DOCTOR OF ENGINEERING (ENGD)

Constraint based simulation of soft and rigid bodies

Lewin, Christopher

Award date:
2016

Awarding institution:
University of Bath

Constraint based simulation of soft and rigid bodies

Chris Lewin

A thesis submitted for the degree of Doctor of Engineering

Abstract

This dissertation presents a number of related works in real-time physical animation, centered around the theme of constraint-based simulation. Methods used for real-time simulation of deformable bodies tend to differ quite substantially from those used in offline simulation; we discuss the reasons for this and propose a new position-based finite element method that attempts to produce more realistic simulations with these methods. We also consider and adapt other methods to make them more suitable for game physics simulation. Finally, we adapt some concepts from deformable body simulation to define a deformable *rod constraint* between rigid bodies that allows us to represent the kinematics of the human spine using fewer degrees of freedom than required with a strictly joint-based model.

Acknowledgements

I would like to express my gratitude to my supervisors- Dr. Philip Willis, Dr. Chris Williams and Dr. Tom Waterson. All three have at various times shown a level of belief in my abilities vastly outstripping my own, and for that trust I am most grateful. Phil's sage advice and humour in all situations have made the last four years much less stressful than they might have otherwise been. My conversations with Chris led to a far greater understanding of the fundamentals of my topic than I would have been able to attain on my own, and any academic rigor I have been able to incorporate into this work is almost certainly due to his influence. Tom believed in me enough to give me a job even after getting to know me, and our many conversations about physics both everyday and esoteric have been hugely useful and enjoyable. One day I may even learn geometric algebra and group theory so he doesn't have to talk down to my level.

Special thanks are due to others at Bath, EA and elsewhere who have made my life as a research engineer more pleasant than it had any right to be. Dr. Mike Bassett fought in ways I will probably never know to get and keep support for the EngD program at EA, and always went out of his way to make me feel part of the team. Katja Haferburg was a fantastic administrator and friend to everyone on the program. Dr. Ashton Mason, Thanasis Vogiannou and Matt Thorman allowed me at various times to bounce ideas off them, at great cost to their time. Tom Nugent was a great friend and long-suffering recipient of excited announcements about my research, as well as anything else I managed to achieve in the last four years.

Finally I would like to thank my family for keeping me both physically and mentally healthy despite setbacks of all kinds, and for regularly deflating my sense of self importance. Without their care, none of this would have been possible.

Contents

CONTENTS

Chapter 1

Introduction

This dissertation is about simulating physical phenomena in real-time for use in videogames. This field is quite distinct from the higher-end work in visual effects and academia, and the ways in which some phenomena, such as cloth, are simulated can seem quite strange at first glance. It turns out that there are good reasons why things are done the way they are, which one discovers as soon as one attempts to do things differently. Nevertheless, there is scope for exploring realistic physical simulation within the specific constraints dictated by the videogame use-case. The first step is to understand these constraints.

1.1 On Videogame Physics

The term 'real-time' is often bandied about when talking about algorithms used in games. This is an adequate shorthand for a more complex list of priorities, which we will get to in a moment. However, it is important to note here that games are not real-time systems in the strict sense of the word. What it actually means for a software system to be real-time is that we can *guarantee* that the system will respond before a given deadline. While game developers certainly favor algorithms with predictable performance, the consequences of a missed deadline are relatively mild; a missed frame in a game will annoy the user but not cause any catastrophic failure. So we should make it clear here that when we - and other practitioners of game physics simulation - refer to real-time methods we are in fact talking about a set of priorities, some of which have little to do with performance. In loose order of importance, they are:

1.1.1 Robustness

The most important quality for a simulation to have is that it must be robust to all possible user input. Unlike simulations performed for visual effects (VFX) shots in films, or for structural analysis in engineering, game physics simulations are expected to work under all possible situations created by player input, without any supervision. Ensuring this condition is met is approached from two ends: we must use a simulation technology that is fundamentally robust, and author our simulations in such a way that they are not easily broken. From the other end, we limit the player's ability to affect the simulation. As an example, simulated clothing on a game character may work perfectly well as long as the character only performs a specific set of movements. However, any new motions we introduce may be violent enough, or pathological enough in some other way, to cause our simulation to break. Similarly, if we give the player more agency - say, the ability to grab and pull the clothing of other characters - we will have to put much more effort into the core simulation to guarantee its robustness in all cases. This creates a tension between the fidelity of a simulation we can provide and our ability to control that simulation. In extreme cases this can lead to physics being hugely over-constrained in the name of guaranteeing robustness, but at the cost of removing most of the expressiveness of the simulation. It is the job of the game physicist to avoid this situation by providing simulations that are naturally robust to input of all kinds.

1.1.2 Reliable Performance

As might be expected, performance is also high on our list of priorities. However, what we mean exactly when talking about performance requirements in games is very different to what is meant in other areas. In the field of high-performance computing, and in non-real-time computer graphics, algorithms are often analysed primarily for their scaling properties; the best algorithms are considered to be those that deal well with ever-increasing problem size. The ability to scale to very large workloads executing in a massively parallel environment is highly valued in these environments. Conversely, in videogame simulation we have a fixed target architecture (the game console hardware *du jour*). Crucially, physics

is not even close to being the primary function of a game; we can expect most of the running time of the target hardware to be taken up with tasks relating to graphics, world management, networking, audio, artificial intelligence, and so on. Thus physics simulation and collision detection get only a small slice of the already-minuscule amount of processing time available in a 30Hz frame. The presence of other, higher-priority workloads on the target hardware is the most important distinguishing feature of game physics.

1.1.3 Appropriate Realism

'Realism' in computer graphics, and games in particular, has a different meaning than in more serious fields like engineering. In these fields, the *predictive power* of an algorithm is the most important factor. We need to be able to rely on the ability of simulations used in (for instance) crash testing to accurately predict the behaviour of real world materials. In contrast, for computer graphics we only need to generate the *impression* of reality in the user. What this means in practice is that the degree of realism required for a given phenomenon depends on how sensitive the audience is to a lack of correctness in that area. For instance, humans are very good at detecting a lack of realism in depictions of other humans (the 'uncanny valley' effect), and surprisingly good at noticing unrealistic trajectories of rigid bodies, but poor at noticing deficiencies in the simulation of rarely-encountered phenomena like collapsing buildings or crashing cars. The fidelity required for a given simulation is therefore strongly coupled to how common that phenomenon is. Another interesting observation is that *transient* phenomena do not need to be modelled as carefully as steady states. The animation of a collapsing building might last only a second but the ruin left behind will persist forever; we can allow the former to be quite unrealistic as long as the latter is convincing. Some phenomena do not need to be physically simulated at all to be appropriately convincing; fire and explosions are common cases found in games where a purely procedural approach is (currently) sufficient.

1.2 Constraint Based Methods

As briefly mentioned previously, humans are remarkably good at detecting unrealistic trajectories of rigid objects. As a result, the realism requirements for rigid body simulation in games are very high and this field was the first to become genuinely well developed. Indeed, some of the software developed for game physics simulation like Bullet [Coumans 2005] is considered the best in the world of its type. Early physics engines that used simple explicit time integration and a force- or acceleration-based contact model proved to be insufficiently robust and realistic for the fairly demanding game use case. It eventually became clear that hard inequality constraints had to be the mainstay of a realistic rigid body simulation, as opposed to the soft springs of force-based methods. Early work such as that of Stewart and Trinkle [1996] was too slow for large-scale simulations, requiring as it did the use of Lemke's algorithm. Eventually the field settled on a family of Projected Gauss Seidel (PGS) iterative processes (the most well known of which is the 'Sequential Impulses' method), which allow cheap and predictable simulation of inequality-constrained rigid bodies. While these methods converge slowly and have very obvious failure cases, it is easy to minimize the player's exposure to such situations by simply not allowing the game to create them. PGS is thus in some sense a model game physics simulation algorithm; it is predictably cheap, robust in the right circumstances, and its shortcomings can be avoided without eroding the realism of the method. It is not surprising, then, that practitioners of game physics have sought to adapt it to other fields. The most successful technique in this vein is usually called *Position Based Dynamics* (PBD).

1.2.1 Position Based Dynamics

PBD is the dominant tool for modelling deformable bodies in videogames. In particular, most game cloth simulations use this technique. It grew out of a method used by Jakobsen [2001] for modelling character ragdolls, but was quickly adapted by Muller et al. [2007] for deformable body simulation. While presented by both authors as an empirical method, PBD is actually easily recognisable as an adaptation of the PGS method for rigid body simulation. Using Verlet

integration (hence the 'position based' name) and solving nonlinear constraints with sequential quadratic programming, PBD inherits both the advantages and disadvantages of PGS rigid body simulations. One might think the ability to use hard inequality constraints would be fairly useless in a simulation of deformable bodies, but in fact this ability is extremely useful when trying to satisfy our primary demand of *robustness*.

As an example, take character clothing. High-end global clothing simulations like Autodesk's nCloth [Stam 2009] are quite unstable when the cloth is only held in place on the wearer by frictional contact with the skin. Particularly heinous actions, such as pinching cloth between two colliders, can result in the clothing being pushed through the character and coming loose. One way of alleviating this problem in a matrix-based simulation is to attach springs to each simulated point that attract that point towards a known-good position, such as the vertex's initial point on the character. While this will work, the effect can be quite obvious and distracting. Springs do not provide any sort of guarantee on the position of each point at the end of the frame, so we can only make them very stiff and hope for the best. This is thus a fairly bad way of controlling our simulation because it is both ineffective and highly restrictive. With the freedom to use inequality constraints, we can instead simply keep each point inside a sphere of a given radius from our known-good point. This provides a cast-iron guarantee that our simulation points will not stray too far, without introducing any ghost forces when they are inside the allowed zone. Thus we can control the simulation without impeding its expressiveness too much. The ability to incorporate inequalities can be, and has been, abused further to create other effects that are not possible in continuous models. For instance, we can model cloth that resists extension but not compression [Müller and Chentanez 2010], or cloth that only resists folding below a certain radius of curvature. Finally, the most obvious advantage is that contact constraints can be modelled exactly, without recourse to troublesome continuous approximations (i.e. the penalty method, in which hard contacts are replaced by soft spring forces). What this adds up to is an attractive package for game physicists. Adapting PBD for greater realism, and adapting force-based methods to achieve some of the positive qualities of PBD, is the major theme of

this dissertation.

1.3 Contributions

The first part of this dissertation explores the modelling of volumetric soft bodies in videogames. In Chapter 2 we cover some relevant continuum mechanics, which forms the basis of our soft body model. We alight on the co-rotational strain as a formidably *robust* deformation model. Computing this measure requires us to track the rotations of deformable bodies, which we cover in Chapter 3. We introduce a highly efficient, approximate method for tracking deformable body rotations. Continuing onwards, we cover discretization and the finite element method in Chapter 4. In Chapter 5 we bring the previous chapters together, discussing various solution procedures that have been used for the simulation of soft bodies in games. We introduce our new Position Based Finite Element (PBFE) method, which fuses the advantages of PBD simulation with the realism of the finite element method. We also make some modifications to the recently proposed method of Projective Dynamics, designed to make the method more suitable for game use. The spin-based rotation tracking of Chapter 3 and PBFE method of Chapter 5 have been submitted for patenting.

The second part of the dissertation concerns adding some aspects of deformable body simulation to a rigid body solver. Despite humans being relatively deformable, they are generally represented in games as collections of rigid bodies connected with joints. After discussing the mechanics of rigid body simulators in Chapter 6, we introduce in Chapter 7 a new type of constraint that models the kinematics of the human spine. This allows us to express a similar range of motion compared with models that use multiple joints for the spine, while decreasing the degrees of freedom required. This work was published at SCA 2013 [Lewin et al. 2013]

1.4 Use in Games

It is worth spending some time motivating our work by discussing how it could be used. The first part of this dissertation concerning soft body simulation is quite

abstract, so it can be difficult to see how it might be applied in a real game.

Inanimate objects are usually modelled as being perfectly rigid in games, which is often a good approximation. However, in many games the only meaningful interaction the player can have with the game world is through violence of some kind, and in this case the rigid indestructibility of the game world can break the player's immersion. To realistically simulate the destruction of inanimate objects, we must first simulate their deformation; the distribution of stress inside an object tells us how that object will fracture. Many materials can undergo a great deal of plastic deformation before failing (so-called 'ductile fracture'). Deformation is thus the key to allowing inanimate objects to react more realistically to the heavy weaponry with which many game characters are equipped.

Characters, both human and nonhuman, can also benefit from simulation of deformation. While there are many possible levels of detail at which one could apply soft body simulation, a very coarse application can noticeably improve the look of certain types of characters. Areas of a character that one might expect to deform and jiggle in response to motion, such as a fat belly, are hard to animate using traditional linear blend skinning. By applying a simple finite element simulation with only a few tens of degrees of freedom, we can add plausible secondary motion to these areas without any effort required from an animator. At a higher quality level, areas such as the lips that require a high level of realism to be convincing can be modelled entirely as soft bodies, with internal muscle fibers that respond to the animation. At a very high level of quality, we can construct an entire character from simulated flesh [McAdams et al. 2011].

Moving away from volumetric soft bodies, many other phenomena that one might want to simulate in real-time also show soft characteristics; hair and cloth are the two natural choices. Although in this dissertation we concentrate on three-dimensional bodies, the techniques covered apply directly to these other areas. The differences between simulations of cloth, hair and flesh are ultimately quite minor when viewed from a distance.

To summarise: soft body simulations can improve the realism of both characters and the scenery that surrounds them. There are some types of effects which are difficult to achieve unless physically simulated, and we need cheap, reliable

techniques that can believably reproduce these effects. Finally, the core details of soft body simulation can transfer to other important fields.

Chapter 2

Continuum Mechanics

The first and most fundamental approximation made when modelling deformable bodies is the continuum assumption: that we can represent a body made up of discrete atoms interacting in highly complex ways using smooth differential equations. This is a useful technique for modelling both solids and fluids, although here we will cover elastic solids exclusively. We will closely follow Sifakis's excellent SIGGRAPH course [Sifakis and Barbič 2012], which is a general introduction to the field with a computer graphics slant.

Our main aim in this chapter is to identify quantities which we can constrain to realistically simulate deformable bodies, in particular the *shape* and *volume* of 3D elastic solids. This is mostly a matter of identifying appropriate *strain measures*, which tell us in various ways the local deformation of a body from its rest pose. For constraints this is far enough, but a more traditional discussion of continuum mechanics would continue on to talk about stress and its differentials. This is interesting to us only for a comparison with force-based methods, so we refer the reader back to Sifakis for the details.

Serious continuum mechanics is often written down using indicial notations such as the summation convention, that ease some of the pain associated with high order tensors. In the fairly simple mechanics used in computer graphics, these notations are less common and authors tend to bend quite far to avoid using anything other than vectors and matrices. We will adopt the same approach for conformity's sake.

2.1 The Continuum Approximation

Consider an elastic deformable body in n dimensions. At any point inside the body we are interested in two positions - the *rest pose* coordinates $\mathbf{X} \in \mathbb{R}^n$ and the

deformed pose coordinates $\mathbf{x} \in \mathbb{R}^n$. The mapping between these coordinates is a vector field that we call the *deformation function* $\mathbf{x} = \boldsymbol{\phi}(\mathbf{X})$. We will cover a case in which both the deformed and undeformed coordinate systems are *curvilinear*; that is, there are an additional set of coordinates $\boldsymbol{\xi}$ such that $\mathbf{X} = \mathbf{X}(\boldsymbol{\xi})$ and $\mathbf{x} = \mathbf{x}(\boldsymbol{\xi})$. This allows us to treat theoretical problems with, for instance, polar coordinate systems; and later on to implement finite elements.

For simulation purposes we are primarily interested in the Jacobian matrix of the deformation function, which we call the *deformation gradient*:

$$\mathbf{F} = \frac{\partial \boldsymbol{\phi}}{\partial \mathbf{X}} = \frac{d\mathbf{x}}{d\boldsymbol{\xi}} \left(\frac{d\mathbf{X}}{d\boldsymbol{\xi}} \right)^{-1} = \mathbf{F}_{\mathbf{x}} \mathbf{F}_{\mathbf{X}}^{-1}. \tag{2.1}$$

We can see why this quantity is important when we consider the changes in length implied by the deformation $\boldsymbol{\phi}$. First let's consider a strongly related concept, which is the change in length implied from the curvilinear coordinate system to each of $\mathbf{x}$ and $\mathbf{X}$. Let there be two points separated by the small vector $d\mathbf{X}$. Then we have $d\mathbf{X} = \mathbf{F}_{\mathbf{X}} d\boldsymbol{\xi}$. The squared length of $d\mathbf{X}$ is

$$d\mathbf{X}^2 = \left(\mathbf{F}_{\mathbf{X}} d\boldsymbol{\xi}\right)^T \left(\mathbf{F}_{\mathbf{X}} d\boldsymbol{\xi}\right) = d\boldsymbol{\xi}^T \mathbf{C}_{\mathbf{X}} d\boldsymbol{\xi}, \tag{2.2}$$

where $\mathbf{C}_{\mathbf{X}} = \mathbf{F}_{\mathbf{X}}^T \mathbf{F}_{\mathbf{X}}$ is the *metric tensor* of the transformation between the curvilinear and rest coordinate systems. We can run the exact same derivation for the deformed configuration, giving

$$d\mathbf{x}^2 = d\boldsymbol{\xi}^T \mathbf{C}_{\mathbf{x}} d\boldsymbol{\xi}. \tag{2.3}$$

We can then easily see that the squared difference in length between the two configurations is

$$dl^2 = d\mathbf{x}^2 - d\mathbf{X}^2 = d\boldsymbol{\xi}^T \left(\mathbf{C}_{\mathbf{x}} - \mathbf{C}_{\mathbf{X}}\right) d\boldsymbol{\xi}. \tag{2.4}$$

Note that this quantity is measured in the curvilinear coordinate system. Depending on the application, this may or may not be useful. For simulation we are more interested in lengths in the deformed configuration, for which we need the metric tensor of the deformation $\boldsymbol{\phi}$:

$$\mathbf{C} = \mathbf{F}^T \mathbf{F}. \tag{2.5}$$

This tensor tells us about changes in the material when moving between the undeformed and deformed configurations; the underlying curvilinear coordinate

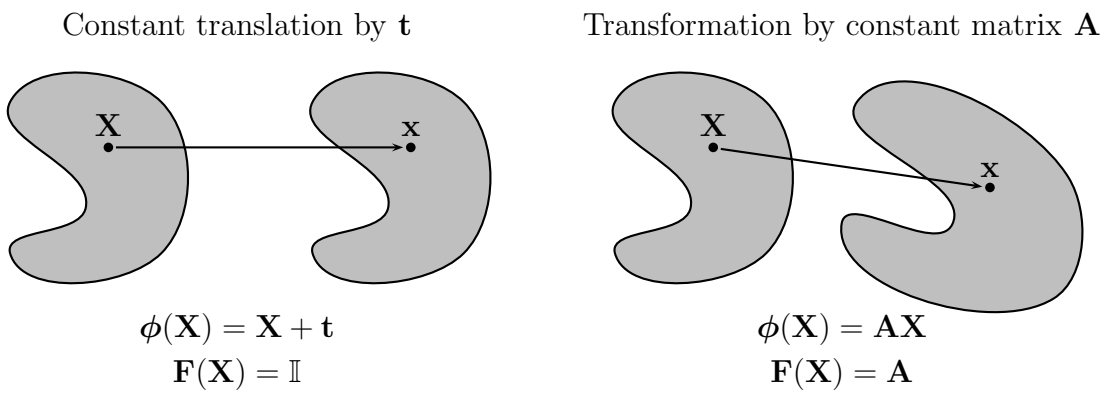

Figure 2.1: Sample deformation functions and their deformation gradients.

system is not referred to at all. The length change moving from $\mathbf{X}$ to $\mathbf{x}$ is:

$$dl^2 = d\mathbf{X}^T(\mathbf{C} - \mathbb{I})d\mathbf{X}. \tag{2.6}$$

We will return to this quantity when discussing strain measures below.

The qualities and behaviour of $\mathbf{F}$ are worth consideration because they form the basis of all behaviour of elastic continua. In Figure 2.1 we can see that this tensor behaves very much as a classical scalar derivative, but with an important caveat. When $\mathbf{F} = \mathbb{I}$ everywhere in the body, the object must be in its rest pose. This is true if $\boldsymbol{\phi}$ encodes only a constant translation, but a constant rotation will produce the same rotation in the deformation gradient: $\mathbf{F} = \mathbf{R} \neq \mathbb{I}$ even though there is clearly no deformation from the rest pose. $\mathbf{F}$ thus lacks the quality of *objectivity*, which is invariance under a rigid transformation. This is problematic when dynamically simulating soft bodies and we will return to it many times.

$\mathbf{F}$ transforms positions in the rest pose to positions in the deformed pose. This is called the *push forward* operation. The inverse *pull back* operation transforms from the deformed to rest pose and can predictably be achieved by multiplying by $\mathbf{F}^{-1}$. If the body is compressed to degeneracy in one or more axes then $\mathbf{F}$ will be singular and pulling-back will not be possible. The quantity $J = \det(\mathbf{F})$ is called the Jacobian determinant of $\mathbf{F}$; this gives the volume change of the deformation.

2.2 Strain Measures

$\mathbf{F}$ on its own is a representation of the *kinematics* of a deformable body. To simulate soft behaviour we need a representation of the *deformation*, which in continuum mechanics we call a *strain measure*. .Consider a simple spring, or *distance constraint* between two particles with positions $\mathbf{x}_a$ and $\mathbf{x}_b$. The kinematics are represented by the displacement $\mathbf{d} = \mathbf{x}_a - \mathbf{x}_b$, but to measure the strain we need to calculate the extension $l_e = |\mathbf{d}| - l_0$. Solving the constraint, or computing the spring forces, requires us to work with this nonlinear strain measure[1]. For the distance constraint this is trivial, but as we will see, the hunt for a well-behaved strain measure is much more complicated when we consider shells and volumes of deformable material.

We will see that like the distance constraint, the only well-behaved strain measures are nonlinear in $\mathbf{x}$. However, unlike that example there are some simple choices of linear strain measures that although flawed, can nonetheless be used to gain some insight.

2.2.1 Linear Strain Measures

A good strain tensor must accurately describe the deviation of the current pose from the rest pose. The deformation gradient seems promising for this task, but is not suitable on its own. Consider a generic *shape constraint* in which we force the strain measure to be zero:

$$\mathbf{E}(\mathbf{F}(\mathbf{X})) = \mathbf{0}. \tag{2.7}$$

We will slot various expressions for the strain tensor $\mathbf{E}$ into this expression and study the results. First consider the trivial strain $\mathbf{E} = \mathbf{F}$. With this strain the constraint is satisfied when $\boldsymbol{\phi}(\mathbf{X})$ is constant, i.e when the object is compressed into a single point. Clearly this quantity does not in fact measure the object's deviation from the rest pose. A better strain measure might be

$$\mathbf{E}(\mathbf{F}) = \mathbf{F} - \mathbb{I}. \tag{2.8}$$

[1] A real engineer would probably prefer the Cauchy strain $e = (|\mathbf{d}| - l)/l$, which has the advantage of being unitless and independent of the original length, but when taken as a constraint equation has exactly the same meaning as our expression.

