_{Σ}^{i} as

$$\mathrm{R}_{\Sigma}^{i} = \sum_{\alpha,\beta} r_{\alpha\beta}^{i}\mathrm{E}_{\alpha\beta} \tag{4.7}$$

We now substitute this expression into eq. (4.6) to obtain:

$$\boldsymbol{a}_{\Sigma}^{i} = \sum_{j=i}^{n} [\boldsymbol{F}^{j}]_{\times}^{T} \left(\sum_{\alpha,\beta} r_{\alpha\beta}^{i}\mathrm{E}_{\alpha\beta}\right) \boldsymbol{P}^{j}$$

[5]See appendix B

Now, since neither $[\boldsymbol{F}^j]_\times^T$ nor $\boldsymbol{P}^j$ vary with α or β, they can be moved inside the inner summation. Then, since the entire nested sum is a finite sum, we can switch the order of summation:

$$\begin{aligned} &= \sum_{j=i}^{n} \sum_{\alpha,\beta} [\boldsymbol{F}^j]_\times^T r^i_{\alpha\beta} \mathrm{E}_{\alpha\beta} \boldsymbol{P}^j \\ &= \sum_{\alpha,\beta} \sum_{j=i}^{n} [\boldsymbol{F}^j]_\times^T r^i_{\alpha\beta} \mathrm{E}_{\alpha\beta} \boldsymbol{P}^j \end{aligned}$$

Finally, we note that, since scalar-matrix multiplication with the scalar $r^i_{\alpha\beta}$ is commutative, and since $r^i_{\alpha\beta}$ is invariant over the index of summation j, we can move it to the left of the inner summation expression, then pull it out of the inner summation entirely:

$$\boldsymbol{a}^i_\Sigma = \sum_{\alpha,\beta} r^i_{\alpha\beta} \sum_{j=i}^{n} [\boldsymbol{F}^j]_\times^T \mathrm{E}_{\alpha\beta} \boldsymbol{P}^j$$

Let us give the name $\boldsymbol{\sigma}^i_{\alpha\beta}$ to the inner summation in this new expression for $\boldsymbol{a}^i_\Sigma$, so $\boldsymbol{\sigma}^i_{\alpha\beta} = \sum_{j=i}^{n} [\boldsymbol{F}^j]_\times^T \mathrm{E}_{\alpha\beta} \boldsymbol{P}^j$. Since none of the quantities in this summation vary with i, we can apply to it the same pattern of vector scan as before to obtain $\{\boldsymbol{\sigma}^1_{\alpha\beta}, \ldots, \boldsymbol{\sigma}^n_{\alpha\beta}\}$ for all i. We must do this 16 times, to obtain a separate vector for each index pair α, β of the 4x4 transformation matrix. Similar to the other vector scans, we only need to compute these 16 vectors once, at the start of the algorithm. These pre-computed scans are initialized on lines 3-5 of algorithm 4.1, and completed for all remaining i inside the pre-computation loop, on lines 10-12.

With these 16 pre-computed vectors in hand, we can now compute $\boldsymbol{a}^i_\Sigma$ in constant time given any R^i_Σ with entries $r^i_{\alpha\beta}$:

$$\boldsymbol{a}^i_\Sigma = \sum_{\alpha,\beta} r^i_{\alpha\beta} \boldsymbol{\sigma}^i_{\alpha\beta} \tag{4.8}$$

The computation takes constant time in n because the size of the summation is always fixed at 16 terms (one for each index pair α, β), and because $\boldsymbol{\sigma}^i_{\alpha\beta}$ was computed previously. This is shown on lines 20-24 of algorithm 4.1.

The $\mathcal{O}(n)$ method of computing $\boldsymbol{a}^i_\Sigma$ we have given here is what we implemented for the results given in chapter 5. For interest's sake, in appendix A we also include an alternative linear-time method of computing $\boldsymbol{a}^i_\Sigma$ from eq. (4.4a), for which a concise and presentable formulation emerged only after implementation.

4.1.3 Putting it all together

Having pre-computed $\boldsymbol{\mu}^i_\Sigma$ for all i, and knowing how to compute $\boldsymbol{a}^i_\Sigma$ and $\boldsymbol{b}^i_\Sigma$ for the current element i of the "outer loop" given the cumulative transformation R^i_Σ up to that point, we can finally compute the total moment (line 27 of algorithm 4.1):

$$\boldsymbol{M}^i_\Sigma = \boldsymbol{a}^i_\Sigma - \boldsymbol{b}^i_\Sigma + \boldsymbol{\mu}^i_\Sigma \tag{4.9}$$

With the total moment $\boldsymbol{M}^i_\Sigma$ at element i and the total force $\boldsymbol{F}^i_\Sigma$, we can now call '**update-Element**,' get from it element i's update transformation R^i, and compose its transformation with the accumulated transformation R^{i+1}_Σ for the next element $i+1$. This all takes place on lines 29-34 of algorithm 4.1.

4.1.4 Extension to branching graphs

We summarize our extension of the optimized chain algorithm to tree-like graphs. Our extension is built upon the abstraction of a graph data structure that supports depth-first traversal with both pre-order and post-order visitor operations. We used the Boost Graph Library [Siek et al. 2011] in our implementation, though any graph structure that supports user-defined visitor callbacks will work.

In the graph-based $\mathcal{O}(n)$ algorithm, each instance of a loop over all the elements is replaced with a depth-first search starting from the root-element, with either a pre-order visitor or a post-order visitor. As expected, a pre-order visitor is called upon to operate on an element before any of that element's children, whereas a post-order visitor is called upon only after all the element's children have been visited. Both types of visitor are given two callback methods they may invoke: an *element callback* to operate on the element directly, and an *edge callback* to operate on a single pair of parent-child elements. The pre-order visitor first invokes the element callback, then invokes the edge callback once for each child, with the current parent element paired with each in turn. The post-order visitor first invokes the edge callback for the element paired with each of its child elements, then invokes the element callback for the parent itself.

The pre-computation loop on lines 7-14 of algorithm 4.1 is thus translated into a depth-first search by providing the appropriate element and edge callbacks. Since the loop only performs suffix scans, a post-order visitor is needed. The element callback adds the element's applied force to its total force, and the edge callback adds the child element's total force to the parent element's total force. Each callback does similarly for the applied moment and the 16 $\boldsymbol{\sigma}^i_{\alpha\beta}$ quantities.

The chain loop on lines 17-35 of algorithm 4.1 is translated using a pre-order visitor, since it must propagate the accumulated rigid transformation from the root to the leaves. The element callback does most of the work, since most operations within the loop can be done in place for the current element. The exception comes at line 31, where the updated accumulated transformation matrix is forwarded to each child via the edge callback.

4.1.5 A critical serialization point

The recursive form of eq. (4.3) would seem to show that the vector of cumulative transformation matrices $\{\mathrm{R}^1_\Sigma, \ldots, \mathrm{R}^n_\Sigma\}$ could also result from a vector scan, where the scan operation is left-multiplication by a matrix on the input $\{\mathrm{R}^1, \ldots, \mathrm{R}^n\}$.

Unfortunately, though we previously noted the existence of efficient parallel algorithms for vector scans, there is an important difference which has thus far prevented parallelization of this case. For the vector scan producing $\{\boldsymbol{\mu}_{\Sigma}^{1}, \ldots, \boldsymbol{\mu}_{\Sigma}^{n}\}$, all the entries of the input vector $\left\{\boldsymbol{M}^{1}, \ldots, \boldsymbol{M}^{n}\right\}$ are known independently and in advance, so the scan can be decoupled from the outer loop of the chain algorithm. For the vector of the individual transformation matrices $\{\mathrm{R}^{1}, \ldots, \mathrm{R}^{n}\}$, however, the value of each matrix R^{i} is dependent on the previous matrix R^{i-1} through the chain calculations of step $i-1$. Thus, each matrix in the vector cannot be known independently in advance, so the same method of vector scan cannot apply in advance of the chain loop. This processing dependency appears to create a critical serialization point between each step of the loop.

4.2 The 3D 3R PRBM for force and moment loads

We extend the 3R PRBM recipe of [Su 2009, Chen et al. 2011] (see section 3.3.2) to three dimensions. Like the planar 3R recipe, this recipe accurately models a cantilevered beam element with a combined force-moment load at the free tip. Unlike the planar recipe, it accounts for out-of-plane bending and twisting.

We replace the 1-DOF spring joints of the planar recipe with Chase et al.'s 3-DOF spring joints [Chase et al. 2011]. We attach a local coordinate frame to the base of each link. Three frames service a 3-DOF revolute joint and indicate the orientation of the following rigid link. The remaining frame serves only to orient the base link, with no associated spring joint.

Since the last rigid link of the planar 3R PRBM recipe directly gives the correct elastic tip orientation (see section 3.3.2), we presumed that our 3D 3R recipe would likewise need no additional rotation information for the final tip. The results in chapter 5 affirm that this was a valid assumption. Thus, no additional local frame is needed for the tip of the last link; all the information needed is provided by the frame of the final spring joint.