In this case the constraint is satisfied when the body is at a constant translation from its rest pose. This can in fact be good enough for certain kinds of analysis (a slightly different derivation will lead to the infinitesimal strain tensor $\boldsymbol{\epsilon} = \frac{1}{2}(\mathbf{F}^T + \mathbf{F}) - \mathbb{I}$), but due to the non-objectivity of $\mathbf{F}$ under rotation this measure is not suitable for general use. In computer animation we expect our deformable models to be able to undergo large rigid motions, as well as large deformations, which invalidate this measure. The practical consequence of using a rotation-sensitive strain measure for finite simulation is the generation of spurious *ghost forces* that act to resist rotation. Unsurprisingly, we must give up linearity in $\mathbf{x}$ if we wish to measure finite[2] deformations.

2.2.2 Green Lagrange Strain

The simplest objective strain measure is the Green-Lagrange strain:

$$\mathbf{E}_{\mathrm{GL}} = \mathbf{C} - \mathbb{I}, \tag{2.9}$$

where $\mathbf{C} = \mathbf{F}^T\mathbf{F}$, called the *right Cauchy-Green deformation tensor*, is the metric of the deformation $\boldsymbol{\phi}$ discussed previously. Note that this tensor is quadratic in $\mathbf{x}$. We can easily show this measure is objective by introducing a pure rotation $\mathbf{R}$:

$$\mathbf{E}_{\mathrm{GL}}(\mathbf{R}) = \mathbf{R}^T\mathbf{R} - \mathbb{I} = \mathbf{0}. \tag{2.10}$$

This measure is mathematically well-grounded in the sense that it measures the difference in squared lengths of lines in the deformed and undeformed configurations. However it has a different fault unrelated to objectivity: it is also invariant under inversion. Consider a body whose deformed pose we have reflected in the X axis. The associated deformation gradient is $\mathbf{F}_{\mathrm{R}} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$. The strain under such a transform will be $\mathbf{F}_{\mathrm{R}}^T\mathbf{F}_{\mathrm{R}} - \mathbb{I} = \mathbf{0}$. Thus the body is perfectly happy to be in this configuration despite it being fully inverted in the x axis! Although not as egregious a fault as rotation sensitivity, this means that if the body is inverted, it will work to further invert itself until it reaches a mirror image of its rest pose.

[2]In continuum mechanics 'finite' means *large*, as opposed to infinitesimal. The boundary between these regimes is mostly determined by where the distortions introduced by using a strain measure like Equation 2.8 become noticeable.

Models based on Green-Lagrange strain suffer quite severely from this fault in practice, and the material is thus not much used for computer animation of solid mechanics[3].

2.2.3 Polynomial Strain

In engineering, Green-Lagrange strain is rarely used unmodified for a different reason: it simply does not represent real material behaviour very well. To improve this situation we can use the *invariants* of $\mathbf{C}$. These are scalar functions of the components of $\mathbf{C}$ that are themselves objective under rigid motion. For symmetric 3x3 tensors these are:

$$\begin{aligned} I_1 &= \mathrm{tr}(\mathbf{C}), \\ I_2 &= \frac{1}{2}\left(\mathrm{tr}\left(\mathbf{C}\right)^2 - \mathrm{tr}\left(\left(\mathbf{C}\right)^2\right)\right), \\ I_3 &= \det\left(\mathbf{C}\right). \end{aligned} \tag{2.11}$$

So-called *polynomial hyperelastic* models define strain energies (or in our case, constraints) for incompressible materials using combinations of these invariants. To define these models we first notice that I_3 is exactly the squared determinant of $\mathbf{F}$; thus it gives the squared volume change of the deformation. We can expoit this fact to define a new set of *isochoric* invariants that are both objective and insensitive to volume change:

$$\begin{aligned} \bar{I}_1 &= J^{-2/3} I_1, \\ \bar{I}_2 &= J^{-2/3} I_2. \end{aligned} \tag{2.12}$$

Because these quantities are isochoric, enforcing them as constraints (for example $\bar{I}_1 = \mathrm{tr}(\mathbb{I}) = 3$) will constrain only the *shape* of the material and not its volume. This is useful when the volume is constrained by other means, for instance by using a stiff penalty term as in the Neo-Hookean model used by [Irving, Teran, and Fedkiw 2004]:

$$\psi_{\mathrm{NH}}(\mathbf{F}) = C_1(\bar{I}_1 - 3) + D_0 \ln(J), \tag{2.13}$$

where C_1 and D_0 are stiffness constants for shape and volume preservation respectively. It should be noted that the term Neo-Hookean refers to a family of

[3]That is, in 3D. For 2D surfaces embedded in 3D (cloth and shells), inversion is a non-issue because the surface can simply bend instead of compressing

models that use I_1 and not I_2, and the exact method of volume preservation is subject to change; for instance, we could enforce $\det(\mathbf{F}) = 1$ through a Lagrange multiplier or through a different penalty term like $D_0(J-1)^2$. Models that also use I_2 go by the name of Mooney-Rivlin:

$$\psi_{\text{MR}}(\mathbf{F}) = C_1(\bar{I}_1 - 3) + C_2(\bar{I}_2 - 3) + \ D_0 \ln(J). \tag{2.14}$$

Again, the exact choice of volume constraint is left to the implementor.

2.2.4 Co-Rotational Strain

Polynomial hyperelastic models are fairly popular in high-end computer graphics, but they are fairly costly to simulate because the derivatives of the isochoric invariants involve the inverse of the deformation gradient, as well as inconvenient powers like $J^{-2/3}$. A much more popular model is the Co-Rotational strain:

$$\mathbf{E}_{\text{CR}}(\mathbf{F}) = \mathbf{F} - \mathbf{R}, \tag{2.15}$$

where $\mathbf{R}$ is the rotational component of $\mathbf{F}$, found via its Singular Value Decomposition (SVD):

$$\mathbf{F} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T, \tag{2.16}$$

where $\boldsymbol{U}$ and $\boldsymbol{V}$ are rotations and $\boldsymbol{\Sigma}$ is a diagonal matrix that contains the singular values of $\mathbf{F}$. $\mathbf{R}$ can be found by simply setting $\boldsymbol{\Sigma}$ to identity to remove the stretch, leaving only rotation:

$$\mathbf{R} = \boldsymbol{U}\boldsymbol{V}^T. \tag{2.17}$$

Thus an equivalent way to state the co-rotational strain is simply using the singular values themselves:

$$\mathbf{E}_{\text{CR}}(\mathbf{F}) = \boldsymbol{\Sigma} - \mathbb{I}. \tag{2.18}$$

A great advantage of this approach is that the singular values can be negative: if the body is reflected in the x axis as in our previous example the first singular value will be -1. Thus if the body is compressed to degeneracy and then inverted, we will maintain restoring forces that will push the body back to its original shape, and not its mirror image. Furthermore unlike cubic-elastic models such as Green-Lagrange strain, this measure does not vanish as we get closer to degeneracy. This

quality gives co-rotational elasticity a huge advantage in robustness over models based on $\mathbf{C}$.

Computing rotations of deformable bodies is not an exact science and the most popular method of computing the SVD is relatively expensive. Some approaches that have been taken, as well as our own method, are detailed in the next section.

2.3 Concluding Remarks

In this chapter, we covered some basic continuum mechanics up to a level required to implement a simple finite element analysis. We discussed various different strain measures, which tell us how far an object is locally deformed away from its rest state. In particular we discussed the co-rotational strain, which has become popular in the graphics field for its robustness. To use this strain measure, we require a reliable method for extracting rotations of our deformable bodies, which is addressed in the following chapter.

Chapter 3

Rotation Estimation

The problem of tracking the rotation of a deformable body is a surprisingly complex one. In this short chapter, we will cover various popular approaches and introduce our novel spin-based method. There are a great deal of different approaches to this problem in the engineering and computer graphics literature; the most important differentiator between them is how they deal with inversion. The most important quality of the co-rotational strain model from our perspective as game developers is the ability to recover from inversion, and methods that are quite suitable for engineering analysis can fail to pass this test.

3.1 Edge-based methods

One approach to co-rotation is to attach a rotation frame to each *node* of the finite element mesh. We can interpret this approach as calculating a 'rotation field' with the same discretization as the finite element position field. A common strategy for this is to choose some local directions and orthogonalize them in some way. Michels et al. [2014] choose a set of three edges $\mathbf{e}_1$, $\mathbf{e}_2$ and $\mathbf{e}_3$ connected to a vertex $\mathbf{x}_i$ and construct the following set of orthogonal vectors:

$$\begin{aligned} \mathbf{n}_1 &= \sum_{i=1}^{3} \mathbf{e}_i / |\mathbf{e}_i|, \\ \mathbf{n}_2 &= \mathbf{n}_1 \times \mathbf{e}_1 / |\mathbf{n}_1 \times \mathbf{e}_1|, \\ \mathbf{n}_3 &= \mathbf{n}_1 \times \mathbf{n}_2. \end{aligned} \tag{3.1}$$

The rotation of each frame is then constructed from the orthogonal sets of the current and previous frames:

$$\mathbf{R}^t = [\mathbf{n}_1^t \ \mathbf{n}_2^t \ \mathbf{n}_3^t] \cdot [\mathbf{n}_1^{t-1} \ \mathbf{n}_2^{t-1} \ \mathbf{n}_3^{t-1}]^T. \tag{3.2}$$

The authors point out that this gives the correct rotation in the presence of rigid motion only. However, it is likely that this method will produce reflections rather

than rotations if the local neighborhood is inverted - in other words, it is only guaranteed to produce an orthogonal frame and not necessarily a true rotation. The authors did not put their method through any particularly strenuous tests like recovery from total inversion, which likely meant this weakness was not apparent.

3.2 Polar Decomposition

Most literature on finite element simulation in computer graphics associates rotations with each *element* rather than each node. More precisely, each integration point at which we calculate the deformation gradient, of which there may be many per element, has an associated rotation. This is the matrix $\mathbf{R}$ that minimizes the difference between the deformation and an isometry [Chao et al. 2010]:

$$E_R = |\mathbf{F} - \mathbf{R}|_F^2. \tag{3.3}$$

Notice that this is exactly the co-rotational strain energy discussed in the previous chapter, except that we are notionally minimizing over $\mathbf{R}$ rather than $\mathbf{F}$. Clearly to minimize this energy we must find the rotation that most closely matches $\mathbf{F}$ in the Frobenius sense. [Hauth and Etzmuss 2001] propose to use the *polar decomposition* of $\mathbf{F}$ for this purpose:

$$\mathbf{F} = \mathbf{US}, \tag{3.4}$$

where $\mathbf{U}$ is orthogonal and $\mathbf{S}$ is a pure deformation. We can perform this decomposition in a slightly roundabout way by noting that $\mathbf{S}^2 = \mathbf{F}^T\mathbf{F}$, and that we can compute $\mathbf{U} = \mathbf{FS}^{-1}$. The only difficulty is computing the matrix square root of $\mathbf{S}^2$, which is straightforward since it will be positive-definite or singular. To do this we compute the eigenvalues λ_i and eigenvectors $\mathbf{v}_i$ of $\mathbf{S}^2$ by any method of our choice, and then $\mathbf{S}$ is simply:

$$\mathbf{S} = \sum_{i=1}^{m} \sqrt{\lambda_i}\mathbf{v}_i\mathbf{v}_i^T. \tag{3.5}$$

Computing the polar decomposition in this way is known as symmetric diagonalization. One popular method due to Rivers and James [2007] for calculating the eigenvalues uses Jacobi rotations warm-started with the previous frame's solution. With this, the diagonalization process can be fairly cheap despite requiring trigonometric calculations.

The greatest drawback of the polar decomposition is that the matrix $\mathbf{U}$ is *not* guaranteed to be an actual rotation; it is only guaranteed to be orthogonal. If the diagonalization process is applied to a deformation gradient that includes inversion, the result will be a reflection rather than a rotation. This method thus shares the reflection-insensitivity drawback with the edge-based methods.

3.3 Singular Value Decomposition

With some modifications, we can ensure that we obtain a true rotation for any input $\mathbf{F}$. Instead of computing $\mathbf{R} = \mathbf{F}\mathbf{S}^{-1}$, we compute the full singular value decomposition (SVD): $\mathbf{F} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$, where $\mathbf{U}$ and $\mathbf{V}$ are orthogonal and $\mathbf{\Sigma}$ is the diagonal matrix of *singular values*. Unlike the entries of the stretch matrix $\mathbf{S}^2$ the singular values can be negative, indicating that $\mathbf{F}$ includes a reflection in the relevant axis. Given the SVD, we can calculate the rotation simply by setting $\mathbf{\Sigma} = \mathbb{I}$, or in other words $\mathbf{R} = \mathbf{U}\mathbf{V}^T$. Rotations calculated in this manner are guaranteed to be true ($\det \mathbf{R} = 1$). Actually calculating the SVD is a complex matter and most published methods are designed for use on large, rank deficient matrices [Press et al. 2007]. McAdams et al. [2011] developed a SVD specifically for the 3×3 matrices encountered in co-rotational simulation, which we will refer to here as the *Fast SVD*. The idea is to perform eigen-decomposition of $\mathbf{S}^2 = \mathbf{F}^T\mathbf{F} = \mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^T$ as before, this time taking the eigenvectors $\mathbf{V}$ instead of computing the matrix square root. Then we can form the matrix $\mathbf{U}\mathbf{\Sigma} = \mathbf{F}\mathbf{V}$. Finally a QR decomposition of $\mathbf{U}\mathbf{\Sigma}$ will give us $\mathbf{U}$ and $\mathbf{\Sigma}$ separately. The authors manage to construct a method that avoids all complex functions, relying on a series of small-angle approximations. Their method is also completely branch-free, with no handling of special cases necessary. While warm-starting (as employed for the polar decomposition) was not explored by the authors, it seems plausible that we could decrease the number of iterations required in the same way.

3.4 Spin-Based Rotation Tracking

The element-based methods discussed so far rely on processing the deformation gradient $\mathbf{F}$ to extract rotational information. We instead take a geometric ap-

proach, operating on groups of particles rather than any kind of representative matrix. We call this method *spin-based* because of its use of skew-symmetric cross product matrices, rather than any connection to quantum mechanics. The central idea is to minimize the following energy over the rotation $\mathbf{R}$:

$$E_{\text{rot}} = \frac{1}{2}|\sum_{i=1}^{n} \mathbf{R}\mathbf{p}_i^0 \times \mathbf{p}_i|^2, \tag{3.6}$$

where there are n particles, each with a rest pose position $\mathbf{p}_i^0$ and deformed position $\mathbf{p}_i$ *relative to the barycenter* $\mathbf{p}_c$. To make things easier, we will split the rotation into a finite part $\mathbf{R}_F$ which we take as constant, and a small axis-angle vector perturbation $\mathbf{r}$ which we will optimise. This allows us to linearise the problem using small-angle approximations. The problem to be solved is thus:

$$\min_{\mathbf{r}} \frac{1}{2}|\sum_{i=1}^{n} \left(\mathbf{p}_i^R \cos|\mathbf{r}| + \left(\hat{\mathbf{r}} \times \mathbf{p}_i^R\right) \sin|\mathbf{r}| + \hat{\mathbf{r}}\left(\hat{\mathbf{r}} \cdot \mathbf{p}_i^R\right)(1 - \cos|\mathbf{r}|)\right) \times \mathbf{p}_i|^2, \tag{3.7}$$

where $\hat{\mathbf{r}} = \mathbf{r}/|\mathbf{r}|$, $\mathbf{p}_i^R = \mathbf{R}_F\mathbf{p}_i^0$ and we have used the Rodrigues rotation formula. We can linearise this expression using the small angle approximation (i.e. $\cos|\mathbf{r}| \approx 1$ and $\sin|\mathbf{r}| \approx |\mathbf{r}|$):

$$\min_{\mathbf{r}} \frac{1}{2}|\sum_{i=1}^{n} \left(\mathbf{p}_i^R + \mathbf{r} \times \mathbf{p}_i^R\right) \times \mathbf{p}_i|^2. \tag{3.8}$$

Clearly the expression is minimized when the quantity inside the norm is zero, which we can rearrange to:

$$\sum_{i=1}^{n} \left(-\mathbf{p}_i^{\times}\mathbf{p}_i^R\right) + \sum_{i=1}^{n} \left(\mathbf{p}_i^{R\times}\mathbf{p}_i^{\times}\right)\mathbf{r} = \mathbf{0}, \tag{3.9}$$

where $\mathbf{a}^{\times}$ is the skew-symmetric cross product matrix of the vector $\mathbf{a}$. The optimal extra rotation is thus:

$$\begin{aligned} \mathbf{r} &= \left(\sum_{i=1}^{n} \mathbf{p}_i^{R\times}\mathbf{p}_i^{\times}\right)^{-1} \sum_{i=1}^{n} \mathbf{p}_i^{\times}\mathbf{p}_i^R, \\ &= \mathbf{J}^{-1}\mathbf{s}. \end{aligned} \tag{3.10}$$

Note that actually applying this small rotation to $\mathbf{R}$ will have a nonlinear effect, and we will not find the exact answer in a single iteration. Our technique thus resembles a quasi-Newton method for minimizing the nonlinear energy E_{rot}. We find, however, that when running at 30fps only a single iteration is required for reasonable behaviour. Also note the similarities between this method and the

dynamics of rigid bodies; in particular the matrix $\mathbf{J}$ is exactly the inertia of the body if the rotated and deformed configurations are the same, and every particle has unit mass.

A significant difference between this method and the ones discussed previously is that this method requires that we maintain a current rotation $\mathbf{R}$ for each group we wish to track. While the method can converge from a poor initial estimate of $\mathbf{R}$ (such as the identity matrix) in some cases, problems tend to arise when the estimate is very far away from the optimum (for instance, a rotation by 180 degrees). We thus do not recommend using this method in situations where there is no history of the deformation of the body in question. However, we find that the quality of the rotations obtained by our method matches that of the much more expensive methods discussed previously. Crucially this method will only ever produce a true rotation, which means we avoid the problems associated with symmetric diagonalization. While our method will not in general produce the same rotation when applied to a finite element as SVDing the deformation gradient, we have found behaviour to be indistinguishable in most cases. A comparison of the rotations generated by our method and the alternatives when applied to tetrahedra can be seen in Figure 3.1.

A further approximation for small deformations

If we expect the deformation to be small as well as the relative rotation, we can further approximate the matrix $\mathbf{J}$ as the rotated pseudo-inertia of the rest state, which means we can avoid calculating its inverse:

$$\mathbf{J}^{-1} \approx \mathbf{R}\mathbf{J}_0^{-1}\mathbf{R}^T, \tag{3.11}$$

where $\mathbf{J}_0$ is the pseudo-inertia of the rest state. Although using this approximation requires us to store this matrix, we avoid forming and inverting the full expression which can substantially improve performance of the method. We have found that this is actually a surprisingly good approximation, staying valid up to quite high stretch ratios. While it is invalidated by very large elastic deformations or the presence of plasticity, we are comfortable using it for simulation of medium-strain elastic behaviour such as that seen in flesh simulation. The most objectionable consequence of using this approximation inappropriately is

Algorithm 1 Approximate rotation estimation for particle systems

function UPDATEROTATION($\mathbf{q}$)

$\mathbf{R} \leftarrow$ TOMATRIX($\mathbf{q}$)

$\mathbf{s} \leftarrow \sum_{i=1}^{n} (\mathbf{R}\mathbf{p}_i^0 \times \mathbf{p}_i)$

$\boldsymbol{\omega} \leftarrow \mathbf{R}\mathbf{J}_0^{-1}\mathbf{R}^T\mathbf{s}$

$\mathbf{u} \leftarrow \begin{pmatrix} 1 \\ \boldsymbol{\omega}/2 \end{pmatrix}$ *// Quaternion form, assuming small angles*

$\mathbf{q} \leftarrow (\mathbf{q}\mathbf{u}/|\mathbf{q}\mathbf{u}|)$ *// Using inexact* RSQRT

end function

that the optimal rotation can be catastrophically over-estimated, an error which will be compounded on subsequent time steps, leading to obvious oscillation and jittering in the simulation.

Implementation and performance

Using our medium-deformation approximation and a quaternion form for the rotation $\mathbf{q}$, we can implement a version of our algorithm (Listing 1) that requires minimal storage for the finite rotation and only a single inexact square root for error control. We use the small-angle approximation for converting $\boldsymbol{\omega}$ to a quaternion, which avoids computing the costly exponential map.

Comparing this algorithm to the gold standard Fast SVD discussed in the previous section, we found when applied to hexahedral finite elements our method took around 200ns per decomposition, whereas the Fast SVD took around 860ns. Our method is thus around four times faster than the alternative. While methods like that of McAdams et al. [2011] that require the actual singular values would not be able to use our rotation estimation, we have found that a standard co-rotational model using our rotation tracking retains the ability to un-invert itself without the aid of a strong volume constraint. This suggests that a stiffness model based on singular values is not necessary to achieve this important property. We also note that methods such as shape matching [Müller et al. 2005] whose most costly computations involve composing a matrix to be polar-decomposed could gain a substantial performance benefit from using our rotation tracking instead.

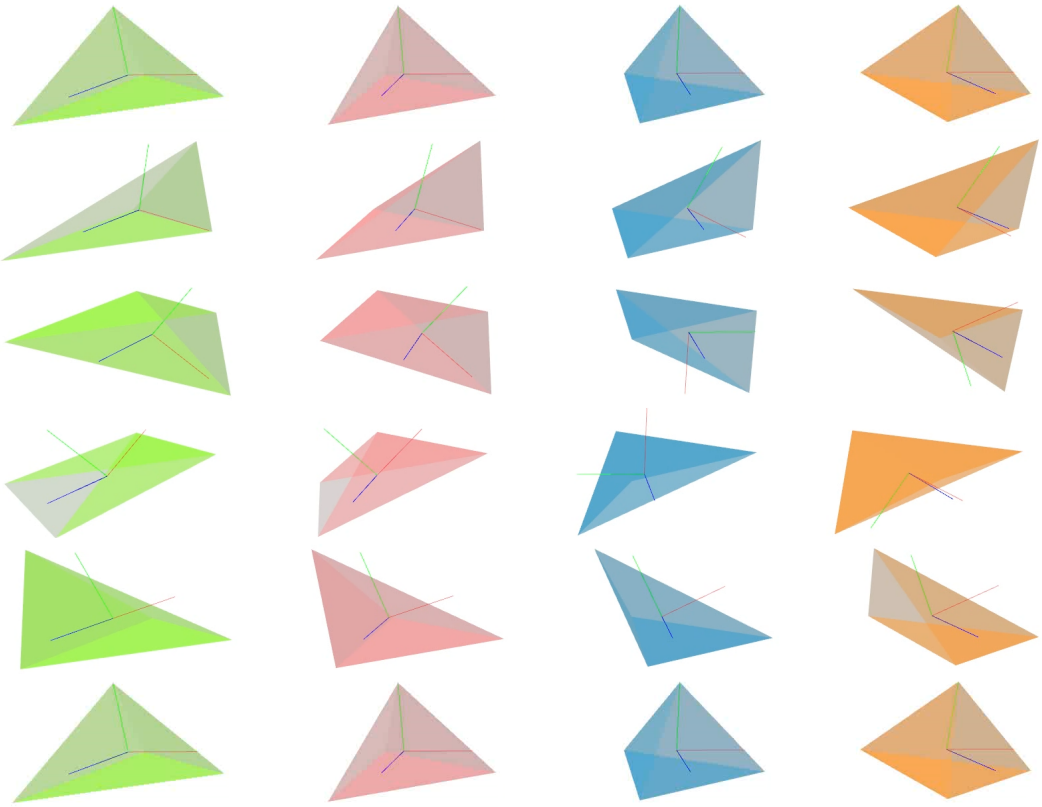

Figure 3.1: Comparison of rotation estimation for tetrahedra. We manipulate the top vertex of a tetrahedron, moving it clockwise around the base in each frame from top to bottom. We estimate the tetrahedron's rotation using, from left to right, our method with a warm start, our method with a cold start from identity, the fast SVD of McAdams et al. [2011], and symmetric diagonalization. Notice that the rotation provided by our method differs significantly from the results of the two matrix-decomposition based methods.

3.5 Concluding Remarks

In this chapter we discussed a number of different methods for tracking the rotation of a deformable body, before introducing our novel method. Our spin-based rotation tracking is as robust as the gold-standard SVD method while being several times cheaper, and is thus a good match for real-time simulation.

Recall that the reason we need rotation tracking is so that we can use it in the popular co-rotational strain metric, which is more robust than other simple metrics. In the next chapter, we will discuss how we can derive discrete finite element equations from the continuous continuum mechanics discussed in Chapter 2. We will then go on to link all this material together in Chapter 5, where we will discuss methods for actually simulating the deformation of soft bodies.

Chapter 4

Discretization

Traditional physics simulation for games revolves around rigid bodies, which can be described succinctly by their position, orientation, mass and so on. The rigid body approximation is thus in some sense already discrete - the approximation gives us the degrees of freedom we will use in simulation. With continuum mechanics this is no longer true; instead we have a continuous problem in which every point in space has a different strain function. To actually solve the problem on a computer we must approximate the infinite space of solutions to the continuous problem with a finite *discretization*. This is in itself a complex topic and the exact discretization chosen can have as much effect on the resulting behaviour of a continuum mechanics simulation as the material model[1].

The mathematical basis for solving differential equations discretization is provided by the theory of weak formulations, and finite elements are an example of Galerkin's method. We will not go deeply into these fundamentals, however, because most of the mathematical effort in these theories goes into proving convergence results in which we are not terribly interested.

4.1 Interpolation

Discretization is essentially a process of designing and implementing an appropriate *interpolation* scheme. The complexity of this task depends on the nature of the simulation domain; for instance on regular grids we can simply use linear interpolation and obtain the well known Finite Difference methods. However, in

[1]Note that one of the goals of serious finite element analysis is to eliminate this effect; different discretizations should agree in the limit of refinement and if they do not then there is something seriously wrong! However at the comparatively low level of detail we operate at in real-time, important differences can persist that would vanish if the mesh was refined.

computer graphics we are usually interested in irregular problems with complicated boundaries, and we thus require a more advanced interpolation scheme. The most common solution to this problem is to turn to the Finite Element method.

4.1.1 Finite Element Approximation

In this method we replace the symbolic position fields $\mathbf{X}$ and $\mathbf{x}$ with point samples $\mathbf{X}_i$ and $\mathbf{x}_i \in \mathbb{R}^3$, commonly called the nodes or vertices of the simulation. For notational convenience we can stack these vectors on top of each other to form the discrete state vectors $\mathbf{X}_d$ and $\mathbf{x}_d \in \mathbb{R}^{3n}$ where n is the number of nodes. We then introduce an interpolation function that lets us approximate the continuous variables:

$$\mathbf{X} = \boldsymbol{\Theta}(\mathbf{X}_d). \tag{4.1}$$

The function $\boldsymbol{\Theta}$ is in principle quite general; for instance every interpolated point could depend on every node in some nonlinear fashion. Various choices of $\boldsymbol{\Theta}$ can give you a Finite Difference method, or even highly unstructured methods like Smooth Particle Hydrodynamics. In the finite element method, however, we construct $\mathbf{X}$ as a sum of *basis functions* N_i:

$$\mathbf{X} = \sum_{i=1}^{n} N_i(\boldsymbol{\xi})\mathbf{X}_i, \tag{4.2}$$

where $\boldsymbol{\xi}$ is a vector of curvilinear coordinates inside each element. These may be barycentric coordinates on tetrahedra, or trilinear coordinates on hexahedra, or more exotic systems like cylindrical or prismatic coordinates. While any system of coordinates can be used in principle, we usually stick with simple barycentric coordinates in the computer graphics field for reasons of efficiency. An example can be seen in Figure 4.1.

Given this interpolation scheme we can also write the Finite Element approximation of $\mathbf{x}$:

$$\mathbf{x} = \sum_{i=1}^{n} N_i(\boldsymbol{\xi})\mathbf{x}_i. \tag{4.3}$$

By taking a time derivative we can obtain the interpolated velocity:

$$\mathbf{v} = \sum_{i=1}^{n} N_i(\boldsymbol{\xi})\mathbf{v}_i, \tag{4.4}$$

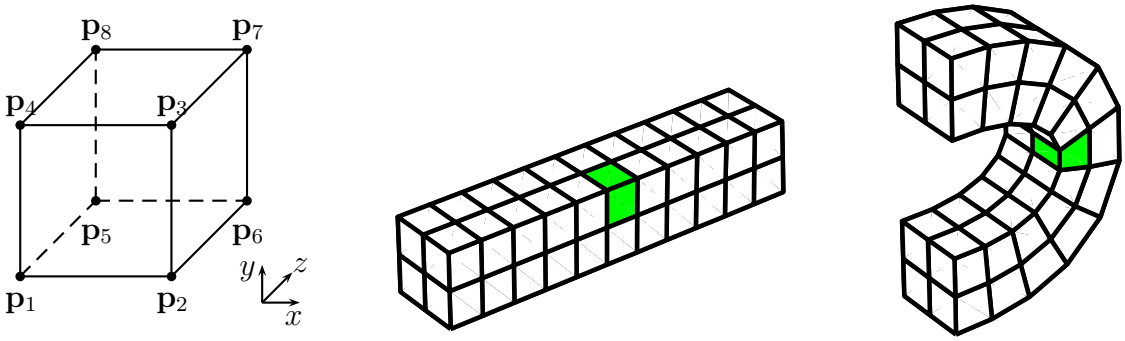

Figure 4.1: Reference, undeformed and deformed configurations for a linear hexahedral mesh. One element is highlighted.

where $\mathbf{v}_i$ is the velocity of node i.

4.1.2 Deformation Gradient

Recall from Chapter 2 that the deformation gradient $\mathbf{F} = \frac{\partial \mathbf{x}}{\partial \mathbf{X}}$ is an important quantity. We can obtain this easily using Equation 2.1:

$$\begin{aligned}
\mathbf{F} &= \frac{\partial \mathbf{x}}{\partial \mathbf{X}}, \\
&= \frac{\partial \mathbf{x}}{\partial \boldsymbol{\xi}} \frac{\partial \boldsymbol{\xi}}{\partial \mathbf{X}}, \\
&= \frac{\partial \mathbf{x}}{\partial \boldsymbol{\xi}} \left(\frac{\partial \mathbf{X}}{\partial \boldsymbol{\xi}} \right)^{-1}, \\
&= \left(\sum_{i=1}^{n} \frac{\partial N_i}{\partial \boldsymbol{\xi}} \otimes \mathbf{x}_i \right) \left(\sum_{i=1}^{n} \frac{\partial N_i}{\partial \boldsymbol{\xi}} \otimes \mathbf{X}_i \right)^{-1}, \\
&= \mathbf{D}_s \mathbf{D}_m^{-1},
\end{aligned} \tag{4.5}$$

where $\mathbf{a} \otimes \mathbf{b} = \mathbf{a}\mathbf{b}^T$ is the tensor product. The matrix $\mathbf{D}_m$[2] transforms from the reference shape to the undeformed shape. The matrix $\mathbf{D}_s$ transforms from the reference shape to the deformed shape. By using the inverse undeformed shape transform $\mathbf{D}_m^{-1}$ we form $\mathbf{F}$, which transforms from the undeformed shape to the deformed shape.

[2] $\mathbf{D}_m$ is sometimes called the element Jacobian and denoted with $\mathbf{J}$. We will avoid this notation to prevent conflict with other uses of the term 'Jacobian', which can refer to this specific finite element matrix, a more general matrix of partial derivatives or the determinant of either of these quantities.

Note the term $\frac{\partial N_i}{\partial \boldsymbol{\xi}}$ that appears in both shape transforms. We call the gradient of each shape function $\mathbf{b}_i = \frac{\partial N_i}{\partial \boldsymbol{\xi}}$. Stacking these vectors together we can form the matrix $\mathbf{B} \in \mathbb{R}^{3\times n} = [\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_n]$. Similarly we can stack the node vectors together to form the element state vector $\mathbf{x}_e \in \mathbb{R}^{3\times n} = [\mathbf{x}_{e1}, \mathbf{x}_{e2}, ...\mathbf{x}_{en}]$, where $\mathbf{x}_{ei}$ is the ith node of the eth element. Then we can express $\mathbf{F}$ more compactly as:

$$\mathbf{F} = \left(\mathbf{B}\mathbf{x}_e^T\right)\left(\mathbf{B}\mathbf{X}_e^T\right)^{-1}, \tag{4.6}$$

where of course $\mathbf{D}_s = \mathbf{B}\mathbf{x}_e^T$ and $\mathbf{D}_m = \mathbf{B}\mathbf{X}_e^T$.

4.1.3 Integration

So far we have shown how to interpolate continuous quantities from the discrete degrees of freedom. However, if our simulation is to be of much use we must be able to map these continuous quantities back to the nodes. In a force based simulation, for instance, we would define a total discrete elastic energy E as a sum of element energies E_e:

$$E(\mathbf{x}) = \sum_{e=1}^{m} E_e(\mathbf{x}_e), \tag{4.7}$$

where m is the number of elements. Each element energy is obtained by integrating over the local coordinates of each element:

$$E_e(\mathbf{x}_e) = \int_{\boldsymbol{\Omega}} \psi(\mathbf{x}, \mathbf{X}) d\mathbf{X}, \tag{4.8}$$

where $\psi(\mathbf{x}, \mathbf{X})$ is the elastic energy density defined in chapter 2. This integral looks, and indeed is, complex to evaluate; in the material models we have covered ψ is a linear or quadratic function of $\mathbf{F}$, which itself depends on the inverse of a matrix that may include $\boldsymbol{\xi}$ terms. For this reason we must often abandon analytic evaluation of this term in favor of numerical quadrature, which we shall cover later.

We can minimize this energy with, for example, Newton's method. We could also add a kinetic term $E_k = \sum_{i=1}^{n} m_i \mathbf{v}_i^2$ (where the mass has been lumped to the nodes), which would turn the problem into one of implicit integration. As has been labored, we are more interested in constraint-based methods. The exact procedure varies depending on the method, but whatever our choice we will end

up integrating quantities that look similar to energy terms and then finding their derivatives in the discrete setting. We will defer the discussion of constraint-based solution procedures, and instead talk about the specifics of different element choices.

4.2 Simplicial Elements

Triangles and tetrahedra are the dominant choice for finite element simulation in computer graphics. With the formulation discussed below, their implementation is very straightforward and does not require any numerical integration. Furthermore they match well with standard practice in computer graphics where surfaces are invariably represented as triangles for rendering purposes, and mesh creation algorithms are generally most well developed for triangles and tetrahedra. There are, however, disadvantages to simplicial elements that restricts them to mostly pedagogical use in computational science and engineering. These include the locking phenomenon, where these elements perform poorly under strong volume preserving forces (see Figure 4.2), and their inefficiency when compared with, for instance, one-point integrated linear hexahedra (Section 4.3). They are thus lacking in both speed and accuracy. Nevertheless they are uncomplicated and useful for learning.

4.2.1 Interpolation

Consider a tetrahedron whose four vertices have positions $\mathbf{p}_i$. We define the barycentric coordinate interpolation as the simple linear sum

$$\mathbf{p} = \mathbf{\Theta}(\boldsymbol{\lambda}) = \frac{\sum_{i=1}^{4} \lambda_i \mathbf{p}_i}{\sum_{i=1}^{4} \lambda_i} \tag{4.9}$$

where $\mathbf{p}$ is the interpolated point. Note that without an additional constraint these coordinates are not unique; the point with $\lambda_1 = 1, \lambda_{2,3,4} = 0$ is the same as the point $\lambda_1 = 2, \lambda_{2,3,4} = 0$. To remove the non-uniqueness we can specify that the λ_is must sum to 1. This allows us to remove one of the coordinates, for instance by stating $\lambda_4 = 1 - \lambda_1 - \lambda_2 - \lambda_3$. There are thus three unique barycentric coordinates, which we can simply map to our natural coordinates:

$\xi_i = \lambda_i,\ i \in 1, ..., 3$. By equation 4.9, the interpolation is then

$$\mathbf{p} = \mathbf{\Theta}(\boldsymbol{\xi}) = \xi_1 \mathbf{p}_1 + \ \xi_2 \mathbf{p}_2 + \ \xi_3 \mathbf{p}_3 + (1 - \xi_1 - \xi_2 - \xi_3)\mathbf{p}_4. \tag{4.10}$$

Comparing with equation 4.2, we can simply read off the basis functions:

$$\begin{aligned} N_1 &= \xi_1, \\ N_2 &= \xi_2, \\ N_3 &= \xi_3, \\ N_4 &= 1 - \xi_1 - \xi_2 - \xi_3. \end{aligned} \tag{4.11}$$

An alternative form for the interpolation can be useful:

$$\mathbf{p} = \mathbf{M}\boldsymbol{\xi}, \tag{4.12}$$

where

$$\mathbf{M} = (\mathbf{p}_1 - \mathbf{p}_4, \quad \mathbf{p}_2 - \mathbf{p}_4, \quad \mathbf{p}_3 - \mathbf{p}_4)^T \tag{4.13}$$

is the *edge matrix* of the tetrahedron; each column of $\mathbf{M}$ is the line from $\mathbf{p}_4$ to each other vertex of the element.

Deformation Gradient

The alternative form for the barycentric interpolation discussed above is useful when computing $\mathbf{F}$. Recall we need to compute the shape transform matrices between reference coordinates and both the undeformed and deformed configurations:

$$\mathbf{F} = \frac{\partial \mathbf{x}}{\partial \boldsymbol{\xi}} \left(\frac{\partial \mathbf{X}}{\partial \boldsymbol{\xi}} \right)^{-1}. \tag{4.14}$$

Substituting in our interpolation, we have:

$$\begin{aligned} \mathbf{F} &= \frac{\partial}{\partial \boldsymbol{\xi}} (\mathbf{M}_s \boldsymbol{\xi}) \left(\frac{\partial}{\partial \boldsymbol{\xi}} (\mathbf{M}_m \boldsymbol{\xi}) \right)^{-1}, \\ &= \mathbf{M}_s \mathbf{M}_m^{-1}, \end{aligned} \tag{4.15}$$

where

$$\mathbf{M}_s = (\mathbf{x}_1 - \mathbf{x}_4, \quad \mathbf{x}_2 - \mathbf{x}_4, \quad \mathbf{x}_3 - \mathbf{x}_4)^T,$$

and

$$\mathbf{M}_m = (\mathbf{X}_1 - \mathbf{X}_4, \quad \mathbf{X}_2 - \mathbf{X}_4, \quad \mathbf{X}_3 - \mathbf{X}_4)^T.$$

Figure 4.2: Demonstration of volumetric locking. Meshes of 855 and 3840 tetrahedra (left, center), and 171 one-point integrated hexahedra (right) are bent into a C shape under a volume constraint. The sawtooth pattern seen in the tetrahedral meshes is caused by volumetric locking, which is present whether the mesh is coarse or refined. Even at low resolution, the deformation produced by the hexahedral mesh remains smooth.

Looking back at the derivation of $\mathbf{F}$ in equation 4.5, we can see that for tetrahedra we have the simple relationship $\mathbf{D}_s = \mathbf{M}_s$ and $\mathbf{D}_m = \mathbf{M}_m$. Note that these shape transform matrices have *no dependency on* $\boldsymbol{\xi}$, or in other words that $\mathbf{F}$ is *constant* over each tetrahedron. All derived quantities of $\mathbf{F}$ will thus also be constant, which has led people to refer to this element as the *constant strain tetrahedron*.

Embedding

A problem peculiar to deformable body modelling in computer graphics is that of embedding rest-pose points (such as the vertices of a high-resolution visual mesh) inside the simulation mesh. This is simply the inverse interpolation operation: given a point $\mathbf{p}$ we seek natural coordinates $\boldsymbol{\xi}$ in $\mathbf{p}$'s enclosing element such that the interpolation operation maps $\boldsymbol{\xi}$ back to $\mathbf{p}$. For tetrahedra this is trivial:

$$\boldsymbol{\xi} = \mathbf{M}^{-1}\mathbf{p}. \tag{4.16}$$

As long as the tetrahedron is not degenerate, $\mathbf{M}$ is invertible and we can find local coordinates that map back to $\mathbf{p}$. Note that this is true for *any* $\mathbf{p}$, not just points inside the tetrahedron. If we allow negative ξ_i weights then the barycentric coordinate system can represent any point in space.

4.2.2 Integration

Because $\mathbf{F}$ is constant, integration of energies over $\mathbf{X}$ is possible analytically:

$$\begin{aligned} E_e &= \int_\Omega \psi(\mathbf{F}) d\mathbf{X}, \\ &= \psi(\mathbf{F}) \int_\Omega 1 \, d\mathbf{X}, \\ &= V_0 \psi(\mathbf{F}), \end{aligned} \tag{4.17}$$

where V_0 is the volume of the rest pose tetrahedron:

$$V_0 = \frac{1}{6} |\det(\mathbf{D}_m)|. \tag{4.18}$$

The volume of the reference tetrahedron is $\frac{1}{6}$, and $\det(\mathbf{D}_m)$ encodes the volume change between the reference and rest shapes.

4.3 Hexahedral Elements

A hexahedron is essentially a cube with the corner vertices arbitrarily perturbed. Hexahedral elements are generally the representation of choice for modelling materials in engineering, but have only recently become more popular in the computer graphics community. They represent a fair amount of additional complexity over tetrahedra, but when properly implemented they can exhibit superior performance in both the quality of their deformation and computational speed.

4.3.1 Interpolation

A natural choice for interpolation over the reference hexahedron is trilinear interpolation. For the node labelling shown in Figure 4.1, this scheme is:

$$\begin{aligned} \mathbf{p} = \boldsymbol{\Theta}(\boldsymbol{\xi}) \;=\; & (1-\xi_1)\,(1-\xi_2)\,(1-\xi_3)\,\mathbf{p}_1 \\ +\; & \xi_1\,(1-\xi_2)\,(1-\xi_3)\,\mathbf{p}_2 \\ +\; & \xi_1\,\xi_2\,(1-\xi_3)\,\mathbf{p}_3 \\ +\; & (1-\xi_1)\,\xi_2\,(1-\xi_3)\,\mathbf{p}_4 \\ +\; & (1-\xi_1)\,(1-\xi_2)\,\xi_3\,\mathbf{p}_5 \\ +\; & \xi_1\,(1-\xi_2)\,\xi_3\,\mathbf{p}_6 \\ +\; & \xi_1\,\xi_2\,\xi_3\,\mathbf{p}_7 \\ +\; & (1-\xi_1)\,\xi_2\,\xi_3\,\mathbf{p}_8. \end{aligned} \tag{4.19}$$

The shape functions can simply be read off this equation: $N_1 = (1-\xi_1)(1-\xi_2)(1-\xi_3)$ and so on.

Deformation Gradient

For tetrahedral elements there was a convenient alternative form for the interpolation function that made computing $\mathbf{F}$ particularly simple. Unfortunately this is not the case for hexahedra and we must grind the crank of Equation 4.5 to perform this calculation. This means finding the shape function gradient vectors $\mathbf{b}_i = \nabla N_i = \left[\frac{\partial N_i}{\partial \xi_1}, \frac{\partial N_i}{\partial \xi_2}, \frac{\partial N_i}{\partial \xi_3}\right]$. Fortunately our shape functions are simple enough that this is an easy proposition; for instance,

$$\mathbf{b}_1 = \nabla N_1 = \left[-(1-\xi_2)(1-\xi_3), -(1-\xi_1)(1-\xi_3), -(1-\xi_1)(1-\xi_2)\right].$$

Stacking the $\mathbf{b}_i$s together we obtain the shape derivative matrix $\mathbf{B}(\boldsymbol{\xi})$. Using equation 4.6, we can construct the deformation gradient as

$$\mathbf{F}(\boldsymbol{\xi}) = \mathbf{B}(\boldsymbol{\xi})\mathbf{x}_e^T \left(\mathbf{B}(\boldsymbol{\xi})\mathbf{X}_e^T\right)^{-1}. \tag{4.20}$$

Embedding

Embedding points for tetrahedra was trivial, amounting to a matrix inverse. It turns out that solving the embedding problem for linear hexahedra actually involves solving a nonlinear system of equations. As before, given a set of coordinates $\mathbf{p}$ in the undeformed pose we seek local coordinates $\boldsymbol{\xi}$ such that the trilinear interpolation operation maps $\boldsymbol{\xi}$ back to $\mathbf{p}$:

$$\boldsymbol{\Theta}(\boldsymbol{\xi}) = \mathbf{p}. \tag{4.21}$$

This is a nonlinear equation, so we turn to our favorite technique of Newton iteration to solve it. We introduce a guess at the solution $\boldsymbol{\xi}^k$ and a correction variable $\Delta\boldsymbol{\xi}$ so that

$$\boldsymbol{\Theta}(\boldsymbol{\xi}^k + \Delta\boldsymbol{\xi}) = \mathbf{p}. \tag{4.22}$$

Then we perform a Taylor expansion to first order about $\boldsymbol{\xi}^k$:

$$\boldsymbol{\Theta}(\boldsymbol{\xi}^k) + \frac{\partial \boldsymbol{\Theta}}{\partial \boldsymbol{\xi}}(\boldsymbol{\xi}^k)\Delta\boldsymbol{\xi} = \mathbf{p}, \tag{4.23}$$

and rearrange to obtain the correction:

$$\Delta\boldsymbol{\xi} = \left(\frac{\partial\boldsymbol{\Theta}}{\partial\boldsymbol{\xi}}(\boldsymbol{\xi}^k)\right)^{-1}\left(\mathbf{p} - \boldsymbol{\Theta}(\boldsymbol{\xi}^k)\right). \tag{4.24}$$

Then we simply iterate ($\boldsymbol{\xi}^{k+1} = \boldsymbol{\xi}^k + \Delta\boldsymbol{\xi}$) until $\boldsymbol{\Theta}(\boldsymbol{\xi}^k) - \mathbf{p}$ is sufficiently small. For hexahedra $\boldsymbol{\Theta}(\boldsymbol{\xi})$ is given by Equation 4.19, and the Newton step is:

$$\Delta\boldsymbol{\xi} = \left(\mathbf{B}(\boldsymbol{\xi}^k)\mathbf{X}_e^T\right)^{-1}\left(\mathbf{p} - \boldsymbol{\Theta}(\boldsymbol{\xi}^k)\right). \tag{4.25}$$

With an initial guess of $\boldsymbol{\xi}^0 = [0.5, 0.5, 0.5]$, this scheme converges within a few iterations as long as the element geometry is well-formed.