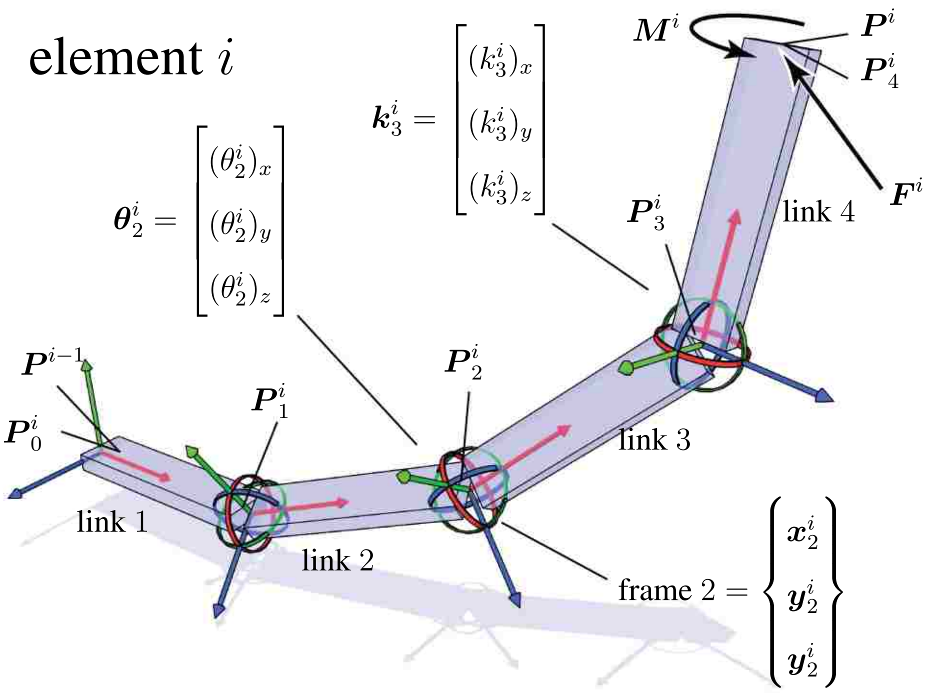

element i
$\boldsymbol{M}^i$
$\boldsymbol{P}^i$
$\boldsymbol{P}^i_4$
$\boldsymbol{k}^i_3 = \begin{bmatrix}(k^i_3)_x \\ (k^i_3)_y \\ (k^i_3)_z\end{bmatrix}$
$\boldsymbol{\theta}^i_2 = \begin{bmatrix}(\theta^i_2)_x \\ (\theta^i_2)_y \\ (\theta^i_2)_z\end{bmatrix}$
$\boldsymbol{F}^i$
$\boldsymbol{P}^i_3$
link 4
$\boldsymbol{P}^{i-1}$
$\boldsymbol{P}^i_2$
$\boldsymbol{P}^i_1$
$\boldsymbol{P}^i_0$
link 3
link 1
link 2
frame 2 = $\begin{Bmatrix}\boldsymbol{x}^i_2 \\ \boldsymbol{y}^i_2 \\ \boldsymbol{y}^i_2\end{Bmatrix}$

Figure 4.1: *Extension of the 3R PRBM to 3D*

Figure 4.1 depicts an example element i for our 3D 3R recipe. As with the planar 3R recipe, the length of the m^{th} link is given by $\gamma_m l^i$, where l^i is the length of the i^{th} element. See eq. (4.10c) for the values of γ_m for all m. At a given joint m, the *angle vector* $\boldsymbol{\theta}^i_m$ stores the current angle of deflection around each local axis in each of its respective components. $\boldsymbol{k}^i_m$ does similarly for the spring stiffness values. The applied force $\boldsymbol{F}^i$ and moment $\boldsymbol{M}^i$ are depicted at the element tip.

Equation (4.10a) gives formulas for setting the stiffness of each of the three springs for joint m of element i, with the necessary constants listed in eqs. (4.10b) to (4.10d). Just like in Chase et al.'s 3D 1R recipe [Chase et al. 2011] (see section 3.3.3), G^i is the shear modulus for the material of element i, E^i is Young's modulus of elasticity for element i, l^i is the element's length, and $(J^i)_y$ and $(J^i)_z$ represent its area moment of inertia for bending up and down and side-to-side, respectively. λ_m is explained below. As in eq. (3.8), the underlined digits in eq. (4.10c) indicate tweaks we've made to to constants from [Chen et al. 2011] so that they sum evenly to 1.

$$
\begin{aligned}
(\boldsymbol{k}^i_m)_x &= \frac{G^i(K^i)_x}{l^i\lambda_m} \\
(\boldsymbol{k}^i_m)_y &= \kappa_m\frac{E^i(J^i)_y}{l^i} \\
(\boldsymbol{k}^i_m)_z &= \kappa_m\frac{E^i(J^i)_z}{l^i}
\end{aligned}
\tag{4.10a}
$$

$$
\kappa_1 = 3.25 \qquad \kappa_2 = 2.84 \qquad \kappa_3 = 2.95 \tag{4.10b}
$$

$$
\begin{aligned}
\gamma_1 &= 0.125\underline{25} \qquad \gamma_2 = 0.35\underline{025} \\
\gamma_3 &= 0.388\underline{25} \qquad \gamma_4 = 0.136\underline{25}
\end{aligned}
\tag{4.10c}
$$

$$
\lambda_m = \frac{\gamma_{m+1}}{\gamma_2+\gamma_3+\gamma_4} \tag{4.10d}
$$

The formulas and constants in eqs. (4.10a) to (4.10c) for the springs at the local y- and z-axes—for bending up and down and bending to the side, respectively—are taken from eqs. (3.6) to (3.8), from the planar 3R PRBM recipe of [Su 2009, Chen et al. 2011].

The stiffness formula and constants in eqs. (4.10a) and (4.10d) for each local x-axis—for twisting at each joint—are designed empirically to provide the same overall twist across the length of the element as in Chase et al.'s 3D 1R recipe [Chase et al. 2011], but with a proportion of the overall twist angle distributed among each of the three joints. The relative twist permitted at joint m is weighted according to the relative length of its following link $m + 1$ by the constant λ_m, which is calculated by eq. (4.10d) as the ratio of the length of link $m + 1$ to the total length of the final three links.

4.2.1 updateElement for the 3D 3R recipe

We provide a version of **updateElement** for the 3D 3R recipe which prepares a 3D 3R element i for use with either version of the element-agnostic chain algorithms (for the unoptimized version, see algorithm 3.1; for the optimized version, see algorithm 4.1). **updateElement** for this recipe applies Chase et al.'s 3D joint update computations from within algorithm 3.2 to each of the three 3D joints in sequence. Since it is based on Su's and Chen et al.'s 3R PRBM [Su 2009, Chase et al. 2011], it also accounts for the total force $\boldsymbol{F}^i_\Sigma$ at the element tip, and computes additional static moment terms for each joint accordingly.

Algorithm 4.2 iterates over all the spring joints of the element, starting near the base and moving towards the tip. It operates on each local axis $\boldsymbol{u}$ in turn for each joint m. The operation on each axis is similar to Chase et al.'s operation for a single axis [Chase et al. 2011]. The loops that perform these nested operations begin on lines 2 and 3.

The most up-to-date moment $\boldsymbol{M}^i_m$ experienced at the current joint m is computed on line 4, taking into account the most recent configuration of the links due to any previous joint/axis operation. The first term of $\boldsymbol{M}^i_m$ is simply the total moment $\boldsymbol{M}^i_\Sigma$ at the element tip. The second term is the moment term due to the total force $\boldsymbol{F}^i_\Sigma$. It comes from the standard torque equation with the lever arm as the vector from the location $\boldsymbol{P}^i_m$ of the current joint m to the element tip $\boldsymbol{P}^i_4$.

On line 6 we compute the current joint and axis angle $(\boldsymbol{\theta}^i_m)_{\boldsymbol{u}}$ by solving Hooke's law, with the joint moment $\boldsymbol{M}^i_m$ from line 4 projected onto the current local axis by the dot product $\boldsymbol{M}^i_m \cdot \boldsymbol{u}^i_m$. On line 7, We compute the difference angle $(\Delta\boldsymbol{\theta}^i_m)_{\boldsymbol{u}}$, which is the difference of $(\boldsymbol{\theta}^i_m)_{\boldsymbol{u}}$ with its value from the previous chain iteration. On line 8, we then form the update matrix $(\mathrm{R}_m)_{\boldsymbol{u}}$, which represents a rotation around the current axis vector $\boldsymbol{u}^i_m$ by the difference angle $(\Delta\boldsymbol{\theta}^i_m)_{\boldsymbol{u}}$ about the joint location $\boldsymbol{P}^i_m$.