4.3.2 Integration

Because $\mathbf{F}(\boldsymbol{\xi})$ and its derived quantities vary over each element, integration over each element is no longer trivial as it was in the case of tetrahedra. In fact, to make any headway at all we must use numerical integration, also known as *quadrature*. The main idea behind quadrature is that we can approximate the integral of a function over some interval by taking samples of that function. The trapezium rule, Simpson's rule and so on are all examples of quadrature rules. A full discussion of quadrature is beyond the scope of this document (see [Belytschko, Liu, and Moran 2000]), but the main takeaway is that we can design quadrature rules that are *exact* for any given class of polynomial function. This is on the one hand good because it eliminates a potential source of error: as long as we use a sufficiently accurate quadrature rule, we can evaluate the elemental elastic energy exactly. On the other hand, exact quadrature can be extremely expensive. Much work in engineering, and more recently in the computer graphics field, has been spent on finding the absolute minimum amount of computation required to accurately simulate the deformation of linear hexahedra. It turns out this is possible with only a single quadrature point as long as we carefully control the resulting *hourglass deformations*. These are objectionable deformation modes that can propagate unopposed if not controlled (See Figures 4.3 and 4.4). This means that carefully implemented hexahedra are substantially more efficient than tetrahedra.

Eight Point Gauss Quadrature

First, let's consider the 'correct' integration rule. Most literature dealing with these elements [Belytschko, Liu, and Moran 2000]; [Belytschko and Bindeman 1993] specifies eight-point Gauss quadrature as a minimum. We can write this integration rule as:

$$E_e = \frac{1}{8}\sum_{i=1}^{8} \psi(\mathbf{F}(\boldsymbol{\xi}_i)), \tag{4.26}$$

where the integration points $\boldsymbol{\xi}_i$ are given by the rows of the matrix

$$\boldsymbol{\Xi} = \frac{1}{6}\begin{bmatrix} 3-\sqrt{3} & 3-\sqrt{3} & 3-\sqrt{3} \\ 3+\sqrt{3} & 3-\sqrt{3} & 3-\sqrt{3} \\ 3+\sqrt{3} & 3+\sqrt{3} & 3-\sqrt{3} \\ 3-\sqrt{3} & 3+\sqrt{3} & 3-\sqrt{3} \\ 3-\sqrt{3} & 3-\sqrt{3} & 3+\sqrt{3} \\ 3+\sqrt{3} & 3-\sqrt{3} & 3+\sqrt{3} \\ 3+\sqrt{3} & 3+\sqrt{3} & 3+\sqrt{3} \\ 3-\sqrt{3} & 3+\sqrt{3} & 3+\sqrt{3} \end{bmatrix}. \tag{4.27}$$

Unsurprisingly, this scheme is computationally expensive; the single evaluation of $\mathbf{F}$ and its derived quantities for tetrahedra must be performed eight times for hexahedra! While some researchers have used this scheme in their work, a casual observer might conclude that higher-order elements are guaranteed to be computationally more expensive than constant-strain tetrahedra. However, this is not the case because we can in fact achieve acceptable behaviour with substantially fewer than eight integration points.

One Point Quadrature

A naïve one point quadrature simply treats $\mathbf{F}$ as if it were constant and uses a single integration point at the element center $\boldsymbol{\xi}_\mathrm{c} = [0.5, 0.5, 0.5]$. We call the deformation gradient evaluated at this point

$$\mathbf{F}_0 = \mathbf{F}(\boldsymbol{\xi}_\mathrm{c}) = \mathbf{B}_0\mathbf{x}_e^T\left(\mathbf{B}_0\mathbf{X}_e^T\right)^{-1}, \tag{4.28}$$

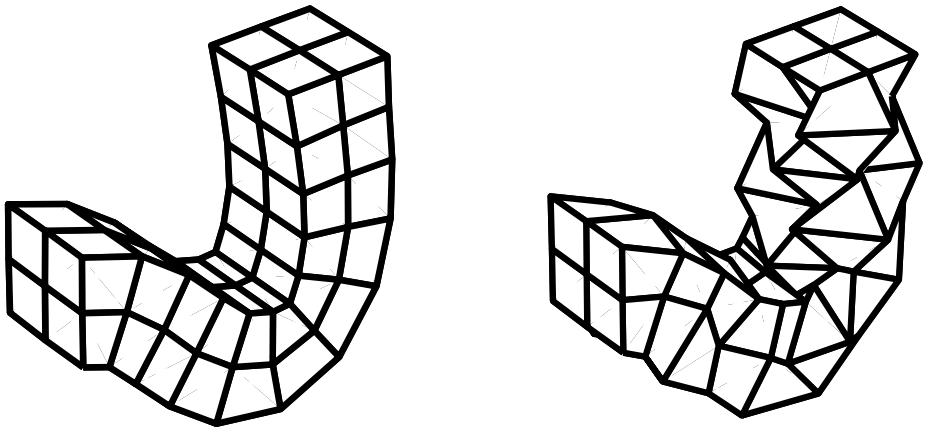

Figure 4.3: Deformations of a hexahedral FE mesh with and without hourglass control (left and right respectively).

where

$$\mathbf{B}_0^T = \mathbf{B}(\boldsymbol{\xi}_c)^T = \frac{1}{4}\begin{bmatrix} -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 \\ -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \end{bmatrix}. \tag{4.29}$$

This minimal quadrature scheme can actually be enough for certain types of analysis in engineering, but is in general insufficient due to the presence of objectionable artifacts often called 'hourglass modes'. These are deformations of the element that do not cause any change in $\mathbf{F}_0$; they are not 'seen' by the discrete energy and can thus propagate unopposed. An example of severe hourglassing can be seen in Figure 4.3.

Because one-point quadrature *almost* works, much effort has been invested in finding the minimal amount of extra computation required to eliminate the hourglass modes.

Hourglass Modes

What exactly are the hourglass modes? Simply put, they are deformations of the element's nodes that have no effect on the deformation gradient tensor. For instance, consider a unit hexahedron subjected to a deformation $\mathbf{x}_e = \mathbf{X}_e + \mathbf{Q}$, where

$$\mathbf{Q}^T = \begin{bmatrix} 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{4.30}$$

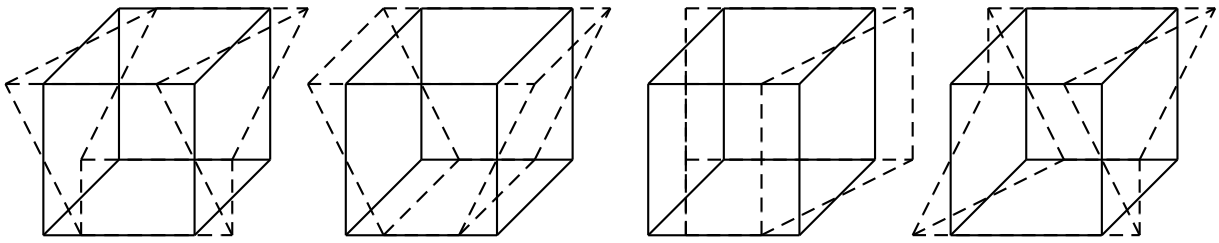

Figure 4.4: Hourglass modes in the x direction, reproduced from [Belytschko, Liu, and Moran 2000].

The resulting deformation gradient at the element center is:

$$\mathbf{F}_0 = \mathbb{I} + \mathbf{B}_0\mathbf{Q}^T. \tag{4.31}$$

Note we have used the fact that $\mathbf{B}_0\mathbf{X}_e^T = \mathbb{I}$ when $\mathbf{X}_e$ is the same as the reference configuration. It is tedious but simple to verify that $\mathbf{B}_0\mathbf{Q}^T = \mathbf{0}$, and thus no deformation in this mode will have any effect on the strain energy. We could excite this mode in the y or z axes and get the same result. All 12 hourglass modes for one-point integrated hexahedra are given by applying the rows of the matrix $\mathbf{H}$ to the x, y or z coordinates of the nodes:

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & -1 & -1 & -1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 & 1 & -1 & 1 & -1 \end{bmatrix}. \tag{4.32}$$

Figure 4.4 shows the x direction modes for a cubic hexahedron.

4.3.3 Hourglass Control

Staggered Grid Method

McAdams et al [2011] introduced the computer graphics field to the concept of hourglass control. They developed a method based on the successful use of staggered grids in Eulerian fluid simulation, expressing the $\boldsymbol{\xi}_1$ derivatives at the centers of the $\boldsymbol{\xi}_1$ faces of the element, and so on. This in effect introduces eight extra quadrature points, but only on the Laplacian term $\|\mathbf{F}\|_F^2$, which as the authors note has a constant stiffness matrix. This scheme is an example of what

an engineer would call *selective reduced integration* (SRI): the pressure terms of the co-rotational strain are separated out and integrated using the one-point approximation while the Laplacian term is given the full-quadrature treatment. The choice of face-centered integration points is non-standard - a more normal SRI scheme would simply use the Gauss quadrature nodes (Equation 4.27).

Because McAdams et al. only consider cubic elements[3] with no plasticity, they do not discuss the application of their control method to arbitrarily shaped hexahedral elements. They also do not compare to any of the standard hourglass control strategies used in engineering. A question that we might ask, then, is: is this method fundamentally different to the assumed strain stabilization methods presented in the next section, and if so is it superior? We will attempt to answer this question after briefly covering the field of hourglass control in engineering finite element analysis.

Perturbation Method

Belytschko [1993]; [2000] presented several methods for stabilizing one-point integrated hexahedra. These methods all rely on calculating the hourglass basis vectors $\boldsymbol{\gamma}_i$ for each element, which we can compute as follows:

$$\boldsymbol{\gamma}_i = \frac{1}{8}\left[\mathbf{H}_i - \sum_{j=1}^{3}(\mathbf{H}_i \cdot \mathbf{X}_j)\mathbf{b}_j\right], \tag{4.33}$$

where $\mathbf{H}_i$ is the ith row of $\mathbf{H}$, $\mathbf{X}_j$ is the jth row of $\mathbf{X}$ (i.e $\mathbf{X}_1$ is the vector of x coordinates of the element nodes) and $\mathbf{b}_j$ is the jth row of $\mathbf{J}_0^{-1}\mathbf{B}_0$. There are a number of ways we can utilise these basis vectors to control hourglass deformation, but the simplest is to directly compose an hourglass strain energy E_{HG} which we can include in our material model:

$$E_{\text{HG}} = C_{\text{HG}} V_0 \sum_{i=1}^{4} \|\boldsymbol{\gamma}_i \mathbf{x}_i^T\|_2^2, \tag{4.34}$$

where C_{HG} is a material parameter that represents the stiffness of the hourglass modes. This energy is quadratic in $\mathbf{x}_i$, and thus the associated forces are linear and the stiffness matrix constant. Unfortunately, the standard problem with

[3]Indeed their method as stated has much in common with a finite difference scheme, and was actually presented as one in an earlier incarnation: see [Zhu et al. 2010].

linear materials is still present, which is that this energy is non-objective. Thus it really needs to be applied in a co-rotated coordinate system (as described in the next chapter) to not introduce distortions in the context of finite deformation. Another problem with this method is the parameter C_{HG} is quite arbitrary; it is separate from the definition of our material model and not particularly physical. Ideally the hourglass modes should participate in the deformation in the exact same manner as the volumetric modes. However, this method is sufficient for simple use in computer graphics where we are not overly concerned with accuracy.

4.3.4 Other Elements

So far in this section we have only covered linear polyhedral elements, but we can in fact use almost any interpolation or approximation scheme as an element. Higher-order polynomial elements - quadratic tetrahedra or hexahedra, for instance - are popular in engineering for use in simulations of incompressible materials. These elements have been tried in the computer graphics field [Bargteil and Cohen 2014], but because we are generally not interested in strict volume preservation the advantages have so far been outweighed by the increased cost of simulating with these element types.

Another element type that has recently become popular for simulating shells is based on subdivision surfaces. This method of surface definition, developed in the early days of the computer graphics field [Catmull and Clark 1978], turns out to be an ideal basis for finite element analysis of thin shells since it provides excellent guarantees of continuity without requiring extra nodal degrees of freedom [Cirak, Ortiz, and Schröder 2000]. Despite its origin, this method has not yet become popular for computer animation.

4.4 Concluding Remarks

In this chapter, we discussed how to go from continuum mechanics principles to discrete equations corresponding to particular interpolation schemes. We also discussed how to invert these interpolation schemes, which is useful when embedding high-detail render geometry into low-resolution simulation meshes. In the next chapter, we will pull all the preceding material together and discuss how to

actually simulate soft body dynamics.

Chapter 5

Solution Procedures

In this chapter, we discuss a number of methods for simulating soft body dynamics, including our novel Position Based Finite Element method (Section 5.3). We also discuss a significant extension to a recently introduced method (Section 5.5) that makes it significantly more useful for real-time simulation.

5.1 Newton's Method

The most straightforward, principled and popular method for simulating soft body physics is to express the behaviour of our system as a discrete energy, and then minimize that energy using Newton's method.

5.1.1 Static Case

Let's first cover the static case, and generalise to dynamics later. Recall that in the discrete settings, $\mathbf{x}$ is the stacked vector of node positions. Given a discrete energy $E(\mathbf{x})$, we want to solve the problem

$$\mathbf{x} = \min_{\mathbf{x}} \left(E(\mathbf{x}) \right). \tag{5.1}$$

This minimum is reached when the energy gradient (i.e the sum of internal forces) is equal to zero:

$$\frac{\partial E(\mathbf{x})}{\partial \mathbf{x}} = \mathbf{f}(\mathbf{x}) = \mathbf{0}. \tag{5.2}$$

To get useful results from this analysis we also introduce an external force vector, which can represent gravity or an applied load:

$$\mathbf{f}(\mathbf{x}) - \mathbf{f}_{\text{ext}} = \mathbf{0} \tag{5.3}$$

If $E(\mathbf{x})$ is quadratic, $\mathbf{f}$ will be linear and we can solve this equation directly. In most interesting cases, however, $\mathbf{f}(\mathbf{x})$ will be nonlinear. There are a number

of methods to solve (5.2), some of which we will cover later. To use Newton's method, we introduce a candidate solution $\mathbf{x}^k$ and correction $\Delta\mathbf{x}$ such that the solution is approximated by $\mathbf{x}^k + \Delta\mathbf{x}$:

$$\mathbf{f}(\mathbf{x}^k + \Delta\mathbf{x}) - \mathbf{f}_{\text{ext}} = \mathbf{0}. \tag{5.4}$$

Then we perform a first order Taylor expansion around $\mathbf{x}^k$ and rearrange:

$$\mathbf{K}(\mathbf{x})\Delta\mathbf{x} = \mathbf{f}_{\text{ext}} - \mathbf{f}(\mathbf{x}^k), \tag{5.5}$$

where $\mathbf{K}(\mathbf{x}) = \frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}}$ is the *stiffness matrix*[1]. Finding the Newton update $\Delta\mathbf{x}$ requires us to solve this equation. However without modification, $\mathbf{K}$ will be singular and non-invertible: it contains rigid body modes that mean there is no unique solution. To obtain a static solution we must set a boundary condition on at least one node. This means, in short, setting the bound node's rows in $\mathbf{K}$ to identity, and overwriting the node's entries in the force vector with the desired position. Appendix A covers some basic linear algebra that can be used to solve systems of this type.

5.1.2 Dynamic Case

When simulating the time evolution of - or *integrating* - a system, one does not necessarily have to form and solve any particular system of equations. So-called *explicit* integrators simply calculate the forces applied to each body at each time step and step the system forward based on these forces. Such methods have difficulty with robustness, since the time step must generally be lower than the highest frequency in the simulation to avoid numerical explosions. While explicit methods can be the right choice for some kinds of highly nonlinear problems, the problems we deal with in real-time soft body simulations are only mildly nonlinear but often extremely stiff (i.e. the natural frequencies of the simulation are very high). We therefore prefer *implicit* methods, which are unconditionally stable.

[1] Also called the *tangent stiffness matrix* and *energy Hessian*; in the first case because it represents a tangent plane to the strain energy field at the point $\mathbf{x}$, and in the second because it is the second derivative of the strain energy.

The most straightforward implicit integration scheme is the *implicit Euler* method. In the static case it was easiest to start with a minimum energy principle, but in the dynamic case it is more convenient to start at the force level:

$$\mathbf{x}^{t+\Delta t} = \mathbf{x}^t + \Delta t \mathbf{v}^{t+\Delta t}, \tag{5.6}$$

$$\mathbf{v}^{t+\Delta t} = \mathbf{v}^t + \Delta t \mathbf{M}^{-1} \left(\mathbf{f}(\mathbf{x}^{t+\Delta t}) + \mathbf{f}_{\text{ext}} \right), \tag{5.7}$$

where $\mathbf{x}^t$ is the stacked position vector of the system at time t, $\mathbf{v}^t$ is the stacked velocity vector, Δt is the (fixed) time step and $\mathbf{M}$ is the mass matrix. We can rearrange the equations to emphasize the position level:

$$\mathbf{M} \left(\mathbf{x}^{t+\Delta t} - \mathbf{x}^t - \Delta t \mathbf{v}^t \right) = \Delta t^2 \left(\mathbf{f}(\mathbf{x}^{t+\Delta t}) + \mathbf{f}_{\text{ext}} \right), \tag{5.8}$$

$$\mathbf{v}^{t+\Delta t} = \Delta t^{-1} \left(\mathbf{x}^{t+\Delta t} - \mathbf{x}^t \right). \tag{5.9}$$

It is easy to show that the position equation (5.8) is the first order optimality condition of the minimization problem:

$$\mathbf{x}^{t+\Delta t} = \min_{\mathbf{x}^{t+\Delta t}} \left(\frac{1}{2} \Delta t^{-2} \| \mathbf{M}^{\frac{1}{2}} \left(\mathbf{x}^{t+\Delta t} - \mathbf{x}^* \right) \|_2^2 + E(\mathbf{x}^{t+\Delta t}) \right), \tag{5.10}$$

where $\mathbf{x}^*$ is the unconstrained solution, i.e the positions of each particle at the end of the time step in the absence of any internal forces:

$$\mathbf{x}^* = \mathbf{x}^t + \Delta t \mathbf{v}^t + \Delta t^2 \mathbf{M}^{-1} \mathbf{f}_{\text{ext}}. \tag{5.11}$$

The first term in (5.10) represents the total work done on all bodies during the time step. Intuitively, the minimization represents a balance between the elastic and kinetic energies. As in the static case, to solve this system we introduce a series of candidate solutions $\mathbf{x}_{k+1}^{t+\Delta t} = \mathbf{x}_k^{t+\Delta t} + \Delta \mathbf{x}$, and linearise around the target. After some rearranging, we end up with:

$$\left(\frac{1}{\Delta t^2} \mathbf{M} + \mathbf{K}(\mathbf{x}_k^{t+\Delta t}) \right) \Delta \mathbf{x} = \frac{1}{\Delta t^2} \mathbf{M} \left(\Delta t \mathbf{v}^t + (\mathbf{x}_k^{t+\Delta t} - \mathbf{x}^t) \right) + \mathbf{f}(\mathbf{x}_k^{t+\Delta t}) + \mathbf{f}_{\text{ext}}, \tag{5.12}$$

whose terms we will call

$$\mathbf{A}_k \Delta \mathbf{x} = \mathbf{b}_k. \tag{5.13}$$

The matrix $\mathbf{A}$ will be full-rank due to the inclusion of $\mathbf{M}$, so there is no need to set boundary conditions in a dynamic analysis. However, we can still use the

same procedure as in the static case if we wish to constrain nodes for other reasons. When we are happy with our solution, we can obtain the velocity $\mathbf{v}^{t+\Delta t}$ by (5.9). The number of Newton iterations required is generally very low in dynamic analysis, since we can use the ballistic solution as an initial guess: $\mathbf{x}_0^{t+\Delta t} = \mathbf{x}^*$. In fact, a notable variant of implicit Euler integration [Baraff and Witkin 1998] uses only one Newton step at each time step. This approach, called *semi-implicit* integration, makes no attempt to genuinely solve (5.10), instead simply replacing (5.8) with its linearisation every frame. In effect, we solve a linear model of the problem rather than the problem itself. This works in many cases because the consequences of not solving the nonlinear problem (5.10) exactly are not particularly dire. Generally, the most noticeable artifact is substantial extra damping, proportional to the nonlinearity of the method. For some applications this can be quite objectionable, but for others it is easy to ignore.

5.1.3 Constraints and Contacts

Before talking about problems in the implementation of Newton's method, we should briefly discuss a *fundamental* limitation of this approach. In short, our ability to express constraints purely as energy terms is very limited. We will discuss the *contact problem* in more detail later, but the long and short of it is that rigid contact between two objects can only be adequately described using inequality constraint conditions. If we wish to include such a condition in the minimization problem (5.10), we must express it as a twice-differentiable function and add it to the stiffness matrix. This is called the *implicit penalty method*. An example of such an energy for a one-sided contact might be:

$$E_{\mathrm{c}}(\mathbf{x}_i) = k_c(\mathbf{x}_i \cdot \hat{\mathbf{n}} - d)^2, \tag{5.14}$$

which models an attraction of the particle $\mathbf{x}_i$ towards the contact distance d in the contact normal direction $\hat{\mathbf{n}}$. k_c is a stiffness parameter for the contact. Note that this contact model is *sticky*: points that do not penetrate will be attracted towards the contact manifold as well as points that do. Stickiness is the most obvious and egregious consequence of using the penalty method for inequalities, but there are other more subtle problems with this choice. To obtain reasonable

behaviour, k_c must be quite large: it has to override the internal forces to prevent the contacting particles being driven through the contact manifold. Even with extremely high values of k_c, there will always be some interpenetration. Furthermore, increasing k_c to minimize this violation will negatively effect the condition number of $\mathbf{A}$, which will increase the number of conjugate gradient iterations required to solve (5.12). Interpenetration artifacts are thus an inevitable consequence of implicit contact modelling.

5.1.4 Problems with Newton's Method

Quite separate to the contact problem, the implementation of Newton's method is littered with pitfalls that can trap the unwary. We will discuss two important ones here.

Computing K

We have glossed over the specifics of $\mathbf{K}$ in this chapter, but it bears mentioning that the computation of $\mathbf{K}$ itself can be both intellectually and computationally demanding. For an example of the first criticism, note the contortions required to differentiate the SVD in [McAdams et al. 2011]. For the second, note the recent interest in quasi-Newton methods for FEM simulation in computer graphics [Hecht et al. 2012]. Dealing with stiffness matrices is, in short, a painful experience.

Positive Definiteness of K

We rely on $\mathbf{K}$ being positive definite to be able to use our favorite tools of Cholesky decomposition and Preconditioned Conjugate Gradients (discussed in Chapter A). However this is only guaranteed to be true at the solution; the further away our guesses are the more likely one of the $\mathbf{K}$ matrices encountered in the Newton solution process is to be indefinite, with one or more negative eigenvalues. To counter this we must ensure every element's contribution to $\mathbf{K}$ is strictly positive definite. Rivers and James [2007] achieved this by explicit eigen-analysis of each element's contribution, while McAdams et al. [2011] proposed an alternative perturbation that avoids the need to construct these matrices explicitly.

Methods such as those below that do not perform any indefiniteness correction suffer from noticeable artifacts when the material is stretched.

5.2 Fast Finite Elements

The previous section derived a principled time integration method for elastic problems. As one might expect, the method we developed is not optimally efficient. Much effort in the computer graphics field has been invested in finding the most efficient formulation of implicit FEM, resulting in a cloud of literature that we will refer to as *fast finite elements*. The only deformation system with an academic record of use in games [Parker and O'Brien 2009] falls squarely into this category, so it makes sense to describe the simplifications and hacks that have been found to be useful.

Stiffness Warping

We previously discussed the co-rotational material model (Section 2.2.4). Although we presented it as a fully nonlinear energy, the co-rotational method was in fact developed in the engineering literature as a post-fix for *linear* materials. This was rediscovered in the computer graphics field [Müller and Gross 2004], where it was given the name *stiffness warping*. We will use this name to refer to the method of using linear elasticity in a rotated reference frame, reserving the term 'co-rotational' to refer to the nonlinear model discussed previously. Note that this is contrary to the use in engineering literature, where co-rotational methods are purely used as corrections for linear elasticity and the nonlinear energy is not, to our knowledge, used at all.

To understand the method of stiffness warping, let's return to the St. Venant-Kirchoff material model discussed previously, which uses the Green Lagrange strain tensor:

$$\mathbf{E}_{\mathrm{GL}} = \frac{1}{2}(\mathbf{F}^T\mathbf{F} - \mathbb{I}). \tag{5.15}$$

Recall that

$$\mathbf{F} = \frac{\partial \mathbf{x}}{\partial \mathbf{X}} = \frac{\partial \mathbf{u}}{\partial \mathbf{X}} + \mathbb{I}, \tag{5.16}$$

where $\mathbf{u} = \mathbf{x} - \mathbf{X}$. This lets us write $\mathbf{E}_{\text{GL}}$ as:

$$\mathbf{E}_{\text{GL}} = \frac{1}{2}\left(\frac{\partial \mathbf{u}}{\partial \mathbf{X}}^T \frac{\partial \mathbf{u}}{\partial \mathbf{X}} + \frac{\partial \mathbf{u}}{\partial \mathbf{X}}^T + \frac{\partial \mathbf{u}}{\partial \mathbf{X}}\right). \tag{5.17}$$

Note that the nonlinearity in this tensor comes from the first term. If the deformation is small, we can neglect this term as second-order. This gives us the infinitesimal strain tensor, which we call $\boldsymbol{\varepsilon}$:

$$\boldsymbol{\varepsilon} = \frac{1}{2}\left(\frac{\partial \mathbf{u}}{\partial \mathbf{X}}^T + \frac{\partial \mathbf{u}}{\partial \mathbf{X}}\right). \tag{5.18}$$

We can rewrite using the familiar $\mathbf{F}$:

$$\begin{aligned} \boldsymbol{\varepsilon} &= \frac{1}{2}\left(\mathbf{F}^T + \mathbf{F}\right) - \mathbb{I}, \\ &= \mathbf{F}_{\text{s}} - \mathbb{I}, \end{aligned} \tag{5.19}$$

where $\mathbf{F}_{\text{s}}$ is the symmetric part of the deformation gradient. We can go on to define materials based on this strain in the usual manner. Energies based on the square of this measure will be quadratic, and thus the forces will be linear and the associated $\mathbf{K}$ will be constant. Unfortunately, for reasons discussed ad nauseam above, using infinitesimal strain for general dynamic analysis is a poor idea due to its lack of objectivity. In stiffness warping, we overcome this by using the co-rotated deformation gradient $\tilde{\mathbf{F}} = \mathbf{R}^T\mathbf{F}$:

$$\tilde{\boldsymbol{\varepsilon}} = \frac{1}{2}\left(\tilde{\mathbf{F}} + \tilde{\mathbf{F}}^T\right) - \mathbb{I}, \tag{5.20}$$

where $\mathbf{R}$ is calculated using any of the methods discussed in chapter 3. Then when calculating the nodal forces, we rotate back into the global reference frame:

$$\tilde{\mathbf{f}} = \mathbf{R}\mathbf{f}(\mathbf{x}) \tag{5.21}$$

Following the discretization process, we arrive at expressions for the elemental blocks in $\mathbf{K}(\mathbf{x})$ that are simply rotated versions of the same blocks in the linear elastic matrix. This is convenient because it removes the need to compute these blocks every frame, which can be the most expensive part of the simulation in a truly nonlinear solver. The set of linear equations we have to solve still changes each frame, but the cost of forming the stiffness matrix is minimized.

A reasonable question to ask is: What is the difference between this method and the nonlinear co-rotational method we discussed earlier? The primary difference is that we neglect the dependence of $\mathbf{R}$ on $\mathbf{x}$, which means stiffness warping

is only suitable for the small-strain, large displacement regime. A second important difference is that the strain $\mathbf{F}-\mathbf{R}$ lives in the global reference frame, whereas the strain $\mathbf{R}^T(\mathbf{F}-\mathbf{R}) = \mathbf{R}^T\mathbf{F} - \mathbb{I}$ lives in a notional un-rotated frame. This can make plastic deformation tricky when using the global form.

Choice of Solver

Obviously, when the core of our simulation is a large positive definite linear system, the speed and efficiency of our method as a whole is strongly dependent on the linear solver we use. We will quickly survey the methods available to us, deferring a discussion of the specifics of the solvers we use to the next chapter. The most popular method by far for problems of this type is that of Preconditioned Conjugate Gradients (PCG), an iterative process that exploits the positive definiteness of $\mathbf{A}$. Most papers that aim for good performance use some variant of PCG [Parker and O'Brien 2009]; [Müller and Gross 2004]; [McAdams et al. 2011]. Generally a diagonal preconditioner is used; despite reports that more complex partial Cholesky preconditioners can be beneficial there is not very much discussion of preconditioning strategies in the literature.

A competing approach that comes with strong recommendations in the computational science community is the sparse Cholesky factorisation; though complicated to parallelise this direct approach has the advantage of a constant runtime with no dependency on the condition number of $\mathbf{A}$. Hecht et al. [2012] used rank-updates of a Cholesky factorisation to only update the stiffness matrix when the element rotations change substantially, making their method a kind of quasi-Newton approach.

Yet another approach applies the multigrid method to elasticity simulation. McAdams et al. [2011] constructed a hierarchy of voxel grids and used a damped Jacobi smoother at each level. While the performance of this method per DoF was reportedly very good, [Parker and O'Brien 2009] and our own experiments suggest multigrid is not particularly useful when the simulation mesh is coarse. Because we emphasize solving small systems extremely fast, multigrid is comparatively unattractive for our purposes.

5.3 Position Based Finite Elements

The ubiquity of Position Based Dynamics (Section 1.2.1) in real-time cloth simulation raises the question of whether soft body simulation can benefit from the same technique. It turns out the application of PBD to Finite Element simulation is fairly straightforward and has some useful properties compared to implicit simulation. The basic approach is to take discretized finite element strain expressions and solve them sequentially as constraints in a Projected Gauss-Seidel fashion, exactly as edge constraints are solved in cloth simulation. The problem in force-based simulation of *minimizing* a large nonlinear system is replaced with the problem of *solving* a small nonlinear system. We can solve a given constraint equation $C_i(\mathbf{x})$ by linearising about an iterate $\mathbf{x}_k$:

$$\mathbf{\Delta x} = -\mathbf{M}^{-1}\frac{\partial C}{\partial \mathbf{x}}(\mathbf{x}_k)\frac{C(\mathbf{x}_k)}{\|\mathbf{M}^{-1/2}\frac{\partial C}{\partial \mathbf{x}}(\mathbf{x}_k)\|^2}. \tag{5.22}$$

The original PBD paper presented this as an empirical method, but it was shown in [Bouaziz et al. 2014] that this is equivalent to applying a step of Sequential Quadratic Programming (SQP). To solve FEM energies in this framework, we can simply reinterpret our discrete energies as constraint equations. Then the constraint gradient $\frac{\partial C}{\partial \mathbf{x}}$ is just the local force vector for the particles involved in the constraint. In this way we can construct a system with the familiar qualities of PBD as well as the advantages of our favorite FEM energies, such as the robust co-rotational strain. See Algorithm 2 for a sketch of how this method is used in practice.

A unique advantage compared to force-based methods is that our constraints do not have to be valid energies; for instance the *exact* volume constraint term

$$\det(\mathbf{F}) = 1 \tag{5.23}$$

has the same desirable properties as the co-rotational model when solved as a constraint; namely that it works to un-invert the element even when inverted. This is a clear advantage over volume penalty terms like $\ln(\mathbf{F}) - 1$ or $\det^2(\mathbf{F}) - 1$ which are either invalid or incorrect under inversion. While this term can be included in a force-based solver using a Lagrange multiplier approach, the constraint-based nature of PBFE means we can treat it the same way as more

Algorithm 2 Position Based Finite Elements update loop

for all particles i **do**
  $\mathbf{p}^i \leftarrow \mathbf{p}^i + \Delta t \mathbf{v}^i$
  $\mathbf{v}^i \leftarrow \mathbf{v}^i + \Delta t \mathbf{f}^i_{\text{ext}}$
end for
DetectCollisions($\mathbf{p}^i$)
for all elements i **do**
  UpdateRotation($\mathbf{R}^i$)
end for
for $i = 1$ to max iterations **do**
  for all elements j **do**
    SolveCollisions()
    $\mathbf{F}^e \leftarrow \frac{1}{8}\mathbf{J}^{e-1} \sum_{k=1}^{8} \mathbf{x}^k \otimes \boldsymbol{\xi}^k$
    Compute $C_{\text{CR}}, C_{\text{HG}}, C_{\text{V}}, \mathbf{f}_{\text{CR}}, \mathbf{f}_{\text{HG}}, \mathbf{f}_{\text{V}}$
    SolveConstraint($C_{\text{CR}}, \mathbf{f}_{\text{CR}}, k_{\text{s}}$)
    SolveConstraint($C_{\text{HG}}, \mathbf{f}_{\text{HG}}, k_{\text{hg}}$)
    SolveConstraint($C_{\text{V}}, \mathbf{f}_{\text{V}}, k_{\text{v}}$)
  end for
end for
for all elements i **do**
  DampVelocities()
end for

standard energies. In Figure 5.1, we compare our exact volume constraint with the standard trace-based co-rotational pressure term, showing the accuracy benefit of our approach.

5.3.1 Discussion

On paper, PBD is a fairly poor method; in particular it scales extremely badly when mesh resolutions are increased. The most obvious consequence of this poor scaling is that running the same number of solver iterations on a more fine mesh will cause the observed behaviour to be much less stiff. This means PBD requires hacks like Long Range Attachments [Kim, Chentanez, and Müller-Fischer 2012]

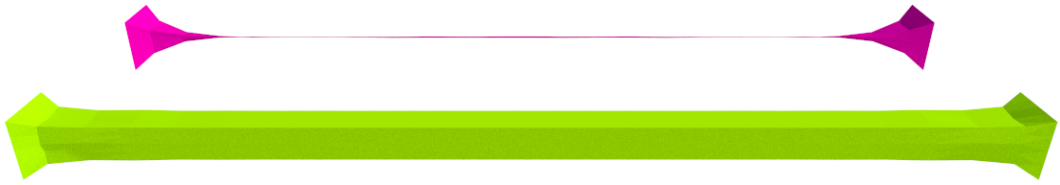

Figure 5.1: Comparison of volume constraints. Meshes using our volume constraint (foreground) and the co-rotational pressure term $\mathrm{tr}^2(\mathbf{R}^T\mathbf{F} - \mathbb{I})$ (background) are stretched to 2.5x their original length. The standard term causes the mesh to lose 90% of its volume; the mesh using our constraint gains 1.2%. Such strains are not uncommon in simulation of flesh deformation, so the benefits of an accurate volume constraint are clear in that case.

to work acceptably even with fairly coarse meshes. The local nature of the non-linear Gauss-Seidel process means that wave propagation inside the simulation mesh is bound by the frame rate and number of iterations. Strong dependence of observed behaviour on these two factors is also a black mark against PBD. It is impossible to dial in the properties of a particular material and get realistic behaviour, as one might expect when using an accurate simulation method. *However*, the local nature of PBD is also its greatest strength because it allows us to constrain particles in ways that are not possible in global, matrix-based methods without a great deal of sacrifice. In fact, one can easily recognise the similarity between PBD and the projected Gauss-Seidel (PGS) method used for rigid body simulation, discussed in the next chapter. PGS has become popular in rigid body simulation because of its stability (which PBD shares, see Figure 5.2) and ability to handle equalities and inequalities alike. One tends not to notice this sort of robustness until one tries another method, whereupon its absence becomes very painful. PBD is thus the best method that we have right now for general simulation of soft bodies in videogames; other methods can be very competitive in certain arenas but the robustness requirements imposed by user interaction mean that we are limited in how we can deploy them.

In terms of performance, it is hard to compare PBD with global methods because in PBD the number of solver iterations is part of the *behaviour* of the material we are simulating, whereas global methods must generally converge completely to not have objectionable errors. The fully-converged state of PBD is total

rigidity, so we cannot directly compare these methods. Thus in our performance comparison (Figure 5.3) we compare PBFE only against itself. We can, however offer the following qualitative comparison: For hexahedral element counts around or below 1000, PBFE converges with adequate speed in our tests to simulate convincing flesh. At this detail level, the robustness advantages discussed above mean that our method is a good choice for realtime simulation. At higher resolutions, the poor convergence of PBD begins to make it difficult to obtain realistic behaviour and global methods are a superior choice.

5.3.2 Use in Games

We have implemented the PBFE system as the back-end for a real-time soft body deformation system. We take a lattice deformation approach, where visual objects are coarsely voxelised and the generated voxels used as hexahedral elements in the simulation mesh. We transfer the deformation of the simulation mesh to the visual mesh by linear blend skinning; we use the deformation gradient at the center of each element as a 'bone' and blend these transforms for each visual mesh vertex, weighted according to a Gaussian distribution. This allows us to have smooth deformations of the render mesh without simulating high-order elements. Artists using this system have found it to be useful for simulating simple dynamic deformations of inanimate objects - for instance, a crashing vehicle, or a metal cabinet reacting to an explosion. While the slow convergence of the method limits the detail that can be achieved using PBFE, users have found our system to offfer robust behaviour with predictable performance; as long as the constraints imposed on the system are compatible the simulation will not explode and the run-time cost of the simulation will not change substantially.

Algorithm 3 PBD constraint solving for hexahedra

function SolveConstraint($C, \mathbf{f}, k$)

$s \leftarrow -kC / \sum_{i=1}^{8} w^i |\mathbf{f}^i|^2$

$\mathbf{p}^i \leftarrow \mathbf{p}^i + s w^i \mathbf{f}^i$

$\mathbf{v}^i \leftarrow \mathbf{v}^i + s w^i \mathbf{f}^i / \Delta t$

end function

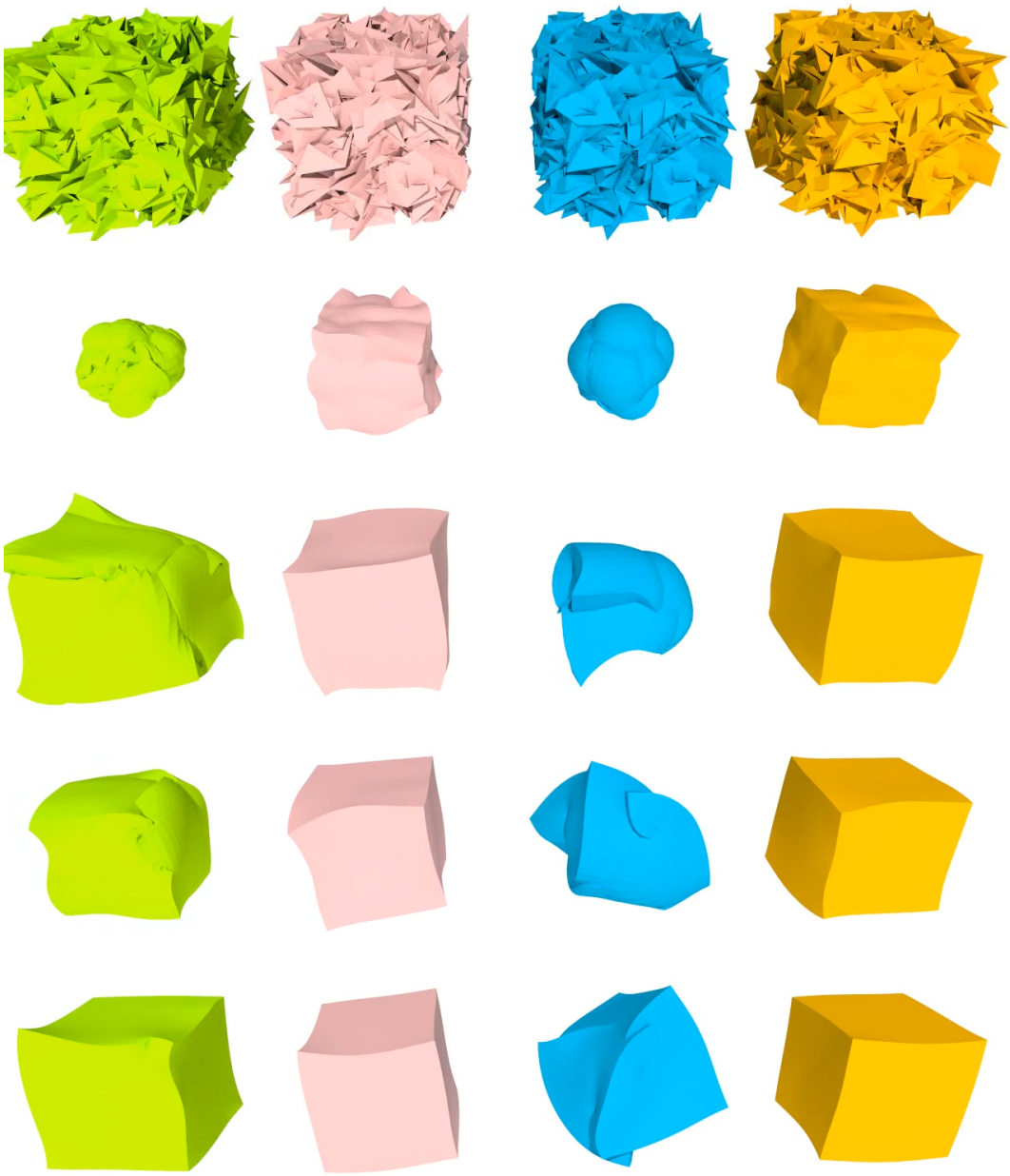

Figure 5.2: A 32x32x32 cube of hexahedral elements has its vertices randomly placed inside a unit cube in order to test recovery from an extreme state. Many of the elements are severely deformed and inverted. From top to bottom are snapshots of the resulting motion taken every 0.5s. From left to right, we use our PBFE method with our spin-based rotation warm started, the same method cold started, the fast SVD of McAdams et al. [2011], and symmetric diagonalization.

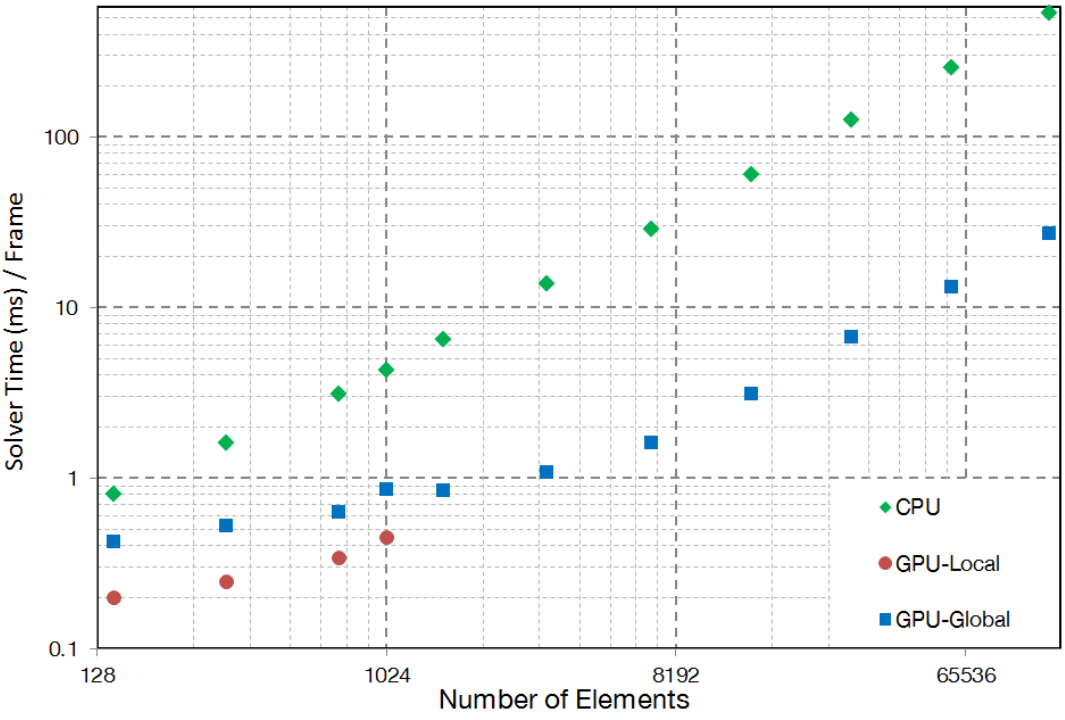

Figure 5.3: Comparison of solver performance (omitting collision detection) on CPU and GPU. Each solver performs 10 iterations. Unsurprisingly the performance scales linearly on both processor types. The GPU only operates efficiently once the problem size becomes large enough.