The loop on lines 10-17 updates the following link end-points and frame basis vectors using $(\mathrm{R}_m)_{\boldsymbol{u}}$. The point $\boldsymbol{P}^i_m$ at the current joint m needs no update, so it is skipped by indexing points

Algorithm 4.2 updateElement for an element i from the 3D 3R PRBM recipe

```
1  $\mathrm{R}^i \leftarrow \mathrm{I}$

2  for joint index $m = 1$ to $3$ do
3    for each local axis $\boldsymbol{u} \in \{\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}\}$ do

4      $\boldsymbol{M}^i_m \leftarrow \boldsymbol{M}^i_\Sigma + \left(\boldsymbol{P}^i_4 - \boldsymbol{P}^i_m\right) \times \boldsymbol{F}^i_\Sigma$
5        // Torque at joint due to $\boldsymbol{M}^i_\Sigma$ and $\boldsymbol{F}^i_\Sigma$

6      $(\boldsymbol{\theta}^i_m)_{\boldsymbol{u}} \leftarrow \left(\boldsymbol{M}^i_m \cdot \boldsymbol{u}^i_m\right) / (\boldsymbol{k}^i_m)_{\boldsymbol{u}}$
7      $(\Delta\boldsymbol{\theta}^i_m)_{\boldsymbol{u}} \leftarrow (\boldsymbol{\theta}^i_m)_{\boldsymbol{u}} - (\boldsymbol{\theta}^i_m)^{\text{old}}_{\boldsymbol{u}}$
8      $(\mathrm{R}_m)_{\boldsymbol{u}} \leftarrow \text{rotationMatrix}\begin{pmatrix} \text{axis} = \boldsymbol{u}^i_m \\ \text{angle} = (\Delta\boldsymbol{\theta}^i_m)_{\boldsymbol{u}} \\ \text{origin} = \boldsymbol{P}^i_m \end{pmatrix}$
9        // Rotation for following points and local frames

10     for this and each following joint index $s = m$ to $3$ do

11       $\boldsymbol{P}^i_{s+1} \leftarrow (\mathrm{R}_m)_{\boldsymbol{u}} \boldsymbol{P}^i_{s+1}$
12         // Update all following link points

13       for each local axis $\boldsymbol{v} \in \{\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}\}$ do
14         $\boldsymbol{v}^i_s \leftarrow (\mathrm{R}_m)_{\boldsymbol{u}} \boldsymbol{v}^i_s$
15       end for
16         // Update this and all following local frames

17     end for

18     $\mathrm{R}^i \leftarrow (\mathrm{R}_m)_{\boldsymbol{u}} \mathrm{R}^i$
19       // Accumulate all rotations for descendant elements

20   end for
21 end for

22 return $\mathrm{R}^i$
```

inside the loop with $s+1$. However, the axis vectors at the current joint m *should* be included—or else the projection expression $\boldsymbol{M}^i_m \cdot \boldsymbol{u}^i_m$ on line 6 will not use the most up-to-date local vector—so the index s is used for basis vectors.

Finally on line 18, we compose the most recent rotation $(\mathrm{R}_m)_{\boldsymbol{u}}$ into the cumulative effect of R^i. R^i is returned after all the joint/axis operations have completed, so that the chain algorithm may propagate the total element deflection on to all the descendants of element i.

4.2.2 transformElement for the 3D 3R recipe

We give pseudo-code in algorithm 4.3 for **transformElement** for the 3D 3R recipe. It applies the transformation matrix R^i, as passed in from the chain algorithm, to all the points and local frame basis vectors of the element j for which it is run.

Algorithm 4.3 transformElement for an element i from the 3D 3R PRBM recipe

```
 1  for each point index $m = 0$ to $4$ do
 2    $\boldsymbol{P}^i_m \leftarrow \mathrm{R}^i \boldsymbol{P}^i_m$
 3      // Transform all points
 4  end for

 5  for each local frame index $m = 0$ to $3$ do
 6    for each local axis $\boldsymbol{u} \in \{\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}\}$ do
 7      $\boldsymbol{u}^i_{\boldsymbol{u}} \leftarrow \mathrm{R}^i \boldsymbol{u}^i_{\boldsymbol{u}}$
 8        // Transform all local frame basis vectors
 9    end for
10  end for
```

We repeat the caution about taking care that points and local basis vectors each be transformed according to their type: points should be affected by the translation operations contained in R^i, whereas basis vectors should only be affected by the rotations.

Chapter 5

Results

Chase et al. verified the correctness of their 3D 1R chain algorithm by comparing chain deflections to those obtained from a finite element analysis of the same chain and applied load [Chase et al. 2011].

Their experiment successfully reproduced *lateral torsional buckling*, a phenomenon where the beam deflects out of the plane parallel to the applied force. This occurs when one cross-sectional dimension of the beam is much stiffer than the other, and minute material imperfections divert a portion of the applied force sideways, and the more pliant dimension gives way.

They simulated a 100-inch-long polypropylene beam cantilevered out from a fixed anchor point, with a narrow rectangular cross section. They applied a relatively strong transverse force against the stiffer (wider) cross-sectional dimension of the beam. To simulate minute redirected forces and thereby initiate out-of-plane buckling, they also applied a weaker perpendicular force against the thinner dimension. Full details of the original experiment can be found in [Chase 2006, Chase et al. 2011].

We performed similar experiments to verify our optimized chain algorithm and 3D 3R PRBM recipe.

5.1 Re-implementing [Chase et al. 2011] as a baseline

Chase published Visual Basic code for his algorithm in his thesis [Chase 2006]. We re-implemented his algorithm in C++ to give fair runtime comparisons with our extensions, which we had also written in C++. The re-implementation gave the same deflection results as [Chase et al. 2011, Chase 2006], so comparing the results of our extensions and optimizations to the results of our re-implementation gives a fair comparison of our methods to those of Chase et al. Hence, the run times and error values we give in this chapter for Chase et al.'s algorithm (the "unoptimized 1R algorithm") are from our C++ version.

5.2 Our experiment setup

To verify that our extensions and optimizations improved runtimes and maintained accuracy, we ran new experiments based on that of [Chase et al. 2011, Chase 2006]. We measured the runtimes and accuracy of two different chain algorithms, each in combination with two different element types, giving four cases for our experiments:

1. The unaltered algorithm of [Chase et al. 2011], with unoptimized $\mathcal{O}(n^2)$ chain calculations and 3D 1R PRBM elements.
2. Our optimized $\mathcal{O}(n)$ chain algorithm with Chase et al.'s original 3D 1R PRBM elements.
3. The unoptimized $\mathcal{O}(n^2)$ algorithm with our 3D 3R PRBM elements.
4. The optimized $\mathcal{O}(n)$ algorithm with our 3D 3R PRBM elements.

We define the error value of a chain algorithm using a metric that compares the deflection the algorithm yields to the deflection resulting from Chase et al.'s original algorithm (case 1) with 200 elements. This error metric is defined fully in section 5.2.1.

Chase et al. [Chase et al. 2011, Chase 2006] only supply results for case 1 at 200 elements. However, our own experiments also require knowing the error of Chase's original algorithm at other element counts. So to generate these error values, we ran case 1 at each of the desired counts, and compared each resulting deflection to the deflection at 200 elements.

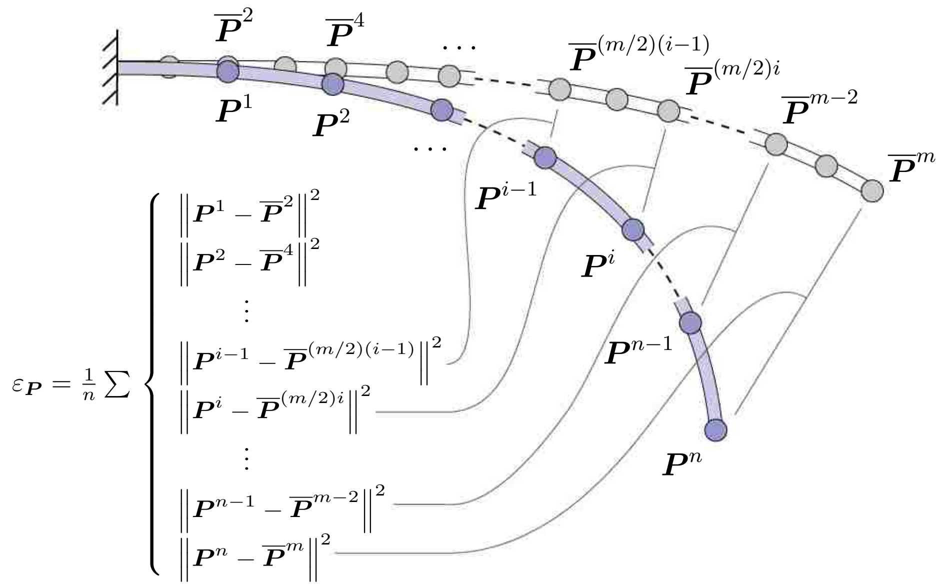
To test our hypothesis that the 3R algorithm would be accurate without the constraint from the 1R algorithm of near-zero element lengths, we needed the error values just mentioned for case 1, as well as error values for the 3R algorithms (cases 3 and 4) at each of the same element counts. The test then consisted in comparing the 3R error to the 1R error at each count. The results of this and other tests are given in section 5.3

5.2.1 Definition of error metric

To define the error of a given chain setup from one of the four cases in section 5.2, we first define a metric $\varepsilon_{\boldsymbol{P}}$ for determining a "difference" between two chains. $\varepsilon_{\boldsymbol{P}}$ is the mean squared error of corresponding element tips on the two chains, using the Euclidean distance formula:

$$\varepsilon_{\boldsymbol{P}} = \frac{1}{n} \sum_{i=1}^{n} \left\| \boldsymbol{P}^{i} - \overline{\boldsymbol{P}}^{(m/n)i} \right\|^{2} \tag{5.1}$$

$\boldsymbol{P}^{i}$ is a tip point in the *test chain*, and $\overline{\boldsymbol{P}}^{(m/n)i}$ is its corresponding point in the *expected chain*, where n is the number of elements in the test chain, and m is the number of elements in the expected chain. In order to ensure that $\boldsymbol{P}^{i}$ and $\overline{\boldsymbol{P}}^{(m/n)i}$ represent points at exactly the same length along their respective chains, n must evenly divide m. The arithmetic for choosing corresponding points between the two chains is illustrated in fig. 5.1.