5.4 Projective Dynamics

Stiffness warping allowed us to minimize the work required to produce the stiffness matrix, but this matrix still changed from frame to frame. However, by changing the problem somewhat we can formulate a method that exhibits the appropriate nonlinear material behaviour but has a *constant* stiffness matrix. This method was developed by Bouaziz et al. [2014], but we will re-state it here for convenience. The central idea is that instead of minimizing a strain energy directly, we cast each element as a constraint, and then minimize the distance between the solution of each constraint. For instance, consider the co-rotational strain constraint:

$$\mathbf{F} = \mathbf{R} \tag{5.24}$$

This condition essentially says that the element should be in its *rotated reference* configuration. Both $\mathbf{F}$ and $\mathbf{R}$ depend on $\mathbf{x}$, but if we take $\mathbf{R}$ to be fixed then the solution to the constraint is trivial. Similarly, if we take $\mathbf{F}$ to be fixed then we can compute $\mathbf{R}$ by any of the methods discussed previously. Projective Dynamics works by alternating between solving the nonlinear part of each constraint locally, then solving the linear parts globally. By pushing all nonlinearity into the local solves, we can ensure the global stiffness matrix is constant. The original paper enforces this by writing the problem in a specific way, which we will state slightly differently for clarity:

$$\mathbf{x}^{t+\Delta t} = \min_{\mathbf{x}^{t+\Delta t}} \left(\frac{1}{2}\Delta t^{-2} \| \mathbf{M}^{1/2} \left(\mathbf{x}^{t+\Delta t} - \mathbf{x}^{*} \right) \|_2^2 + \sum_{i=1}^{n} \frac{k_i}{2} \| \mathbf{D}_i \mathbf{S}_i \mathbf{x}^{t+\Delta t} - \mathbf{P}_i \|_F^2 + \delta_{C_i}(\mathbf{P}_i) \right), \tag{5.25}$$

This equation demands some explanation. Note the three terms inside the minimization. The first term is the familiar work potential from (5.10). The second and third terms represent the energy of each constraint. The global optimization is represented by:

$$\sum_{i=1}^{n} \frac{s_i}{2} \| \mathbf{D}_i \mathbf{S}_i \mathbf{x}^{t+\Delta t} - \mathbf{P}_i \|_F^2, \tag{5.26}$$

where s_i is the stiffness of each constraint C_i, $\mathbf{D}_i$ and $\mathbf{S}_i$ are matrices associated with the constraint that we will elaborate on later, and $\mathbf{P}_i$ is the auxiliary variable for the constraint. This is the solution of the local step, which is represented by

$\delta_{C_i}(\mathbf{P}_i)$. This rather uncouth notation evaluates to $+\infty$ when $\mathbf{P}_i$ is not the solution to the local step, and 0 when it is.

Global Step

The optimality condition for this minimization problem taking $\mathbf{P}_i$ to be constant defines the global step. Although the global system is linear, we still introduce the iteration number k because the nonlinearity is present in the local step. The global linear system is:

$$\left(\mathbf{M}\Delta t^{-2} + \sum_{i=1}^{n} s_i \mathbf{D}_i^T \mathbf{D}_i \mathbf{S}_i\right) \mathbf{x}_{k+1}^{t+\Delta t} = \mathbf{M}\Delta t^{-2}\mathbf{x}^* + \sum_{i=1}^{n} s_i \mathbf{S}_i^T \mathbf{D}_i^T \mathbf{P}_i. \tag{5.27}$$

The method of projective dynamics alternates between solving independent local steps to find $\mathbf{P}_i$ for each constraint, and solving the above global system to obtain a new candidate $\mathbf{x}^{t+\Delta t}$. Note that changing $\mathbf{P}_i$ only changes the right hand side vector; if the particle mass, constraint topology and constraint stiffnesses are unchanged then the matrix on the left hand side is *constant* and can be pre-factorised. We need only perform a double back-substitution of the Cholesky factorisation at every global step.

Local Step

We solve the right hand side of each constraint locally. Taking the positions to be fixed, we attempt to find auxiliary variables $\mathbf{P}_i$ that best match the current configuration of the constraint. Because we have concentrated all nonlinearity into the right hand side, this is at worst a small nonlinear root finding problem. Many constraints, such as the corotational constraint $\mathbf{F} = \mathbf{R}$, have closed-form solutions. Those that do not can be linearised and solved as a small Newton system.

5.4.1 Examples

So what are the mystery matrices $\mathbf{S}_i$ and $\mathbf{D}_i$? Essentially, these matrices make sure we compose our constraint in a linear way. $\mathbf{S}_i$ is the constraint *selection* matrix, which reduces the global vector $\mathbf{x}_k$ to only the nodes involved in the constraint. $\mathbf{D}_i$ we call the *mangling* matrix, which reconfigures the constraint node vector to the geometric quantity involved in the constraint.

Finite Elements

For instance, consider the familiar co-rotational constraint $\mathbf{F} = \mathbf{R}$ discretized on linear tetrahedra. We can write it as:

$$\mathbf{D}_s\mathbf{D}_m^{-1} = \mathbf{R}, \tag{5.28}$$

where $\mathbf{D}_s = [\mathbf{x}_1 - \mathbf{x}_4, \mathbf{x}_2 - \mathbf{x}_4, \mathbf{x}_3 - \mathbf{x}_4]$ and so on as before. Let's isolate only the geometric information on the left hand side:

$$\mathbf{D}_s = \mathbf{R}\mathbf{D}_m. \tag{5.29}$$

Computing the right hand side can be done through SVD or our spin-based method, as described below. Our task is to find the selection and mangling matrices that bring $\mathbf{x}_k$ to $\mathbf{D}_s$. First of all we should note that directly representing $\mathbf{D}_s$ as a matrix with this process requires a packed system vector; using the stack convention we have to unwrap $\mathbf{D}_s$ to a vector in a similar manner to Voigt notation. We'll use the pack convention here for simplicity. The first task is to reduce $\mathbf{x}_k$ to only the nodes involved in the tetrahedron using the element selection matrix. This can be done with the rectangular matrix $\mathbf{S}_{i_{kl}} \in \mathbb{M}^{4\times n}$, whose entries are 1 if the lth node in the global numbering is the kth node in the ith tetrahedron, and 0 otherwise. Now that the nodes are in a canonical form, the mangling matrix is:

$$\mathbf{D}_{\text{CR}} = \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & -1 \end{bmatrix}. \tag{5.30}$$

It's easy to verify that $\mathbf{D}_{si} = \mathbf{D}_{\text{CR}}\mathbf{S}_i\mathbf{x}_k$.

Mass-Spring

Mass-Spring systems have persisted through the history of computer graphics and are the basis of many successful simulations. It's easy to fit mass-spring simulation into the projective dynamics framework; the constraint is $\mathbf{x}_1 - \mathbf{x}_2 = \mathbf{R}(\mathbf{X}_1 - \mathbf{X}_2)$. The local step is trivial: we simply normalize $\mathbf{x}_1 - \mathbf{x}_2$. As for the global step, the selection matrix $\mathbf{S}_{i_{kl}} \in \mathbb{M}^{2\times n}$ has 1s where the lth node in the global ordering is the kth node in the ith spring, and the mangling matrix is:

$$\mathbf{D}_{\text{MS}} = [1, -1]. \tag{5.31}$$

It's interesting to note that the global matrix arising from a spring network is simply a graph Laplacian of the network weighted by stiffness and length.

Shape Matching

We can also include shape matching in the projective dynamics framework. This is useful because the shape matching energy is extremely simple to work with and is not susceptible to, for instance, hourglass anomalies. Our strategy for adding shape matching to this framework is to use the selection and mangling matrices to compute the offsets from the center of mass of the shape matching group. The constraint equation for a group of m nodes is:

$$\begin{bmatrix} \mathbf{x}_1 - \mathbf{x}_c \\ \mathbf{x}_2 - \mathbf{x}_c \\ \cdots \\ \mathbf{x}_m - \mathbf{x}_c \end{bmatrix} = \mathbf{R} \begin{bmatrix} \mathbf{X}_1 - \mathbf{X}_c \\ \mathbf{X}_2 - \mathbf{X}_c \\ \cdots \\ \mathbf{X}_m - \mathbf{X}_c \end{bmatrix}, \tag{5.32}$$

where $\mathbf{x}_c$ and $\mathbf{X}_c$ are the barycenters of the current and initial shape matching group, i.e $\mathbf{x}_c = \frac{1}{m}\sum_{i=1}^{m} \mathbf{x}_i$. The local step as usual requires solving the nearest-rotation problem, which can be done by any of the methods discussed. The global step requires forming the left hand side of (5.32) from $\mathbf{x}_k$. The selection matrix for the group is straightforward, and the mangling matrix $\mathbf{D}_{\text{SM}} \in \mathbb{M}^{m\times m}$ is $\mathbf{D}_{\text{SM}} = \mathbb{I} - \frac{1}{m}\mathbf{1}$ where $\mathbf{1}$ is the $m \times m$ matrix of 1s. It turns out that computing the barycenter in this way is enough to obtain the benefits of projective dynamics, making this a viable alternative to regular shape matching using PBD.

5.4.2 Advantages

So what are the advantages of projective dynamics? Comparing to Newton's method the answer is straightforward: We obtain the benefits of nonlinear simulation with a constant stiffness matrix, which gives a substantial *per iteration* performance improvement. However, the method is only first-order; at no point do we use the derivatives of our constraint functions. Projective dynamics thus only exhibits linear convergence to the solution of the minimization problem, inferior to the quadratic convergence of Newton's method. If we were interested in the fully converged solution, then, this method could not be recommended.

Fortunately, as game developers we have no appetite for actual convergence, and care more about the artifacts caused by operating far from the converged state. Here projective dynamics shows remarkable improvement: solving the global system *completely* removes the stretching artifacts associated with PBD with only a single iteration. The artifacts induced by using too few iterations in projective dynamics are much more subtle, with their severity being determined by what we choose to solve in the local iterations. Thus far this has mostly been rotation extraction of one kind or another, and this predictably results in inaccurate treatment of rotational motion when only a small number of iterations are used. However, the fact that the static equilibrium is so much closer to correct is a huge advantage over PBD.

5.4.3 Limitations

We mentioned previously that the matrix associated with the global problem does not change as long as constraints are not added to or removed from the system. Unfortunately, in a general dynamic simulation we *are* likely to want to change the constraint set; we will at least want to add and remove contact constraints to prevent objects from interpenetrating. A simple contact constraint in the PD framework is $\mathbf{x}_i = \mathbf{p}_i$, which binds the node $\mathbf{x}_i$ to the target position $\mathbf{p}_i$, which we can manipulate in the local step. This single-particle constraint only results in a single extra diagonal entry in the global matrix. However, in the general case this will change *every* entry in the Cholesky factorisation of that matrix. Dealing with this problem is the most important consideration in an implementation of PD, and we will discuss two approaches to it below.

Another significant weakness of this method, unaddressed in the original paper, is that it causes a great deal of damping of rotational motion. While a single iteration of PD is enough to solve the linear parts of the strain energy, rotations are essentially seen as fixed at every iteration. When using a low number of iterations, this can manifest as obvious 'rotation locking' that is particularly objectionable when simulating cloth. Although we can alleviate this locking by doing more iterations, this seriously erodes the attraction of the method. PD also converges very slowly in this sense; we need to do a great deal of iterations

to reduce the rotation locking effect to an acceptable level. The same problem manifests when using some of the techniques for inequality constraints suggested in the paper; while we *can*, for instance, modify the contact constraint targets every iteration to enforce something akin to inequality, this suffers from exactly the same problem of damping and slow convergence as the rotations. PD thus does not work quite as well as advertised in this sense.

5.4.4 Implementation

The basic implementation of PD is straightforward and uncontroversial; we will describe it briefly and move on to the more interesting question of dynamic constraints. The local step essentially the same as what we have already described for the various PBD methods.

Global Solve

The global solve requires us to compute the Cholesky factorisation of the associated matrix $\mathbf{A} = \mathbf{L}\mathbf{D}\mathbf{L}^T$. This does not need to be particularly efficient, since it can be done as a pre-process. The meat of the algorithm involves solving right hand side vectors $\mathbf{b}$ using this decomposition, or in other words computing $\mathbf{x}_{k+1} = \mathbf{L}^{-T}\mathbf{D}^{-1}\mathbf{L}^{-1}\mathbf{b}$. Because we have to do this at least once per iteration, it needs to be as fast as possible. Unfortunately the two *sparse* triangular solves involved in this operation are fundamentally serial, which gives us limited options for parallelisation. An alternative is to simply calculate the full inverse of $\mathbf{A}$, which although symmetric is likely to be much more dense than the Cholesky factor of $\mathbf{A}$. What we lose in memory size we can potentially gain in speed, however, since it is much easier to parallelise multiplication by $\mathbf{A}^{-1}$ than triangular solves. This suggests that for GPU simulation, at least, dense linear algebra may be preferable. We have not yet tested that possibility, however.

Dynamic Constraints with Rank-One Updates

The question of handling dynamic constraints is critically important to the speed of a PD implementation. The original paper [Bouaziz et al. 2014] suggested rank-one updates to the Cholesky factor of $\mathbf{A}$; in short, we can efficiently calculate the Cholesky factor of a perturbation $\mathbf{A} + \mathbf{u}\mathbf{v}^T$ where $\mathbf{u}$ and $\mathbf{v}$ are vectors. The

perturbation $\mathbf{u}\mathbf{v}^T$ is an outer-product matrix with a rank of one; hence the name *rank-one update*. Because we are mostly interested in adding and removing single entries from the diagonal of $\mathbf{A}$, we can say $\mathbf{u} = \mathbf{v}$. Rank-one updates can also be applied to the full inverse matrix, using the *Sherman-Morrison formula*. In general, every entry of the Cholesky factor or inverse will change when we change one diagonal entry, so the cost of adding or removing one contact is proportional to the size of the updated matrix. Adding entries to the diagonal is at least guaranteed to not change the sparsity pattern of the resulting Cholesky factor, which means the memory size of the simulation will not change when we make updates of this type. However, one can imagine situations in game simulation where very large numbers of contacts might be added or removed in a single frame. This makes the strategy of using rank-updates problematic, because the huge spikes in runtime cost in frames which include heavy collision are highly disagreeable from an optimisation point of view. It is desirable, therefore, to find a method which does not cause cost spikes of this kind.

5.5 Dynamic Constraints with PCG

An alternative to modifying the decomposition of $\mathbf{A}$ is to treat $\mathbf{A}$ as a preconditioner for the perturbed system. This relies on the the idea that the inverse of $\mathbf{A}$ is *similar* to the inverse of the perturbed matrix $\tilde{\mathbf{A}}$; perhaps close enough that it could be an effective *preconditioner* for the perturbed system $\tilde{\mathbf{A}}\mathbf{x}_{k+1} = \mathbf{b}$. In the language of iterative methods, a preconditioner is any method that approximately solves the system and allows the iterative method to converge much faster. This idea leads directly on to a potential algorithm: every frame instead of simply solving the global step with our pre-computed decomposition of $\mathbf{A}$, we do conjugate gradient (CG) iterations with the factor of $\mathbf{A}$ as a preconditioner. If $\tilde{\mathbf{A}} = \mathbf{A}$ then no iterations will be required. If we change $\tilde{\mathbf{A}}$ then we can hope that we will only have to use a small number of CG iterations, since the nearly-exact preconditioner will do most of the work for us. So is this a viable alternative to rank-updates? The answer depends on how many CG iterations are required. A CG iteration preconditioned by the Cholesky factor of $\mathbf{A}$ requires two triangular solves, a multiplication by $\tilde{\mathbf{A}}$ and a vector norm calculation. Thus a single

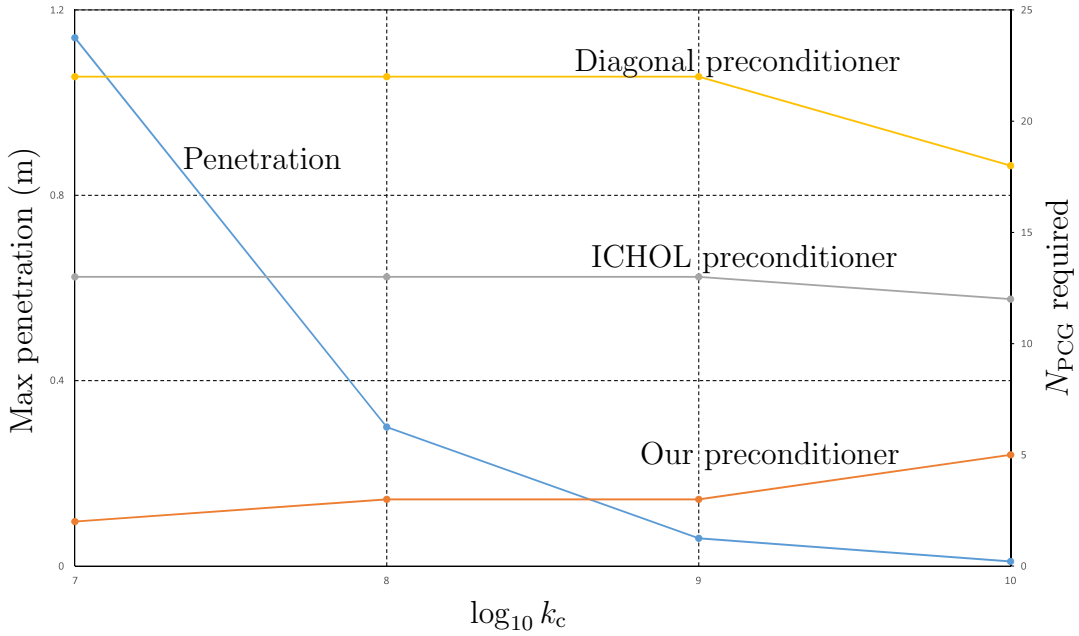

Figure 5.4: Maximum penetration and required iterations for various methods against contact strength for a $1m^3$ block of metal hitting the ground at 10m/s.

iteration is substantially more expensive than the simple substitutions required when $\tilde{\mathbf{A}} = \mathbf{A}$. While a larger fixed cost is preferable to a small cost with large intermittent spikes, the viability of this technique depends on exactly how much larger this fixed cost is. To get an idea of how this method performs, we did some experiments in MATLAB. While raw performance numbers from such an environment are meaningless, the number of iterations required will be the same no matter the implementation environment and give good insight into the viability of this method.

Comparisons

In our experiment we emulated a 'worst-case' game scenario guaranteed to stress the contact model. A $1m^3$ block of hexahedral elements using a shape-matching material model with density and stiffness of steel ($\rho = 8050\ \mathrm{kg\ m^{-3}}$, $E = 200\mathrm{GPa}$) impacts the ground at $10\mathrm{ms}^{-1}$. A contact constraint of the type discussed previously is generated between all nodes at the bottom of the cube and the ground. Using various values for the contact stiffness k_c, we record the maximum penetration of any node in the block into the ground. We solve the system using unpreconditioned conjugate gradients (line N_{CG}), our full Cholesky precondi-

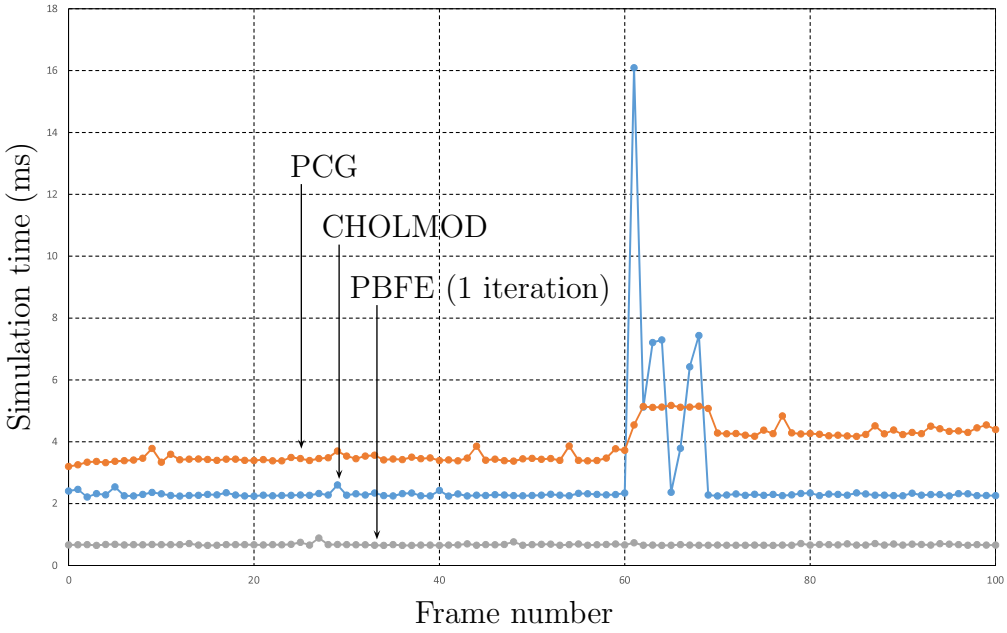

Figure 5.5: Total simulation time per frame for different methods of solving the contact-perturbed system $\tilde{\mathbf{A}}$.

tioned method (line $N_{\text{PCG-A}}$), and an incomplete Cholesky preconditioner (line $N_{\text{PCG-I}}$). The aim was to find how many CG iterations are necessary to solve the system when the contact stiffness is large enough to reduce penetration to an acceptable level, and whether our full Cholesky preconditioner was particularly effective. The results are presented in Figure 5.4. This graph shows that increasing the relative strength of the dynamic contacts k_c will, as one would expect, decrease the maximum penetration experienced by the dropped block (Note the penetration is not eliminated completely, as a result of regularizing the contact constraint). Increasing the strength of the dynamic components of the system increases the difference between the static and dynamic systems, which one would imagine might degrade the effectiveness of our preconditioner. However, the graph shows that even for very large diagonal terms our preconditioner is still vastly better than other common preconditioning approaches[2].

We also ran a full simulation (in C++, with reasonable efforts taken towards optimization) comparing the rank-update approach using CHOLMOD with our

[2]Note that the Incomplete Cholesky and diagonal preconditioners become more effective as k_c increases. This makes sense because the matrix becomes more diagonally dominant as a result, which naturally makes these preconditioners more effective.

PCG-based approach. A 1m×1m×1m block of material with 2197 DoFs and 8640 tetrahedral elements is dropped from a height of 5m, impacting the ground on frame 61. We recorded the total simulation time for each frame. From the results in Figure 5.5 we can see that the CHOLMOD-based approach suffers from a severe 16ms spike when the contacts are added on the ground impact frame. Although our PCG-based approach costs more in the steady state (thanks mostly to inefficiencies in the Eigen library), the large spike is completely removed. Instead the solver takes only two extra iterations while the collision is taking place, and one extra when the object is resting on the ground. This suggests that for simulations including a heavy load of temporary contact constraints, our PCG approach is strongly preferable to CHOLMOD. We expect these situations to arise very commonly in game simulation, so it is fair to say that our approach is much more well-suited to this use case.

5.6 Concluding Remarks

In this chapter, we discussed various methods for simulating soft bodies. We introduced a new position-based finite element method that is more suitable for use in games, as well as making some modifications to the fast global Projective Dynamics method for the same purpose. Both of these methods have their own set of advantages and disadvantages, and it remains to be seen if it is possible to combine the global convergence properties of the matrix-based methods with the remarkable convenience of PBD. In the meantime, there are situations in games in which one might want to use either of the methods we have introduced.

Chapter 6

Position Based Rigid Body Dynamics

The next chapters are about simulating constrained rigid body dynamics. This is in many ways a more difficult problem than soft body simulation. First of all, the inclusion of rotations makes the calculus more complicated. Second, because rigid body simulations generally affect gameplay we require a much higher level of correctness. The final and most vexing reason is that we are just not very good at solving problems of this type. In the world of soft body simulation we were always able to reduce the problem to a linear system of some kind, even if it meant making compromises like penalty-method contacts. In rigid body simulation this is just not good enough. Human beings are quite good at noticing inaccurate simulation of rigid bodies, and to reliably simulate colliding rigid bodies we must model contact constraints as inequalities. This means we must solve not a system of equations but a *complementarity problem*. While there are extremely efficient methods available to systems of (linear and nonlinear) equations, equivalent methods do not exist for complementarity problems. As a result we are forced to turn to Projected Gauss-Seidel (PGS) iteration, which is more or less the only reliable tool for solving problems of this kind.

Compounding the above is the nature of the academic literature on the subject, which more or less terminates in the late 1990s with the independent formulation of the Stewart and Trinkle [1996] and Anitescu and Potra [2001] models. While these models are so far the only principled and provable model of contact dynamics, they cannot yield real-time performance (due to their reliance on Lemke's algorithm) and bear little relationship to how rigid body dynamics is actually implemented in practice. Most material on this topic is in un-peer-reviewed formats - industry conference presentations, internet forum discussions and source

code. The few recent papers on real time rigid body dynamics generally avoid top-down formalism and concentrate on implementation [Tonge, Benevolenski, and Voroshilov 2012], and the recent survey paper [Bender et al. 2012] presents a confused picture of the field; systems are presented in isolation without a taxonomy to link them together. While I would like to present an understandable and approachable summary of rigid body simulation as it is actually practiced, reconciling all the various methods and codes available could probably form the basis of a PhD on its own. Instead, this chapter will focus on the ways that EA's internal physics engine *differs* from other methods that have been presented. We discuss some interesting additions that we believe are novel; specifically, the use of a fully position-based solver, robust predictive contacts, and cheap but accurate high-fidelity joints.

6.1 EAPhysics

The EAPhysics engine works somewhat differently to other physics engines that have been described in the literature, and its workings are harder to justify. The main difference is that EAPhysics works at the *position level* of the constrained dynamics problem, as opposed to the much more standard *velocity level*. Simulations that take the latter approach seek to satisfy the time derivatives of the relevant constraint equations rather than the equations themselves. This is particularly useful for simulations involving rotations, since solving at the velocity level allows our angular constraints to be linear. However, the great disadvantage of a velocity-level solver is that positional errors in our constraints can accumulate due to floating-point error or a lack of convergence. Velocity-level methods require *stabilization* to prevent these errors from building up. Stabilization measures like Baumgarte's method require a certain amount of tuning and do not work well in all circumstances. In particular, tightly coupled systems like ragdolls tend to develop a great deal of error that leads to unrealistic behaviour in more violent situations. Our position-level method does not require *any* stabilization, and (anecdotally) deals with tightly coupled systems more effectively than other physics engines we have compared it to. While the method is, as mentioned, hard to justify formally, it is effective and worth covering as we move

towards discussing our work on curve constraints.

6.2 Rigid Body Representation

From a mathematical perspective, the only difference between a discretized finite element node and a rigid body i is that the latter has an orientation $\mathbf{q}_i \in \mathrm{SO}(3)$ and inertia $\mathbf{I}_i \in \mathbb{R}^{3\times3}$ as well as a position $\mathbf{p}_i$ and mass m_i. SO(3) is the 3D *rotation group* which we can represent in a wide variety of ways; to avoid any discussion of group theory we will simply state that the dominant representation of orientation in computer graphics is the *unit quaternion*, hence the label $\mathbf{q}$. Note that the *unit* part is usually left off, and whenever we refer to quaternions it can safely be assumed they are normalised.

Conversely, we represent small rotations using *Euler vectors* $\boldsymbol{\omega}_i \in \mathbb{R}^3$. Not to be confused with a vector of Euler angles, which are sequenced rotations about pre-agreed axes, an Euler vector is simply the rotation axis multiplied by the rotation angle. While this representation is usually introduced as the angular velocity vector, we eschew velocities in favor of a position-only representation. The next section will explain this choice in more detail.

We stack $\mathbf{p}_i$ and $\mathbf{q}_i$ together to obtain the state vector $\mathbf{x}_i \in \mathbb{R}^7$ for each rigid body, and stack the n $\mathbf{x}_i$s together to form the system state vector $\mathbf{x} \in \mathbb{R}^{7n}$. This notation is useful for reducing mathematical complexity later on, and does not imply that we actually store the data in this way.

6.3 Time Discretization

The rigid body problem is already discrete in space, but is continuous in time. To actually simulate anything we have to choose an integration scheme. This is equivalent to discretizing time. We assume a *fixed* timestep h and a sequence of frames $t = 0, 1, ..., m$. Then we define our integration scheme using *central differences*. The *displacement* vector is the amount of state change over one frame:

$$\mathbf{d}^{t+1} = \mathbf{x}^{t+1} - \mathbf{x}^{t}, \tag{6.1}$$

where $\mathbf{d}_i \in \mathbb{R}^6$ is the *velocity displacement* of each body. This equation makes little sense on first glance since $\mathbf{d}_i$ and $\mathbf{x}_i$ are not even the same length. To restore some order we have to treat the minus sign in the above equation as a *generalised difference* operator, or in other words:

$$\begin{aligned} \mathbf{l}_{\mathrm{v}i}^{t+1} &= \mathbf{p}_i^{t+1} - \mathbf{p}_i^t, \\ [0, \mathbf{a}_{\mathrm{v}i}^{t+1}] &= \log((\mathbf{q}_i^{t+1})^{-1}\mathbf{q}_i^t), \end{aligned} \tag{6.2}$$

where $\mathbf{l}_{\mathrm{v}i}$ is the linear[1]- and $\mathbf{a}_{\mathrm{v}i}$ the angular- velocity displacement of the ith body. We can safely drop the zero, leaving us with a state vector of the right length. We can also use the central difference approximation for acceleration, giving us the *acceleration displacement*:

$$\mathbf{s}^{t+1} = \mathbf{d}^{t+1} - \mathbf{d}^t, \tag{6.3}$$

We can safely assume that angular displacements will change slowly enough that the approximation implied by subtracting Euler vectors remains valid, and thus:

$$\begin{aligned} \mathbf{l}_{\mathrm{a}i}^{t+1} &= \mathbf{l}_{\mathrm{v}i}^{t+1} - \mathbf{l}_{\mathrm{v}i}^{t}, \\ \mathbf{a}_{\mathrm{a}i}^{t+1} &= \mathbf{a}_{\mathrm{v}i}^{t+1} - \mathbf{a}_{\mathrm{v}i}^{t}, \end{aligned} \tag{6.4}$$

where $\mathbf{l}_{\mathrm{a}}$ and $\mathbf{a}_{\mathrm{a}}$ are the linear and angular acceleration displacements respectively.

Given this time discretization we can easily obtain our integration scheme by substituting the previous equations into one another and re-arranging:

$$\mathbf{x}^{t+1} = \mathbf{x}^t + \mathbf{d}^t + \mathbf{s}^{t+1}. \tag{6.5}$$

Here we must treat the + sign as a generalised addition operator, i.e.

$$\begin{aligned} \mathbf{p}_i^{t+1} &= \mathbf{p}_i^t + \ \mathbf{l}_{\mathrm{v}i}^{t} + \mathbf{l}_{\mathrm{a}i}^{t+1}, \\ \mathbf{q}_i^{t+1} &= \mathbf{q}_i^t \exp([0, \mathbf{a}_{\mathrm{v}i}^{t} + \mathbf{a}_{\mathrm{a}i}^{t+1}]). \end{aligned} \tag{6.6}$$

Without any external forces or constraints in the system we are more or less done. However, these restrictions would result in a very boring simulation.

6.4 Constraint Models

For believable rigid-body physics worlds we must be able to model *hard* equality and inequality constraints. We specify both types using constraint equations and

[1]It is difficult to avoid this confusing use of the word 'linear' to describe the parts of the rigid body state that are not angular. In all other contexts we use it to describe equations that are not nonlinear.

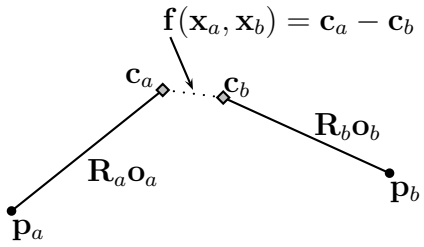

Figure 6.1: A ball-and-socket joint constraint. The constraint states that two points in the local space of bodies a and b should be coincident in world space - $\mathbf{c}_a$ should be equal to $\mathbf{c}_b$.

inequalities. Generally these are simple conditions applied to points in the local spaces of the two bodies involved in the constraint. For instance, a simple ball and socket joint constraint (Figure 6.1) can be modelled by the vector equation (or three scalar equations if you prefer):

$$\mathbf{f}_i(\mathbf{x}_a, \mathbf{x}_b) = \mathbf{c}_a - \mathbf{c}_b = \mathbf{0}, \tag{6.7}$$

where $\mathbf{f}_i$ is the ith constraint equation, a and b are the indices of the two bodies involved in the constraint and $\mathbf{c}_j$ is an arbitrary point in the local space of body j:

$$\mathbf{c}_j = \mathbf{p}_j + \mathbf{R}_j\mathbf{o}_i, \tag{6.8}$$

where $\mathbf{p}_j$ is the center of mass position of body j as before, $\mathbf{R}_j$ is the matrix representation of its orientation, and $\mathbf{o}_i$ is the constraint *offset*: the constrained position in the body's local space. Using local-space positions to construct constraint equations in this manner is sometimes called the method of *connectors* [Bender et al. 2012]; [Witkin, Gleicher, and Welch 1990], hence the notation $\mathbf{c}_j$.

Equality constraints are fairly simple to work with, but the real complexity of rigid body simulation comes from *inequality* constraints. As we will see later, this is because we can no longer boil down the problem to a system of linear equations and instead must solve a *linear complementarity problem*. The most important inequality constraint is the *contact*, which models prevention of interpenetration, restitution and friction. Contact modelling is so central to rigid body simulation that many papers do not even consider other types of constraints.

A simple contact model without friction or restitution can also be defined using connectors:

$$g_i(\mathbf{x}_a, \mathbf{x}_b) = \hat{\mathbf{n}} \cdot (\mathbf{c}_a - \mathbf{c}_b) \geq 0, \tag{6.9}$$

where the $\mathbf{c}_j$s are the closest points of approach (or deepest penetration, if the bodies are already penetrating) on each body and $\hat{\mathbf{n}}$ is the contact normal: the direction along which we aim to separate the bodies. Restitution can be added fairly easily to this model with some caveats, and friction can be added with more effort and hand waving. However, we will defer the discussion of specific constraints and instead talk about how to form and solve the constrained rigid body dynamics problem.

6.5 Problem Definition

We would like to state the problem in the generic form of a *constrained minimization*[2], allowing us to use standard mathematical machinery to analyse and solve it. The standard way of defining the rigid body problem is to start from an integration rule and directly add constraint and complementarity terms. Here we will instead obtain these terms naturally as a result of applying the Karush-Kuhn-Tucker conditions to our constrained minimization problem. Without further ado, the problem we solve is:

$$\begin{gathered}\mathbf{x}^{t+1} = \min_{\mathbf{x}} \left(\frac{1}{2}(\mathbf{x} - \mathbf{x}^*)^T \mathbf{M}^t (\mathbf{x} - \mathbf{x}^*) \right), \\ \text{subject to } \mathbf{f}(\mathbf{x}) = \mathbf{0},\ \mathbf{g}(\mathbf{x}) \geq \mathbf{0}.\end{gathered} \tag{6.10}$$

Here $\mathbf{x}^*$ is the unconstrained solution:

$$\mathbf{x}^* = \mathbf{x}^t + \mathbf{d}^t + \mathbf{s}^{t+1}, \tag{6.11}$$

and $\mathbf{f}(\mathbf{x})$ and $\mathbf{g}(\mathbf{x})$ are vectors of equality and inequality constraint functions respectively. The energy we minimize is exactly the *work done by the solver* in maintaining the constraint conditions. In the absence of any constraints the minimum of this energy is the unconstrained solution $\mathbf{x}^*$, as one would expect. Note that $\mathbf{M}^t$ fixes the mass matrix at the beginning of the time step, even though it varies with orientation. This approximation works as long as angular velocities in the simulation are not too high with respect to the time step. Note that the difference operator between $\mathbf{x}$ and $\mathbf{x}^*$ is still general; a more concise way of writing

[2]Also known as a *nonlinear program* in the optimisation field.

down the minimization part would be

$$\min_{\mathbf{x}} \left(\frac{1}{2} \mathbf{v}^T \mathbf{M}^t \mathbf{v} \right), \tag{6.12}$$

where $\mathbf{v} = \mathbf{x} - \mathbf{x}^*$ is the *solver displacement*, a vector of length $6n$. A wordy way to state the problem is that we must *minimize the kinetic energy of the solver displacements used to maintain the constraints.*

An issue that immediately presents itself is that many quantities in the problem are nonlinear. The way we deal with this is by replacing the constraints with their linearisations each frame in a manner similar to, but *not* the same as, the standard optimization technique of *sequential quadratic programming* [Nocedal and Wright 2006]. In this technique we solve the full nonlinear program by constructing a sequence of Taylor expansions, each of which is a quadratic program. Our method differs in two ways: First we continue to use our current objective function, rather than the second derivative of the associated Lagrangian. This choice in effect neglects the curvature of the constraint functions [Boggs and Tolle 1995]. The second difference is that while SQP is an iterative process that attempts to find a solution to the global nonlinear problem, we only do a single iteration. In other words, at each frame we solve a linearised model of the true problem, accepting that the results will only be correct to first order. If this technique sounds familiar, that's because we used it back in Chapter 4 to define fast finite element methods: This is simply semi-implicit integration applied to (6.10)! In the absence of constraints, the two approaches are completely equivalent.

Linearising the constraints, we obtain the following *quadratic program* (QP):

$$\begin{aligned} \mathbf{x}^{t+1} = \min_{\mathbf{x}} (&\frac{1}{2} \mathbf{v}^T \mathbf{M}^t \mathbf{v}), \\ \text{subject to } &\mathbf{J}_f \mathbf{v} + \mathbf{f}^t = \mathbf{0} \\ \text{and } &\mathbf{J}_g \mathbf{v} + \mathbf{g}^t \geq \mathbf{0}, \end{aligned} \tag{6.13}$$

where $\mathbf{J}_f = (\nabla \mathbf{f}(\mathbf{x}))_{\mathbf{x}^t}$ is the Jacobian of the equality constraints at the beginning of the time step and $\mathbf{f}^t$ is the equality constraint equation vector evaluated at the beginning of the time step. $\mathbf{J}_g$ and $\mathbf{g}^t$ represent equivalent quantities for the inequality constraints. To solve this QP we turn to the Karush-Kuhn-Tucker (KKT) conditions, which are the first-order optimality conditions of this problem. To continue the analogy: in Newton's method, the first-order optimality

conditions are that the derivative of the objective function should be zero at the minimum. The KKT conditions generalize this to constrained minimization. Applied to our problem, these are:

$$\begin{aligned} \mathbf{M}^t\mathbf{v} - \mathbf{J}_f^T\boldsymbol{\lambda} - \mathbf{J}_g^T\boldsymbol{\mu} &= \mathbf{0}, \\ \mathbf{J}_f\mathbf{v} + \mathbf{f}^t &= \mathbf{0}, \\ \mathbf{J}_g\mathbf{v} + \mathbf{g}^t &\geq \mathbf{0}, \\ \boldsymbol{\mu} &\geq \mathbf{0}, \\ \boldsymbol{\mu}^T(\mathbf{J}_g\mathbf{v} + \mathbf{g}^t) &= \mathbf{0}. \end{aligned} \tag{6.14}$$

This is a *Mixed Linear Complementarity Problem* (MLCP). It can be solved in a variety of ways, including directly with Lemke's algorithm, a nonsmooth Newton method [Erleben and Ortiz 2008]; [Silcowitz-Hansen, Niebe, and Erleben 2009] or most commonly with a gradient projection technique such as *Projected Gauss Seidel* (PGS) [Coumans 2014]; [Tonge, Benevolenski, and Voroshilov 2012]. A particularly easily understood version of the latter that is very popular in the games industry is *Sequential Impulses* [Catto 2014]. EAPhysics uses a PGS method with modifications that are discussed below. We will not discuss these methods in depth; for our purpose as designers of constraints the main takeaway is that we only require a constraint equation and its Jacobian to implement a new type of joint.

6.6 EAPhysics Oddities

The preceding method is fairly well principled but is missing some important components. In particular, it only considers inelastic collisions without friction or restitution. In implementing these features our solver departs either from the method presented above or from standard practice, demanding a brief explanation here. We also use some techniques for joint constraints that are difficult to justify formally, which we will also describe.

Predictive Contacts and Restitution

Standard contact models fall into two broad camps - discrete contacts, which detect if two bodies are interpenetrating and try to push them apart, and con-

tinuous collision systems in which the simulation is advanced in such a way that collisions are caught exactly. This latter choice is expensive and difficult, but avoids situations like *tunneling* where objects can pass through one another (this is often called the 'bullet-through-paper effect'). We do not have the time budget for full continuous collision, and instead use a mixture of these two strategies called the *predictive contact*. The central idea is that the problems with discrete contacts occur when objects are moving quickly relative to their size. Therefore, we simply expand their collision shape as their velocity increases such that contact constraints are generated with *any* objects they could possibly hit during the simulation time step. This completely eliminates tunneling, although the costs for very fast moving objects can be substantial because of how many contacts we must generate. There is another problem, which is the interaction of this scheme with restitution. Consider an elastic ball approaching a wall at a high speed. We expect the ball to bounce off the wall with some fraction of its initial speed, but we cannot both give the ball the correct velocity at the end of the frame and make sure the collision appears to happen at the right time. We choose to prioritise getting the velocity correct, which results in what we call the *bounce too soon* problem - fast-moving objects will appear to bounce a frame too early. This problem is not noticeable at low velocities, and thus we make no effort to mitigate it. Fast moving objects such as foot- or golf balls can be dealt with in their own physics world using a continuous collision model, while slow-moving objects like players work well enough with predictive contacts.

Friction

Friction is probably the second most vexing complication in rigid body simulation, after the need to deal with rotations. In the Coulomb friction model, the tangential force at a contact can only have a magnitude that is some fraction of the normal force, determined by the coefficient of friction. One way that this is often stated is that the force at a contact point must lie inside a cone pointing in the direction of the contact normal; if the contact force is outside this cone then the tangential friction is stronger than the normal force which is not physical. Early work [Stewart and Trinkle 1996] in the field used a polygonal approxima-

tion to the friction cone, which allowed friction to be neatly incorporated into the LCP formulation. However, it turns out this approximation prevents us from using certain methods (such as PGS) to solve the resulting problem. Early work on rigid body simulation generally used Lemke's algorithm for solving the LCP, which although robust in this case is nowadays considered too slow for general use in real-time.

Instead of a friction-cone approximation, it is more usual in modern real-time simulation to decouple the normal and tangential directions at a contact and treat friction as a secondary effect. We can solve for the normal component of the collision impulse, and simply work out the frictional force using the coefficient of friction, applying it after the normal component in the solver loop. This technique has the advantage of computing the exact friction: the locus of contact forces is the interior of the Coulomb friction cone and not a polygonal approximation. However, this method is hard to justify at the LCP level and has to be added to the PGS method directly. It introduces a coupling between the normal and tangential parts of the contact that effectively make the PGS fixed-point iteration process *nonlinear*. Consequently there is no proof of convergence for this technique, unlike the polygonal cone approximation for which Lemke's algorithm is guaranteed to find an answer. Nevertheless, this technique is widespread and tends to work extremely well despite being difficult to justify from first principles.

High Fidelity Joints

Our application of semi-implicit integration to our constraint equations means that without further measures, any nonlinear constraints will remain violated at the end of the time step *even if we accurately solve our LCP*. The most visible constraints of this type are angular joints. By representing our angular constraint equations as their first-order Taylor expansions, we are effectively solving the small-angle approximations of our constraints rather than the constraints themselves. This can result in a noticeable springiness in angular joints even when the underlying LCP is solved exactly. One way we have found to deal with this problem is to simply evaluate the angular constraint *error* exactly. This way the correct error term is always used and *if* the PCG iteration process converges

the constraints will have been solved exactly. Crucially, we have found we can get away with not updating the associated *Jacobian* at all, at least for simple angular joints. This of course makes the PCG iteration process nonlinear. If our original method of a single iteration of SQP was akin to applying Newton's method to the position-level problem, then this alteration to the method is akin to a *quasi-Newton* method - we attempt to solve a nonlinear problem using a tangent matrix that we update as little as possible; in our case not at all. We call this technique the *High Fidelity Joint*.

6.7 Concluding Remarks

In this chapter we discussed EA's rigid body simulation framework, concentrating on differences from other more standard methods. We covered three specific differences when compared to published systems: Solving constraints purely at the position level, using a predictive contact model, and trading accuracy for convergence speed with the high fidelity joint method. However, the main features of our algorithm are similar to other real-time methods. In the next chapter, we describe a specific type of rigid body constraint that we can use to enable efficient simulation of ragdolls in games.