$\overline{\boldsymbol{P}}^2$
$\overline{\boldsymbol{P}}^4$
$\overline{\boldsymbol{P}}^{(m/2)(i-1)}$
$\overline{\boldsymbol{P}}^{(m/2)i}$
$\overline{\boldsymbol{P}}^{m-2}$
$\overline{\boldsymbol{P}}^m$
$\boldsymbol{P}^1$
$\boldsymbol{P}^2$
$\boldsymbol{P}^{i-1}$
$\boldsymbol{P}^i$
$\boldsymbol{P}^{n-1}$
$\boldsymbol{P}^n$
$\varepsilon_{\boldsymbol{P}} = \frac{1}{n}\sum \left\{ \left\|\boldsymbol{P}^1 - \overline{\boldsymbol{P}}^2\right\|^2, \left\|\boldsymbol{P}^2 - \overline{\boldsymbol{P}}^4\right\|^2, \vdots, \left\|\boldsymbol{P}^{i-1} - \overline{\boldsymbol{P}}^{(m/2)(i-1)}\right\|^2, \left\|\boldsymbol{P}^i - \overline{\boldsymbol{P}}^{(m/2)i}\right\|^2, \vdots, \left\|\boldsymbol{P}^{n-1} - \overline{\boldsymbol{P}}^{m-2}\right\|^2, \left\|\boldsymbol{P}^n - \overline{\boldsymbol{P}}^m\right\|^2 \right.$

Figure 5.1: *Illustration of how elements from an example test chain (shaded in lavender) are paired with elements from the expected chain (white) for each term of the error metric $\varepsilon_{\boldsymbol{P}}$. The expected chain has m elements, and the test chain has $n = m/2$ elements.*

Now, to determine the accuracy of one of the four cases from section 5.2 with a given number of elements, we first set aside a copy of the undeflected chain. We deflect the first copy as a test chain using the appropriate type of PRBM element and either the optimized or unoptimized chain algorithm, as prescribed by the case being tested, with the given number of elements. We then deflect the second chain copy as a baseline or expected chain, with 200 1R elements using Chase et al.'s unoptimized algorithm (Case 1), which is known to give correct results when 200 elements are used. The accuracy of the first copy, or test chain, is determined by computing $\varepsilon_{\boldsymbol{P}}$ between the two deflected chain copies using eq. (5.1).

Though Chase et al. used only the tip point of the chain to verify their results, we included interior chain points in our error metric in order to highlight some subtle improvements seen from the use of our 3D 3R PRBM. During implementation and testing of the chain algorithm with 3D 3R elements, we noticed some circumstances in which the 3D 3R element gave rise to chain poses that visually appeared closer to the correct pose all along the length of the chain, where the 3D

1R element also gave the correct chain tip but looked less correct along the interior elements. We therefore chose an error metric that would include interior element configurations, to capture such differences. We elaborate on the observed differences in section 5.3.3.

5.3 Experiment results

5.3.1 Run times

We ran each of the four chain algorithm cases from section 5.2 with element counts of 1, 2, 4, 5, 8, 10, 20, 25, 40, 50, 100, and 200, which constitute all the whole divisors of the element count 200 from Chase et al.'s original experiment [Chase et al. 2011]. The tests were each run on a single execution thread, on a laptop computer with an Intel Core i7-3630Q processor at 2.4GHz.

Table 5.1 gives run times for each algorithm at each element count. These are plotted in fig. 5.2. Both axes of the graph are logarithmic to make the graph readable despite greatly increasing differences between consecutive values across the domain and range.

These figures demonstrate that, as expected, the $\mathcal{O}(n)$ 1R chain algorithm is considerably faster than the original $\mathcal{O}(n^2)$ 1R algorithm, with a x25 speedup in the case with 200 elements. It offers a x18 speedup at 200 elements with 3R PRBM elements.

The 3R algorithms are about 1.6 times slower on average than their 1R counterparts, which is not surprising given the increased complexity of their **updateElement** method.

The steep incoming slope in the lower-left of the graph for the unoptimized and optimized 1R algorithms results from drastically smaller runtimes at 1 element.

5.3.2 Error of the $\mathcal{O}(n)$ optimization

Table 5.2 gives the error value $\varepsilon_{\boldsymbol{P}}$ for each of the same algorithms, PRBM types, and element counts as in section 5.3.1 and table 5.1. These are plotted in fig. 5.3, again with logarithmic axes.

Table 5.1: *Runtimes, in seconds, for all four cases of the chain algorithm (see section 5.2) at various element counts.*

	Algorithm runtimes (s)			
Elements	1R Unopt.	1R Opt.	3R Unopt.	3R Opt.
1	0.002	0.003	0.366	0.409
2	0.405	0.349	0.665	0.569
4	0.971	0.628	1.326	0.976
5	1.190	0.686	1.799	1.249
8	1.891	0.970	3.117	1.806
10	2.889	1.147	4.153	2.132
20	7.121	1.842	9.866	3.495
25	9.645	2.122	13.118	4.081
40	18.519	2.867	24.470	5.453
50	25.778	3.298	33.240	6.243
100	73.499	5.378	95.534	10.101
200	255.877	9.583	330.970	18.670

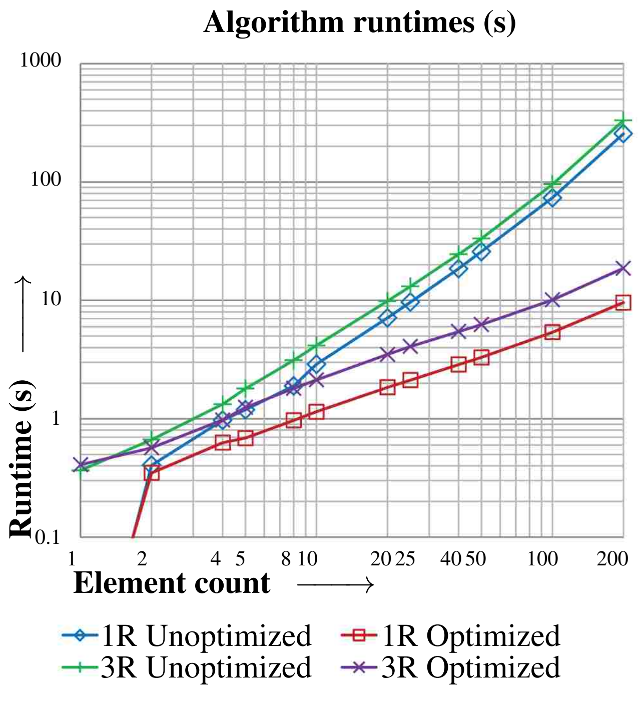


Figure 5.2: *A graph of the runtimes from table 5.1. The element count is plotted on the horizontal axis, with its corresponding runtime on the vertical axis. Both axes are logarithmic.*

Table 5.2: *Error ε_P for all four cases of the chain algorithm (see section 5.2) at various element counts. Each error value is for comparison against the original unoptimized 1R algorithm (case 1) at 200 elements.*

	Error ε_P of element tips (in.^{2})			
Elements	1R Unopt.	1R Opt.	3R Unopt.	3R Opt.
1	$1.31e{+}4$	$1.31e{+}4$	$1.99e{+}4$	$5.93e{+}3$
2	$1.79e{+}3$	$4.58e{+}3$	$2.86e{+}3$	$9.40e{+}3$
4	$3.62e{+}3$	$5.60e{+}3$	$3.45e{+}3$	$2.37e{+}3$
5	$3.30e{+}3$	$5.42e{+}3$	$2.59e{+}3$	$3.06e{+}3$
8	$3.01e{+}3$	$2.39e{+}3$	$1.43e{+}3$	$8.28e{+}2$
10	$1.46e{+}3$	$1.47e{+}3$	$2.21e{+}3$	$1.10e{+}3$
20	$8.76e{+}2$	$8.76e{+}2$	$5.45e{+}2$	$5.45e{+}2$
25	$1.49e{+}2$	$1.49e{+}2$	$2.79e{+}1$	$2.79e{+}1$
40	$7.83e{-}1$	$7.83e{-}1$	$4.05e{-}1$	$4.05e{-}1$
50	$4.58e{-}1$	$4.58e{-}1$	$2.18e{-}1$	$2.18e{-}1$
100	$5.47e{-}2$	$5.47e{-}2$	$1.88e{-}2$	$1.88e{-}2$
200	$0.00e{+}0$	$1.86e{-}21$	$9.28e{-}3$	$9.28e{-}3$

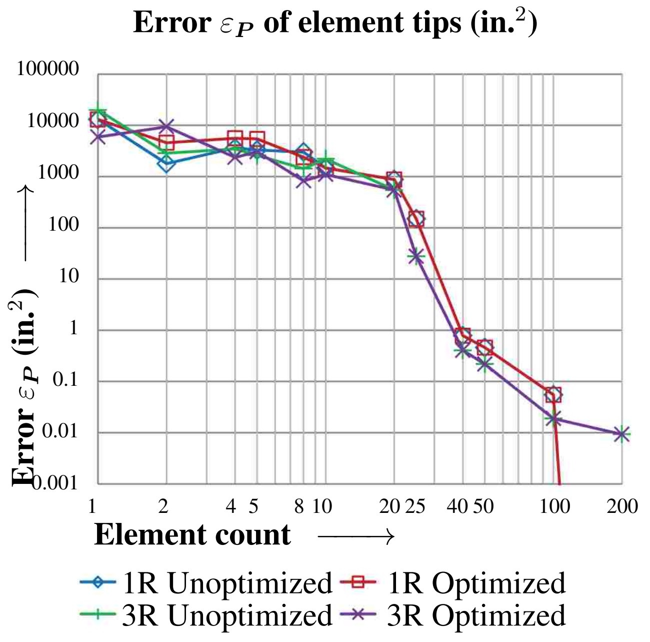


Figure 5.3: *A graph of the error values ε_P from table 5.2. The element count is plotted on the horizontal axis, with the resulting error ε_P on the vertical axis. Both axes are logarithmic.*

Observe that there is very little difference in the numbers and graphs for the error values between the optimized 1R algorithm and the unoptimized version, suggesting that they behave much the same per element count. This supports our claim that the optimized $\mathcal{O}(n)$ version of the chain calculations is correct, especially since its error is vanishingly small at 200 elements. Similar observations apply for the optimized and unoptimized versions of the 3R chain algorithms.

The steep outgoing slope near the right edge of the graph in fig. 5.3 is the symptom of an asymptote, which is expected in a logarithmic plot with zero and near-zero values.

5.3.3 Error of 3R algorithms versus 1R algorithms

Table 5.2 and the graph in fig. 5.3 show that for the element counts tested within the range 20 to 100, both the optimized and unoptimized versions of the 3R algorithm show lower error measures than both of the 1R algorithms.

It is interesting, however, that the accuracy of both the 3R and 1R versions begins to fail significantly near the same point in the spectrum of element counts, as can be seen by the jump in $\varepsilon_{\boldsymbol{P}}$ by two orders of magnitude between 25 and 40 elements for all four algorithms. The effects of this jump in the overall pose error can be seen clearly in fig. 5.4.

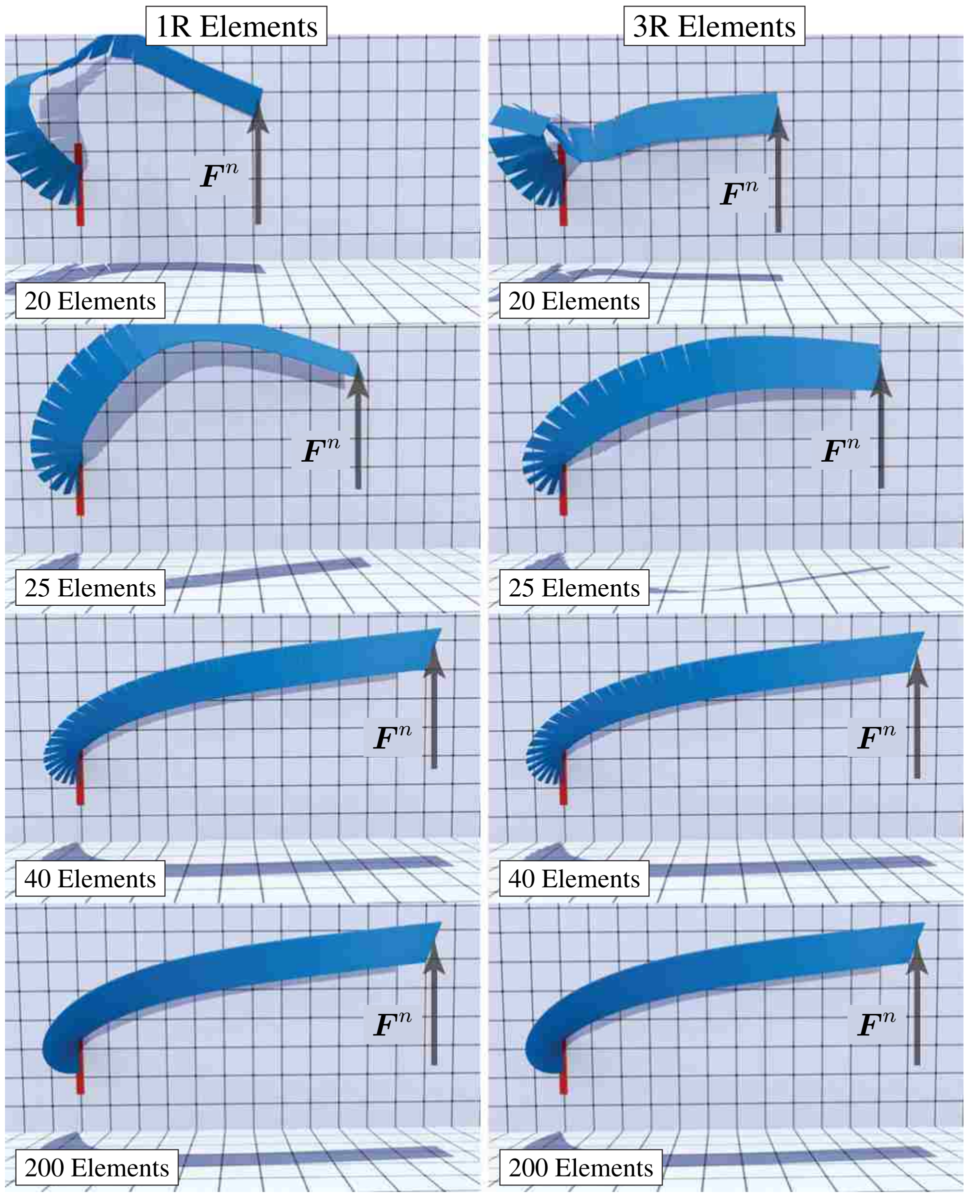


Figure 5.4: *Renders of deflected chains, each run with an optimized version of the chain algorithm. The left column shows chains of 1R PRBM elements at counts of 20, 25, 40, and 200, running top to bottom. The right column depicts 3R PRBM chains with each of the same element counts. The same force, depicted by an arrow, is applied to the tip of each chain. This force points upward, perpendicular to the floor of the scene, and parallel to the back wall.*

The left column of fig. 5.4 shows the chain pose that results from application of the optimized 1R chain algorithm with 20, 25, 40, and 200 elements in each panel going from top to bottom, and the right column shows the same but with the optimized 3R algorithm. The image in the bottom left panel of the figure shows the expected, or correct, pose of the chain, since the error for the 1R optimized run with 200 elements approaches zero. The same force is applied to the tip of each chain. This force points upward, perpendicular to the floor of the scene, and parallel to the back wall.

Compared visually to the expected pose, the poses for 20 and 25 elements for both 1R and 3R algorithms (first and second rows of fig. 5.4, both columns) are clearly wrong, though the 3R chain with 25 elements (second row, right column) is more correct. Looking at the third row of fig. 5.4, it is clear that we can only reasonably begin to call the chain poses “correct” starting at 40 elements. Thus, these renders confirm visually what the jump in the graph of fig. 5.3 indicated.

Though the chain looks better at 25 3R elements than with 25 1R elements, these results still weaken support for our hypothesis that a chain of 3R PRBM elements would be fully accurate for lower element counts than the 1R chain, since both the 1R and 3R algorithms begin to miss the correct chain tip location at around the same element count.

It is worth making one final observation about the error of the 3R algorithms. Table 5.2 and fig. 5.3 show a greater error for the 3R algorithms at 200 elements than the 1R algorithms. This is because the 1R algorithm is the error baseline, so its error is zero by definition. It is possible that the 3R algorithm at 200 elements is nonetheless more physically accurate than our baseline; to be sure, we would need to compare its deflection directly with the Finite Element simulation from [Chase et al. 2011].

Chapter 6

Conclusion and Future work

The runtime and accuracy results presented in chapter 5 demonstrate that PRBM-based algorithms offer potential as a new, viable method for simulating elastic materials in computer graphics. They offer the simplicity of mass-spring simulations, but also support torsion, and offer a clear parameterization from measurable physical quantities. Their extensions to 3D naturally support reduced coordinate formulations, which integrate well into traditional animation workflows with great advantages [Hadap 2006] (for more detail, see section 6.2.1).

The optimized $\mathcal{O}(n)$-complexity chain algorithm we have presented will give immediate speedup benefits to engineering work flows based on the $\mathcal{O}(n^2)$ version. Its application for rapid static simulation may also be of interest in film and animation production as an alternative solution for simulation pre-roll, or as a static modeling tool for physically plausible poses, such as tree branches swept sideways under a prevailing wind, or deducing initial "gravity-free" poses from models captured under the effects of gravity.

There remain improvements to be made to these methods. We suggest some extensions to the static methods in section 6.1. To fully realize their potential for animation and gaming applications, our methods will need to be extended to fully support dynamic simulations with constraint enforcement, collisions, and friction. Section 6.2 therefore explores promising leads for dynamic PRBM-based methods with these features. Finally, in section 6.3, we explore a number of ways the PRBM could efficiently support dynamic, online topology and material changes such as automatic fracture detection or time-variant elasticity.

6.1 Extending the static algorithm

This section presents some initial leads for extending the static methods in this paper. Section 6.1.1 proposes an experiment to investigate whether it is worth further pursuing extensions based on the 3R PRBM as a way of reducing necessary element counts, and section 6.1.2 identifies resources for introducing constraint enforcement to the static chain algorithm.

6.1.1 Alternative 3D 3R configurations

Our **updateElement** method and spring configurations for the 3D 3R PRBM was inspired as a direct extension of Chase et al.'s technique from the 3D 1R PRBM [Chase et al. 2011]. Though we verified in chapter 5 that our method gives correct results in the chain algorithm, it does not reduce the number of elements required for the fully correct result, as compared to the 1R element. Yet the hypothesis still seems reasonable that a PRBM recipe for combined loads would accurately support chains of fewer elements.

A near-term follow-on study could experiment with Su's and Chen et al.'s unmodified planar 3R element [Su 2009, Chen et al. 2011] in a strictly planar chain algorithm. If this 3R planar chain does support accuracy with lower element counts than a planar 1R chain, there would be reason to suppose the fault in our 3D case lay with our particular way of extending the 3R recipe to 3D. It would then be worth exploring alternative configurations for a 3D 3R recipe and **update-Element** method. Mechanics- or robotics-based methods for rigid serial linkages, such as those in [Tsai 1999], would be a worthwhile place to start.

6.1.2 Static constraints

We did not address constraints or collision handling for the static case in this work, but we wish to note that some promising leads for constraint enforcement can be gleaned from chapters 2 and

6.2 part (A) of Chase's thesis [Chase 2006], where he mentions techniques for the application of boundary conditions beyond the applied forces and moments we address in this work.

6.2 Proposed dynamic PRBM methods

Here we discuss what we regard as the next major step for PRBM-based methods: the development of general, dynamic simulations with PRBM elements. We first identify, in sections 6.2.1 and 6.2.2, some important attributes and characteristics that dynamic PRBM methods should support, in terms of how well they integrate with existing animation pipelines, as well as practical issues of stability and design complexity for the numerical integration methods they will require. Sections 6.2.3 and 6.2.4 then recommend some directions and general approaches for satisfying these criteria, based on preliminary research.

6.2.1 Reduced coordinate approaches

Reduced coordinates (also known as *generalized coordinates*) are so called because each joint deflection is given as a single scalar in terms of the local joint axis rather than as a vector in world or global coordinates. Chase et al.'s extension of the PRBM to 3D is essentially a reduced coordinate formulation, since the spring stiffness and deflection parameters are represented in local joint space. Thus, we regard a reduced coordinate formulation to be a natural choice for extending the chain solution to the dynamic case.

Using a reduced coordinate method should also hold significant benefits for incorporation with existing animation and simulation pipelines. For example, Hadap points out that for his work [Hadap 2006], a reduced coordinate formulation exactly matched the rig-space parameterizations for character articulation, which made the simulation method fit naturally into an existing animation pipeline, and which also allowed him to trivially extend the methods to support a number of tools for artistic direction of the simulation results. Hahn et al. dedicated an entire paper [Hahn et al. 2012] to the benefits gained from a simulation method that can output its results directly to rig-space, so it can be edited and manipulated directly by animators within their familiar work flow. Since our method is a reduced coordinate method that works directly in the space of joint articulations, it should benefit similarly.

6.2.2 Numerical integration in two key cases

Issues of numerical integration will need to be addressed for the dynamic case. The stiffness values generated by the PRBM recipes naturally lead to stiff differential equations, so stability will be a prime consideration in choosing an integration scheme.

We find it useful to view the problem of integrating a chain of elements in terms of two key cases:

1. Stable and correct integration to update individual element deformations
2. Stable and correct integration of inertial forces from element to element across the chain (take, for example, the "acceleration propagation" steps of Featherstone from [Mirtich 1996])

If the two cases can be cleanly decoupled, the dynamic algorithm will be much simpler. With the integration of one element being independent of other elements (the inter-relationships being correctly handled by case 2), the complexity and dimensionality of its internal differential equations will be constant regardless of the number of elements, being only a function of the local, fixed-size spring system. These will be much easier to solve than the more general non-linear equations that are often necessary with general mass-spring methods. (for a classical example in graphics of solving such systems, see [Baraff and Witkin 1998]).

6.2.3 Approaches to element integration (case 1)

The place to start for dynamic PRBM methods would be [Yu et al. 2005], which gives new stiffness parameters for correct energy conservation when the planar 1R recipe is used in a dynamic context.

This dynamic PRBM will need to be extended to 3D. Similarly, the 3R PRBM of [Su 2009, Chen et al. 2011] would either need to be validated as correct for dynamic applications or adjusted accordingly, and extended to 3D.

An early dynamic prototype of ours incorporates our own 3D version of Yu et al.'s dynamic 1R PRBM, with an original integration scheme, into the unaltered static chain algorithm—essentially ignoring case 2 while dropping in a solution for case 1. While this leads to incorrect overall results—it seems the accumulated quantities of the chain algorithm do not correctly propagate inertial forces for the dynamic case, leading to wildly exaggerated whipping motions—our internal element integration method did show promise. Judging subjectively from the animated results over long simulation times, the internal element integration is very stable, since individual elements retain their correct lengths and a reasonable appearance, and the chain retains its topology and overall integrity even while passing through extreme velocities and in and out of tightly curled configurations. (Though an alternative explanation of this seeming stability may be that the reduced coordinate formulation is simply doing a very good job maintaining element lengths).

Our dynamic element integration method was inspired by the derivation of an implicit integration method in [Baraff and Witkin 1998], though ours is simplified by solving a system of fixed size for each PRBM element locally, where [Baraff and Witkin 1998] must be equipped to solve a more generalized, if sparse, spring system. We first re-express the spring update equations as a Backwards Euler formulation. We then solve the first-order Taylor expansion of this formulation using analytic derivatives of the rotation matrices.

6.2.4 Approaches to chain integration (case 2)

We have already begun designing a reduced-coordinate method for a dynamic PRBM chain which is based on Featherstone's classic method for articulated rigid bodies. In exploring dynamic designs, we have found a tutorial on Featherstone's method by Mirtich to be of great value. The tutorial is found in Mirtich's oft-cited Ph.D. thesis [Mirtich 1996], and gives a clear derivation from physics and mechanics, and even adds an extension for treating free bodies (we have limited our discussion to anchored bodies in this paper). [Kokkevis 2004] will also be useful, for expanding Featherstone's method to handle constraint enforcement.

We are unsure at this point whether it is physically correct to disregard internal element deformations when computing Featherstone's inertial force propagations across the chain, or how they must be adjusted if cases 1 and 2 are inherently coupled. [Barbič and Zhao 2011, Bertails 2009] both implement algorithms similar to Featherstone's, so they merit closer study for insights about how to develop Featherstone with non-rigid elements.

With some additional research since beginning the dynamic design, however, we have reason to fear that Featherstone will be unstable for long chains of elements. Hadap lists several reasons why networks of rods can give rise to "stiff and highly non-linear differential equations … [which] call for an implicit integration[6] scheme" [Hadap 2006]. We surmise that these differential equations would be especially difficult to stabilize with a recursive method over long chains of elements, since instability by its very nature is compounded by propagation and repetition. The problem would only be exacerbated by the stiff springs and potentially large local deformations of each PRBM element.

Unfortunately, Hadap, citing his earlier work [Hadap 2003], claims that implicit integration is difficult to implement with prior reduced coordinate methods such as Featherstone's. Though [Barbič and Zhao 2011, Bertails 2009] show stable results, their examples are limited to fairly shallow graphs. By contrast, PRBM recipes give lower-order elements, requiring serial sub-chains of numerous elements in order to capture detailed shapes. Thus, if it is true that the implied instability of Featherstone is related to connectivity depth, an alternative will be necessary to keep PRBM chains stable.

Hadap presents an alternative reduced coordinate method which is based on *Differential Algebraic Equations*, combining the benefits of reduced coordinates with stable integration

[6]Stiff, non-linear differential equations can be difficult to solve stably, that is, without irretrievably straying from the correct solution. *Implicit integration*, so called because it typically employs an iterative or indirect means of arriving at a solution, rather than an *explicit* closed formula, is a classic method in graphics for stable simulation of stiff systems. See [Baraff and Witkin 1998] for an important introduction of implicit integration to the graphics field.

[Hadap 2006]. The method has linear runtime complexity in the number of elements, and supports collisions, constraint enforcement, and static and dynamic friction. Hadap's method may therefore be more appropriate than Featherstone-like methods such as [Barbič and Zhao 2011, Bertails 2009] for addressing case 2 when PRBM elements are involved. With luck, completing a viable dynamic PRBM chain will be as simple as substituting our dynamic PRBM prototype from section 6.2.3 into Hadap's framework.

6.3 Directable, dynamic topology and material change

Here we discuss a number of ways the rapid, efficient parameterizations of PRBM recipes could be used to re-parameterize dynamic PRBM chains and elements during actual simulation to achieve dynamic effects not possible with pre-computed reduction methods.

6.3.1 Automatic stress and fracture detection

Many of the published PRBM recipes include simple formulas for computing the stresses on a PRBM element under a given load, as well as for computing the maximum stress an element may support. For highly complex simulations in particular, these formulas could be the basis of an efficient system for automatically detecting fracture conditions.

Consider a hypothetical film scene with numerous trees in a forest under heavy winds, with dynamically snapping limbs, twigs, and leaves blowing free. It would be prohibitively time-consuming for animators to manually script and direct each instance of fracture.

PRBM-based fracture detection could solve this problem simply and efficiently. Augmenting a dynamic PRBM method with stress and fracture detection would require a trivial extension to the **updateElement** method. When a complete fracture is detected and initiated, separating the branch as a free, independent PRBM chain would be a simple matter of severing the child relationship from the tree and designating a new root element for the branch.

By contrast, FEM reduction methods would be less simple, both for computing stresses and locations of fracture, and for executing the actual separation as distinct reduced bodies. For example, Barbič and Zhao's substructuring [Barbič and Zhao 2011] would either limit fracture points to the rigid sub-domain interfaces, or would require re-computing additional sub-domains to replace the existing domain containing the true site of fracture. Dynamic correctness would require that the new sub-domains be re-reduced from the original full resolution volume mesh; simply partitioning and splitting the existing reduced domain would not suffice.

Finite Elements simulations can nonetheless produce detailed, higher-dimensional effects of which PRBMs are incapable, such as cracking, splitting, and splintering, as well as loosening and ejecting minor debris. Where such detail is desirable, the PRBM method could still be used to detect and locate fracturing elements very efficiently. A suitable FEM volume mesh could then be generated on-demand, and the FEM simulation could be limited and localized to the fracture region. If the FEM simulation determines that only partial fracture should result, and the branch will hold on by a smaller thread of wood or bark, a new PRBM element could be substituted and re-parameterized by the geometry and elasticity of the remaining connection, and the system could return to the simpler PRBM simulation. If stress sufficient for full fracture should later arise, the new PRBM element could detect it, and the FEM process could be repeated.

6.3.2 Efficient, online material change

PRBM recipes and their prescribed parameterizations are so simple that dynamic simulations could be augmented to support any number of online topology changes or material changes very efficiently, simply by updating PRBM input parameters during simulation. For example, *creep* and *fatigue*, or the tendency of elastic materials to lose their elasticity fully or by degrees under a sustained or heavy load, could be modeled by measuring the time duration of a load or by detecting high stresses at a given moment, and adjusting the PRBM's input elasticity accordingly.

Since the PRBM input parameters are simple functions of intuitive physical quantities, time-dependent changes could be readily controlled by animators, simply by adjusting key-frames and animation curves for the intuitive parameters. Imagine, for example, a time-lapse of a growing tree, with changing graph topology as new branches emerge and are added, changing elasticity as green shoots are replaced by wood, and automatically updated area moments of inertia as the trunk and branches thicken. Times and locations for the addition of new branches, the changing elasticity values, and the rate of diameter increases could all be directly controlled by animators. All the resulting PRBM parameter changes could take place frame by frame with little additional cost.

In summary, future dynamic PRBM-based methods should readily support online topology and material changes, whether they are pre-scripted by animators, or they arise at simulation time as a result of programmatic detection methods.

Appendix A

Alternative $\mathcal{O}(n)$ formulation

Overall, the alternative optimization of the chain algorithm is identical to the optimization presented in section 4.1, except for how we compute eq. (4.4a) for $\boldsymbol{a}_{\Sigma}^{i}$ from section 4.1.2. $\boldsymbol{a}_{\Sigma}^{i}$ is still computed in constant time for a given step i of the chain algorithm, using the transformation matrix R_{Σ}^{i} resulting from the previous steps of the algorithm. For consistency with matrix indexing, we will switch from indexing vector coordinates with the alphabetic symbols x, y, z, w to the corresponding numeric indices $1, 2, 3, 4$.

To begin, we define the matrix S^i as a sum of vector outer products,

$$\mathrm{S}^i = \sum_{j=i}^{n} \boldsymbol{P}^j \left(\boldsymbol{F}^j\right)^T \tag{A.1}$$

Consequently, for a given entry $s_{\alpha\beta}$ of S^i,

$$s_{\alpha\beta} = \sum_{j=i}^{n} (\boldsymbol{P}^j)_{\alpha} (\boldsymbol{F}^j)_{\beta} \tag{A.2}$$

S^i may be pre-computed for all i as a vector suffix scan over matrix addition. Then, during a given step i of the chain algorithm, a useful matrix Q^i can be computed in constant time:

$$\mathrm{Q}^i = \mathrm{R}_{\Sigma}^{i} \mathrm{S}^i \tag{A.3}$$

Its usefulness will be evident in a moment.

Referring back to eq. (4.4a) and applying the cross product in its right-hand side, we obtain

$$\begin{aligned}
\boldsymbol{a}_{\Sigma}^{i} &= \sum_{j=i}^{n} \left(\mathrm{R}_{\Sigma}^{i}\boldsymbol{P}^{j}\right) \times \boldsymbol{F}^{j} \\
&= \sum_{j=i}^{n} \begin{bmatrix} (\mathrm{R}_{\Sigma}^{i}\boldsymbol{P}^{j})_2(\boldsymbol{F}^{j})_3 - (\mathrm{R}_{\Sigma}^{i}\boldsymbol{P}^{j})_3(\boldsymbol{F}^{j})_2 \\ (\mathrm{R}_{\Sigma}^{i}\boldsymbol{P}^{j})_3(\boldsymbol{F}^{j})_1 - (\mathrm{R}_{\Sigma}^{i}\boldsymbol{P}^{j})_1(\boldsymbol{F}^{j})_3 \\ (\mathrm{R}_{\Sigma}^{i}\boldsymbol{P}^{j})_1(\boldsymbol{F}^{j})_2 - (\mathrm{R}_{\Sigma}^{i}\boldsymbol{P}^{j})_2(\boldsymbol{F}^{j})_1 \\ 0 \end{bmatrix}
\end{aligned}$$

Here, we recall from matrix multiplication that with $r_{\alpha\beta}$ denoting an entry of R_{Σ}^{i}, we have $(\mathrm{R}_{\Sigma}^{i}\boldsymbol{P}^{j})_{\alpha} = \sum_{\boldsymbol{u}=1}^{4} r_{\alpha\boldsymbol{u}}(\boldsymbol{P}^{j})_{\boldsymbol{u}}$. Hence, continuing the derivation, $\boldsymbol{a}_{\Sigma}^{i}$ becomes

$$\sum_{j=i}^{n} \begin{bmatrix} \left(\sum_{\boldsymbol{u}=1}^{4} r_{2\boldsymbol{u}}(\boldsymbol{P}^{j})_{\boldsymbol{u}}\right)(\boldsymbol{F}^{j})_3 - \left(\sum_{\boldsymbol{u}=1}^{4} r_{3\boldsymbol{u}}(\boldsymbol{P}^{j})_{\boldsymbol{u}}\right)(\boldsymbol{F}^{j})_2 \\ \left(\sum_{\boldsymbol{u}=1}^{4} r_{3\boldsymbol{u}}(\boldsymbol{P}^{j})_{\boldsymbol{u}}\right)(\boldsymbol{F}^{j})_1 - \left(\sum_{\boldsymbol{u}=1}^{4} r_{1\boldsymbol{u}}(\boldsymbol{P}^{j})_{\boldsymbol{u}}\right)(\boldsymbol{F}^{j})_3 \\ \left(\sum_{\boldsymbol{u}=1}^{4} r_{1\boldsymbol{u}}(\boldsymbol{P}^{j})_{\boldsymbol{u}}\right)(\boldsymbol{F}^{j})_2 - \left(\sum_{\boldsymbol{u}=1}^{4} r_{2\boldsymbol{u}}(\boldsymbol{P}^{j})_{\boldsymbol{u}}\right)(\boldsymbol{F}^{j})_1 \\ 0 \end{bmatrix}$$

Since for all α, $r_{\alpha\boldsymbol{u}}$ is constant over the index of summation j, we can push the outermost summation inside the inner summation contained in each entry of the vector, yielding

$$\boldsymbol{a}_\Sigma^i = \begin{bmatrix} \sum_{\boldsymbol{u}=1}^{4} r_{2\boldsymbol{u}} \sum_{j=i}^{n} (\boldsymbol{P}^j)_{\boldsymbol{u}}(\boldsymbol{F}^j)_3 - \sum_{\boldsymbol{u}=1}^{4} r_{3\boldsymbol{u}} \sum_{j=i}^{n} (\boldsymbol{P}^j)_{\boldsymbol{u}}(\boldsymbol{F}^j)_2 \\ \sum_{\boldsymbol{u}=1}^{4} r_{3\boldsymbol{u}} \sum_{j=i}^{n} (\boldsymbol{P}^j)_{\boldsymbol{u}}(\boldsymbol{F}^j)_1 - \sum_{\boldsymbol{u}=1}^{4} r_{1\boldsymbol{u}} \sum_{j=i}^{n} (\boldsymbol{P}^j)_{\boldsymbol{u}}(\boldsymbol{F}^j)_3 \\ \sum_{\boldsymbol{u}=1}^{4} r_{1\boldsymbol{u}} \sum_{j=i}^{n} (\boldsymbol{P}^j)_{\boldsymbol{u}}(\boldsymbol{F}^j)_2 - \sum_{\boldsymbol{u}=1}^{4} r_{2\boldsymbol{u}} \sum_{j=i}^{n} (\boldsymbol{P}^j)_{\boldsymbol{u}}(\boldsymbol{F}^j)_1 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} \sum_{\boldsymbol{u}=1}^{4} r_{2\boldsymbol{u}} s_{\boldsymbol{u}3} - \sum_{\boldsymbol{u}=1}^{4} r_{3\boldsymbol{u}} s_{\boldsymbol{u}2} \\ \sum_{\boldsymbol{u}=1}^{4} r_{3\boldsymbol{u}} s_{\boldsymbol{u}1} - \sum_{\boldsymbol{u}=1}^{4} r_{1\boldsymbol{u}} s_{\boldsymbol{u}3} \\ \sum_{\boldsymbol{u}=1}^{4} r_{1\boldsymbol{u}} s_{\boldsymbol{u}2} - \sum_{\boldsymbol{u}=1}^{4} r_{2\boldsymbol{u}} s_{\boldsymbol{u}1} \\ 0 \end{bmatrix}$$

The second equality follows directly from eq. (A.2). Finally, from eq. (A.3) and the definition of matrix multiplication, we see that $q_{\alpha\beta} = \sum_{\boldsymbol{u}=1}^{4} r_{\alpha\boldsymbol{u}} s_{\boldsymbol{u}\beta}$, so

$$\boldsymbol{a}_\Sigma^i = \begin{bmatrix} q_{23} - q_{32} \\ q_{31} - q_{13} \\ q_{12} - q_{21} \\ 0 \end{bmatrix} \tag{A.4}$$

In summary, S^i is pre-computed from eq. (A.1) for all i at the start of the chain algorithm, then during each step i of the algorithm, Q^i is computed from eq. (A.3) using the cumulative transformation R_Σ^i from previous steps. Finally eq. (A.4) computes $\boldsymbol{a}_\Sigma^i$ from the entries of Q^i.

We find this alternative formulation of the linear-time optimization interesting for a curious pattern which emerges: while the cross product appearing in eq. (4.4a) prevents a trivial factorization of the summation, the final form of the optimization in eq. (A.4) still bears a striking notational resemblance to the definition of the cross product. The only difference is the use of direct juxtaposition of indices to denote a matrix entry, rather than the juxtaposition of indexed symbols to denote multiplication of two scalars:

$$\boldsymbol{b} \times \boldsymbol{c} = \begin{bmatrix} (\boldsymbol{b})_2(\boldsymbol{c})_3 - (\boldsymbol{b})_3(\boldsymbol{c})_2 \\ (\boldsymbol{b})_3(\boldsymbol{c})_1 - (\boldsymbol{b})_1(\boldsymbol{c})_3 \\ (\boldsymbol{b})_1(\boldsymbol{c})_2 - (\boldsymbol{b})_2(\boldsymbol{c})_1 \\ 0 \end{bmatrix}$$

Appendix B

The cross product matrix

The traditional cross product operator '$\times$' for 3D vectors can be re-written in terms of matrix multiplication. Given two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ (with homogenous w-coordinates, per our implementation), the cross product $\boldsymbol{a} \times \boldsymbol{b}$ can be written one of two ways, depending on which vector is used to form the *cross product matrix* [Wikipedia 2013a]:

$$\boldsymbol{a} \times \boldsymbol{b} = [\boldsymbol{a}]_\times \boldsymbol{b} = [\boldsymbol{b}]_\times^T \boldsymbol{a}$$

where ^{T} denotes matrix transposition, and

$$[\boldsymbol{a}]_\times = \begin{bmatrix} 0 & -a_z & a_y & 0 \\ a_z & 0 & -a_x & 0 \\ -a_y & a_x & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

References

BARAFF, D., AND WITKIN, A. 1998. Large steps in cloth simulation. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, SIGGRAPH '98, 43–54.

BARBIČ, J., AND ZHAO, Y. 2011. Real-time large-deformation substructuring. *ACM Trans. on Graphics (SIGGRAPH 2011) 30*, 4, 91:1–91:7.

BARBIČ, J., SIN, F., AND GRINSPUN, E. 2012. Interactive editing of deformable simulations. *ACM Trans. Graph. 31*, 4 (July), 70:1–70:8.

BERGOU, M., WARDETZKY, M., ROBINSON, S., AUDOLY, B., AND GRINSPUN, E. 2008. Discrete elastic rods. *ACM Trans. Graph. 27*, 3 (Aug.), 63:1–63:12.

BERTAILS, F. 2009. Linear time super-helices. *Computer Graphics Forum 28*, 2, 417–426.

CASATI, R., AND BERTAILS-DESCOUBES, F. 2013. Super space clothoids. *ACM Trans. Graph. 32*, 4 (July), 48:1–48:12.

CHASE, R. P., TODD, R. H., HOWELL, L. L., AND MAGLEBY, S. P. 2011. A 3-d chain algorithm with pseudo-rigid-body model elements. *Mechanics Based Design of Structures and Machines 39*, 1, 142–156.

CHASE, R. P. 2006. *Large 3-D Deflection and Force Analysis of Lateral Torsional Buckled Beams*. Master's thesis, Brigham Young University.

CHEN, G., XIONG, B., AND HUANG, X. 2011. Finding the optimal characteristic parameters for 3r pseudo-rigid-body model using an improved particle swarm optimizer. *Precision Engineering*

35, 3, 505 – 511.

COROS, S., MARTIN, S., THOMASZEWSKI, B., SCHUMACHER, C., SUMNER, R., AND GROSS, M. 2012. Deformable objects alive! *ACM Trans. Graph. 31*, 4 (July), 69:1–69:9.

HADAP, S. 2003. *Hair simulation*. PhD thesis, MIRALab, CUI, University of Geneva.

HADAP, S. 2006. Oriented strands: Dynamics of stiff multi-body system. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, SCA '06, 91–100.

HAHN, F., MARTIN, S., THOMASZEWSKI, B., SUMNER, R., COROS, S., AND GROSS, M. 2012. Rig-space physics. *ACM Trans. Graph. 31*, 4 (July), 72:1–72:8.

HARRIS, M., SENGUPTA, S., AND OWENS, J. D. 2007. Parallel prefix sum (scan) with CUDA. In *GPU Gems 3*, H. Nguyen, Ed. Addison Wesley, August, ch. 39, 851–876.

HILDEBRANDT, K., SCHULZ, C., VON TYCOWICZ, C., AND POLTHIER, K. 2012. Interactive spacetime control of deformable objects. *ACM Trans. Graph. 31*, 4 (July), 71:1–71:8.

HOWELL, L. L. 2001. *Compliant Mechanisms*. John Wiley & Sons.

KHAREVYCH, L., MULLEN, P., OWHADI, H., AND DESBRUN, M. 2009. Numerical coarsening of inhomogeneous elastic materials. *ACM Trans. Graph. 28*, 3 (July), 51:1–51:8.

KOKKEVIS, E. 2004. Practical physics for articulated characters. In *Game Developers Conference*.

MARTIN, S., KAUFMANN, P., BOTSCH, M., GRINSPUN, E., AND GROSS, M. 2010. Unified simulation of elastic rods, shells, and solids. *ACM Trans. Graph. 29*, 4 (July), 39:1–39:10.

MIRTICH, B. V. 1996. *Impulse-based Dynamic Simulation of Rigid Body Systems*. PhD thesis, University of California at Berkeley.

MÜLLER, M., AND CHENTANEZ, N. 2011. Solid simulation with oriented particles. *ACM Trans. Graph. 30*, 4 (July), 92:1–92:10.

NESME, M., KRY, P. G., JEŘÁBKOVÁ, L., AND FAURE, F. 2009. Preserving topology and elasticity for embedded deformable models. *ACM Trans. Graph. 28*, 3 (July), 52:1–52:9.

PAULY, J., AND MIDHA, A. 2006. Pseudo-rigid-body model chain algorithm, part 1: Introduction and concept development. In *Volume 2: 30th Annual Mechanisms and Robotics Conference, Parts A and B*, vol. 2, ASME, 173–181.

SELLE, A., LENTINE, M., AND FEDKIW, R. 2008. A mass spring model for hair simulation. *ACM Trans. Graph. 27*, 3 (Aug.), 64:1–64:11.

SIEK, J., LEE, L.-Q., LUMSDAINE, A., SUTTON, A., AND WILLCOCK, J., 2011. The boost graph library. Retrieved Oct. 16, 2011 from `http://www.boost.org/doc/libs/1_47_0/libs/graph/doc/index.html`.

SU, H.-J. 2009. A pseudorigid-body 3r model for determining large deflection of cantilever beams subject to tip loads. *Journal of Mechanisms and Robotics 1*, 2.

SUEDA, S., JONES, G. L., LEVIN, D. I. W., AND PAI, D. K. 2011. Large-scale dynamic simulation of highly constrained strands. *ACM Trans. Graph. 30*, 4 (July), 39:1–39:10.

TSAI, L.-W. 1999. *Robot Analysis : The Mechanics of Serial and Parallel Manipulators*. John Wiley & Sons.

WIKIPEDIA, 2013. Cross product — wikipedia, the free encyclopedia. Retrieved July 26, 2013 from `http://en.wikipedia.org/w/index.php?title=Cross_product&oldid=563797268`.

WIKIPEDIA, 2013. List of moment of areas — wikipedia, the free encyclopedia. Retrieved Aug. 11, 2013 from `http://en.wikipedia.org/w/index.php?title=List_of_moment_of_areas&oldid=560450988`.

YOUNG, W. C., AND BUDNYAS, R. G. 2002. *Roark's Formulas for Stress and Strain*, 7 ed. McGraw-Hill.

Yu, Y.-Q., Howell, L. L., Yue, Y., He, M.-G., and Lusk, C. 2005. Dynamic modeling of compliant mechanisms based on the pseudo-rigid-body model. *Journal of Mechanical Design 127*, 4 (February), 760–765.