Chapter 7

Rod Constraints

Ragdoll simulation is a common use case for rigid body physics engines. A ragdoll is a piecewise-rigid representation of a character's body connected with joints that mimic the kinematics of the human skeleton, allowing for a more realistic representation of a character in the game physics world. A common use for a basic ragdoll is for a character to collapse realistically after being (for example) shot. Adding a representation of the character's intent in the form of muscle constraints can allow for richer simulation, such as characters pushing against each other to achieve a desired pose. Whatever the use they are put to, ragdolls can be one of the more performance-intensive parts of a physics simulation. Beyond the simple cost of simulating more rigid bodies, the tightly coupled nature of ragdolls greatly increases the number of solver iterations required for reasonably accurate simulation. It is thus of great importance to make ragdolls as simple as possible. One region of a ragdoll where this is problematic is the spine; simply decreasing the number of bodies and joints along the spine leads to a kinematics that poorly represents the motion of the human skeleton. In this chapter, we propose a specialised constraint that allows us to have reasonable kinematics with fewer bodies along the spine. We do this by modelling the spine as a simple plane circular curve.

7.1 Related Work

Simulation of elastic rods is a popular field, so we will only cover the most closely related work here. The reader is referred to the recent survey paper [Ward et al. 2007] for a more comprehensive view of the field as it pertains to computer graphics. Maximal coordinate methods based on mass-spring networks [Selle, Lentine, and Fedkiw 2008] and shape matching [Rungjiratananon et al. 2012]

can be very fast and can capture bending and twisting dynamics effectively, but have difficulty preserving length.

Other researchers adopt reduced coordinate approaches. Bergou et al. [2008] perform a comprehensive discretisation of Langer and Singers' work [1996] on Kirchhoff rods, allowing them to accurately simulate instability phenomena such as the buckling of elastic rings. Bertails et al. [2006] use helical elements of constant curvature and torsion to model curled hair wisps. Chains of rigid bodies [Hadap and Magnenat-Thalmann 2001] have also been used effectively. Methods that use the control points of spline curves as reduced coordinates [Remion, Nourrit, and Gillard 1999]; [Nocent and Remion 2001] have been used to accurately model rod-like mechanical parts [Theetten et al. 2008] as well as knitted cloth [Kaldor, James, and Marschner 2008]. Discounting collision detection, these methods are not overly expensive. However, for reasons discussed previously we prefer not to use reduced coordinate methods.

There is little academic work on ragdoll construction. Techniques which synthesize motion at runtime such as [Yin, Loken, and Panne 2007] have no need to interpolate from (and extrapolate back to) an authored animation skeleton, and thus can use ragdolls that are as simple as possible. However, much work has been done on alternative joint representations for inverse kinematics. Lee and Terzopolous [2008] developed a joint model based on spline surfaces, with the spline parameters as the reduced coordinates of the simulation. This method allowed them to model complex joints, although it would be prohibitively expensive to use in a maximal coordinate setting. Engell et al. [2012] proposed a data-driven approach based on distance fields in angle-space, allowing them to model the complex constraint manifold of the shoulder joint with constant-time constraint projections, at the cost of a large memory footprint.

7.2 Contributions

We show how to model inextensible elastic rods as constraints between rigid body pairs. We define the rod's shape implicitly by the position and orientation of the two bodies, imposing an additional constraint to ensure the resulting shape is convenient - that of a circular arc. We emulate the bending and twisting forces

using soft constraints, allowing stiff rods to be stable even when using long time steps.

We improve the convergence behaviour of character ragdolls by replacing the many joints in the spine region with our rod constraint. This leads to significantly reduced joint violation when using a fixed number of iterations, as well as improved controllability.

7.3 Background

We adopt the uniform variation of the Kirchhoff elastic rod model [Langer and Singer 1996], which describes a rod Γ as a centerline curve $\boldsymbol{\gamma}(s)$ and an orthonormal material frame $F(s) = \{\mathbf{t}(s), \mathbf{m}_1(s), \mathbf{m}_2(s)\}$, representing the rotation of the rod's cross-section about the centerline. The material frame is called *adapted* because $\mathbf{t}$ is tangent to the centerline: $\mathbf{t}(s) = \boldsymbol{\gamma}(s)' = \frac{d}{ds}\boldsymbol{\gamma}(s)$. Because we assume inextensibility $s \in [0, 1]$ must be an arc length parameterisation, and $|\mathbf{t}(s)| \equiv 1$.

The elastic energy of the rod is related to the bending and twisting strains:

$$E(\Gamma) = \frac{1}{2}\int k_b\kappa^2 \, ds + \frac{1}{2}\int k_t\tau^2 \, ds, \tag{7.1}$$

where $\kappa = |\mathbf{t}'|$ is the bending strain or *curvature* of the centerline and τ is the twisting strain, which is the angle between the material frame and the *natural frame* of the centerline. The natural frame is one of many possible framings of a space curve, with the distinguishing factor that it has no inherent twist. Because we assume the rod is inextensible, there is no stretching component to the energy.

While a full discussion of rigid body physics is beyond the scope of this dissertation (see [Bender et al. 2012] for a recent survey), we should still make some notation clear. We consider a rigid body i to be an oriented particle with coordinates $\mathbf{x}_i = \{\mathbf{p}_i, \mathbf{q}_i\}$, where $\mathbf{p}_i$ is the position of its center of mass and $\mathbf{q}_i$ is a quaternion representing its orientation. Each body is equipped with a mass m_i and inertia $\mathbb{I}_i$. We use an impulse-based method [Bender and Schmitt 2006], solving the system from a set of preview coordinates $\mathbf{x}^*$. We deal with collision detection and contact resolution in standard ways, and as they are orthogonal to the purpose of this paper we will not discuss them here.

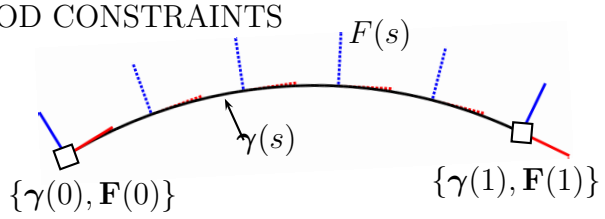

Figure 7.1: Adapted framed curve between rigid bodies.

7.4 Our Method

To build a rod constraint between the rigid bodies $\mathbf{x}_a$ and $\mathbf{x}_b$, we need the ingredients of a rod in the Kirchhoff model: a vector-valued curve function $\boldsymbol{\gamma}(s, \mathbf{x}_a, \mathbf{x}_b)$ and a quaternion-valued material frame function $\mathbf{F}(s, \mathbf{x}_a, \mathbf{x}_b)$. We specify that each end of the rod is attached to a rigid body at its center of mass (Figure 7.1):

$$\begin{aligned} \boldsymbol{\gamma}(0) = \mathbf{p}_a, \boldsymbol{\gamma}(1) &= \mathbf{p}_b, \\ \mathbf{F}(0) = \mathbf{q}_a \mathbf{F}_a^{\text{bod}}, \mathbf{F}(1) &= \mathbf{q}_b \mathbf{F}_b^{\text{bod}}, \end{aligned} \tag{7.2}$$

where $\mathbf{F}_i^{\text{bod}}$ is the orientation of the rod in body i's local space. The *body tangents* $\mathbf{t}_i$ and *body normals* $\mathbf{n}_i$ are then defined as directions in $\mathbf{F}_i^{\text{bod}}$.

Given a specific form for the curve $\boldsymbol{\gamma}(s)$, we can constrain its length and find soft constraint equivalents of the bending and twisting forces. Our constraint equations are scalar or vector functions of the coordinates of rigid body pairs $\mathbf{C}(\mathbf{x}_a, \mathbf{x}_b)$ that are equal to zero when the constraint is satisfied.

7.4.1 Length constraint

To constrain the length of the curve $\boldsymbol{\gamma}(s)$, we need to compare its arc length to the length of the undeformed rod l_0:

$$C_{\text{len}}(\mathbf{x}_a, \mathbf{x}_b) = \int_0^1 |\boldsymbol{\gamma}'(s, \mathbf{x}_a, \mathbf{x}_b)| \, ds - l_0. \tag{7.3}$$

To enforce this constraint we need to be able to calculate the arc length of $\boldsymbol{\gamma}$, which in general is a hard problem. While we can numerically obtain an arc length parameterisation for any curve, the amount of computation required would render the constraint unsuitable for real-time use. This problem must inform our choice of $\boldsymbol{\gamma}$.

7.4.2 Elastic constraints

Instead of using forces to simulate the bending and twisting behaviour of the rod, we will use soft constraints. This is a technique used to great effect in the field of *position-based dynamics* [Müller et al. 2007]. The major advantage over force-based methods is guaranteed stability: increasing the strength of the constraint will not create numerical stiffness. To use this technique we take the energy terms as constraint equations:

$$C_{\text{bend}}(\mathbf{x}_a, \mathbf{x}_b) = \frac{1}{2}\alpha_b \int_0^1 \kappa^2 \, ds, \tag{7.4}$$

$$C_{\text{twist}}(\mathbf{x}_a, \mathbf{x}_b) = \frac{1}{2}\alpha_t \int_0^1 \tau^2 \, ds. \tag{7.5}$$

Note that instead of bending and twisting stiffnesses k_b and k_t we use equivalent relaxation factors α_b and α_t. In position-based dynamics we make this substitution because our constraints will be solved as velocity updates rather than forces. True stiffness constants have a range of 0 (completely soft) to infinity (completely rigid). Using specific relaxation factors re-scales this range from 0 to 1.

The major disadvantage of this approach is a loss of physical realism. In particular, using soft constraints will cause damping of the resulting oscillations. For games this is not a big problem, as stability is valued over physical realism. Another issue encountered by Müller et al. [2007] is that it is difficult to estimate the effect of a given specific relaxation when compounded over many sequential iterations. In their case they were able to find a suitable re-scaling, but for more complex constraints this may not be possible.

7.4.3 Choice of curve

Note that we have left the curve function $\gamma(s, \mathbf{x}_a, \mathbf{x}_b)$ undefined. The expressiveness and efficiency of this constraint hinge strongly on our choice of curve type. We require a curve that is fully determined by the body positions and frames, and which has an easily-calculated arc length. One might consider following Theetten et al. [2008] and using a cubic spline curve to form γ. This passes the first test but fails the second, as expressions for the arc length of cubic splines are complex.

We thus instead follow Bertails et al. [2006] and choose to restrict our curves to those with easily-obtainable expressions for arc length, curvature and torsion.

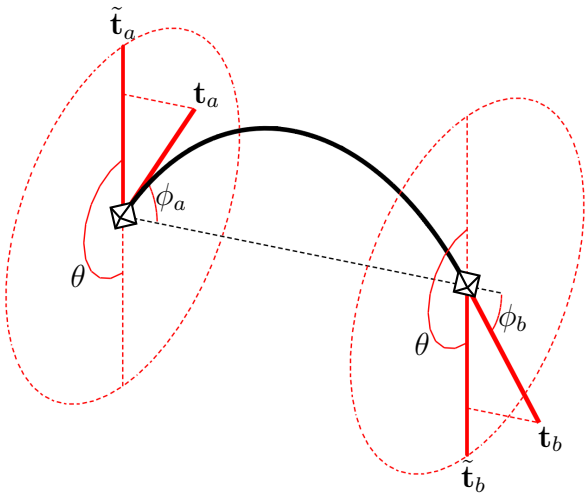

Figure 7.2: Visualisation of the arc constraint. The angle θ between the tangents' rejections onto $\hat{\mathbf{o}}$ ($\tilde{\mathbf{t}}_a$ and $\tilde{\mathbf{t}}_b$) must be equal to π, and the angles ϕ_a and ϕ_b must be equal and opposite.

While they chose helical segments, which have constant curvature and constant Frenet torsion, we choose circular arcs, which have constant curvature and zero torsion. This choice satisfies the second condition, but poses some problems for the first. While a unique cubic Hermite spline exists for any set of coordinates $\mathbf{x}_a$ and $\mathbf{x}_b$, this is not true for circular arcs. We need to include an additional constraint on the two bodies that will force them into an admissible configuration - one in which we can draw a valid circular arc between them.

7.4.4 Curve constraint

To constrain a rigid body pair such that they have an adjoining circular arc, we need to find sufficient conditions on the positions and orientations of the bodies at each end of the curve. We do this by inspecting a valid arc (Figure 7.2). We note that $\mathbf{t}_a, \mathbf{t}_b$ and $\hat{\mathbf{o}}$ must all lie in the same plane, which means that the angle θ must be equal to π. We also note that the angles ϕ_a and ϕ_b must be equal and opposite. Effectively, these conditions require that $\mathbf{t}_b$ be the *reflection* of $\mathbf{t}_a$ about $\hat{\mathbf{o}}$, and we can encode this using a similar equation to specular reflection about a surface normal:

$$\mathbf{C}_{\text{curve}}(\mathbf{x}_a, \mathbf{x}_b) = \mathbf{t}_a + \mathbf{t}_b - 2(\mathbf{t}_a \cdot \hat{\mathbf{o}})\hat{\mathbf{o}}. \tag{7.6}$$

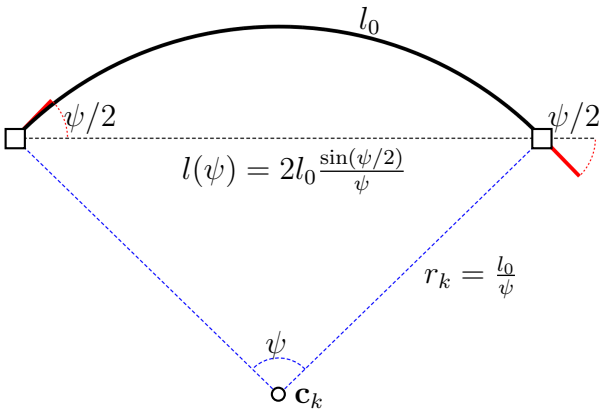

Figure 7.3: Circular arc in the plane of curvature. Assuming the rod is the correct length l_0, the length of the offset l can be determined as a function of ψ.

7.4.5 Specific form of length and elastic constraints

When this constraint is satisfied we can look at the bodies' configuration in the plane of curvature (Figure 7.3) to find simple expressions for the length preservation and elastic constraints. We can define $\boldsymbol{\gamma}(s)$ as a sweep about the center of curvature $\mathbf{c}_k$ from $\mathbf{p}_a$ to $\mathbf{p}_b$:

$$\boldsymbol{\gamma}(s, \mathbf{x}_a, \mathbf{x}_b) = \mathbf{c}_k + \tilde{r}(s)(\mathbf{p}_a - \mathbf{c}_k), \tag{7.7}$$

where $\tilde{r}(s)$ is the matrix form of an axis-angle rotation about the plane normal:

$$r(s, \mathbf{x}_a, \mathbf{x}_b) = \psi s \hat{\mathbf{n}}, \tag{7.8}$$

where $\psi(\mathbf{x}_a, \mathbf{x}_b) = \arccos(\mathbf{t}_a \cdot \mathbf{t}_b)$ is the curve bending angle. We take the approach of correcting the length purely through linear motion along the offset axis; thus assuming a fixed bend angle ψ we can derive the correct length of the offset for a given rest length l_0:

$$C_{\text{len}}(\mathbf{x}_a, \mathbf{x}_b) = |\mathbf{p}_b - \mathbf{p}_a| - 2l_0\frac{\sin(\psi/2)}{\psi}. \tag{7.9}$$

With this constraint satisfied, we notice the radius of curvature $r_k = |\mathbf{c}_k - \mathbf{p}_a| = l_0/\psi$. Thus the curvature is constant over the rod:

$$\kappa = \frac{1}{r_k} = \frac{\psi}{l_0}, \tag{7.10}$$

and the bending constraint is

$$C_{\text{bend}}(\mathbf{x}_a, \mathbf{x}_b) = \frac{\alpha_b \psi^2}{2l_0^2}. \tag{7.11}$$

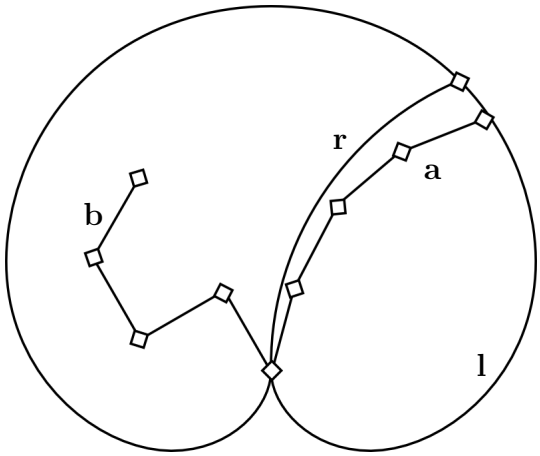

Figure 7.4: Comparison of the kinematics of chains of ball and socket joints and a single rod constraint. If each joint in a chain makes the same angle (chain **a**), then the locus of points the end of the chain can achieve (curve **l**, a cardioid) is similar to that of the rod constraint (curve **r**). If this is not true (chain **b**) then the assumption breaks down.

We recall that the twist τ is the angle between the natural and material frames. We know the material frames at the beginning and end of the circular arc: $\mathbf{F}(0) = \mathbf{q}_a\mathbf{F}_a^{\text{bod}}$, $\mathbf{F}(1) = \mathbf{q}_b\mathbf{F}_b^{\text{bod}}$. Thus we can find the total angle between the two frames and subtract the bending angle, leaving us with the twisting angle:

$$\tau = \xi - \psi, \tag{7.12}$$

where ξ is the total angle between the quaternions $\mathbf{F}(0)$ and $\mathbf{F}(1)$. The twisting constraint is thus:

$$C_{\text{twist}}(\mathbf{x}_a, \mathbf{x}_b) = \frac{\alpha_t \tau^2}{2}. \tag{7.13}$$

7.5 Ragdoll Considerations

Although our rod constraint has a range of motion similar to a group of ball and socket joints, the two are not interchangeable. Specifically, if the internal angles of the ball and socket chain are very different to one another then the end of the chain can take very different positions to the end of the rod constraint (Figure 7.4). The rod approximation is thus closest to being valid when there are

a large number of joints in the chain, and the internal angles are all the same. Fortunately, this is usually close to the truth in animations authored for games.

However, most animations do deviate by some amount from this ideal state. A very important case to consider for ragdolls is that a character is initially animated kinematically, and some action from the player causes it to transition to a ragdoll. This means we need to allow any configuration of the ragdoll given to us by a source animation. We can deal with this by instantiating the rod only when the transition occurs, setting its rest length and body frames such that the new constraint is valid.

Finally, we need to interpolate the results of the simplified simulation back to intermediate animation bones that do not have associated physics bodies. We can do this by linearly interpolating the position or orientation changes of the rigid body at the top of the spine, sharing them out between the animation bones. If we use the orientation changes only, the bone lengths will be conserved but the end of the chain of animation bones will end up in a different position to the rigid body (this effect gets worse if the spine is severely kinked). If we also interpolate the position changes, the bone lengths will not be conserved but the positions will match. We have found using orientations only tends to produce fewer artifacts in practice.

7.6 Implementation

Although our rod model consists of several constraint equations, none of these are particularly useful individually. By solving all the constraints together in one function, we can share some calculations common to each and end up doing less work overall.

Our solver and constraint functions are implemented in our physics engine as branch-free SIMD code, solving four constraints at once. It takes on average 60 ns to solve one rod joint on a 2.4GHz Intel Xeon CPU, while the equivalent ball and socket joint takes 20ns on average. Because the number of solver iterations we need to run is usually determined by the complexity of our ragdoll, removing these joints will enable us to use fewer total iterations. In a sports-game situation this can mean savings of 50%, possibly making multiple-ragdoll simulation viable

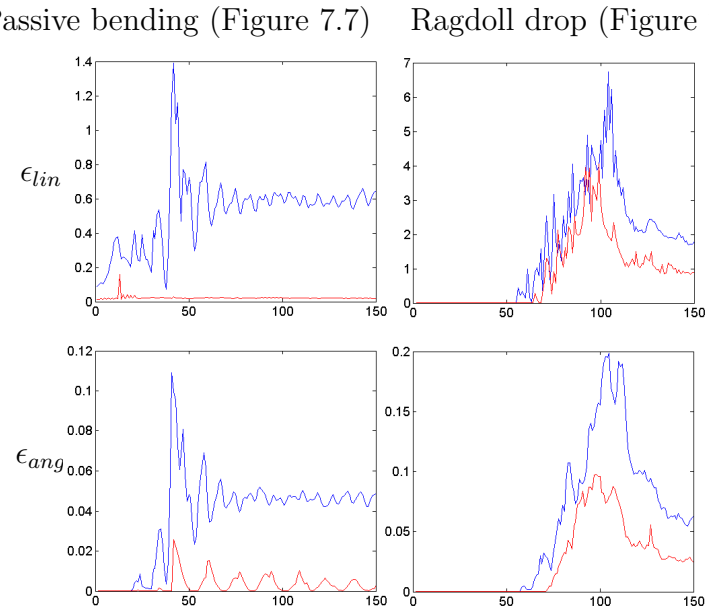

Figure 7.5: Residual error per frame after 20 iterations in ragdolls using spines with three joints (blue) and our rod constraint (red). The top graphs show the total dislocation of the joints in each ragdoll in cm, and the bottom graphs show the total angular violation in radians. The specific test cases can be seen in Figures 7.7 and 7.8.

in situations it was not before our simplification.

7.7 Results

Because game physics engines typically use a fixed number of iterations, slow convergence of the constraint solver manifests as residual error at the end of each frame. It thus makes sense to look at this metric when comparing methods. Figure 7.5 shows that there is a dramatic improvement in residual error between a standard and simplified ragdoll in a passive bending scene, and a smaller but still consistently positive difference in a more natural scene that includes collisions.

This improvement in convergence speed is particularly important when we try to control the ragdoll. In methods such as PD control, the kinematic depth of the ragdoll has a major effect on the ability of a character to follow a target animation. As a crude metric for controllability, we can look at each ragdoll's ability to maintain a target pose using internal torque constraints in the face of external forces. Figure 7.6 shows each type of ragdoll's response to such a

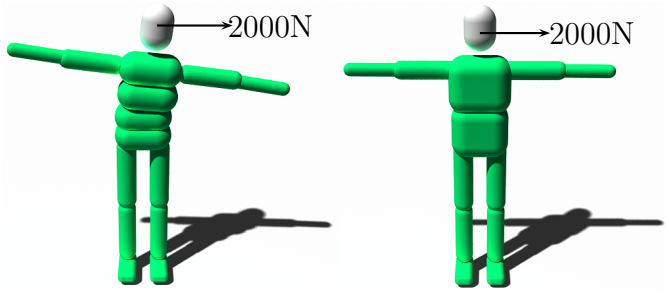

Figure 7.6: Comparison of controllability between a ragdoll with three joints in the spine (left) and a single rod constraint (right). With the hips fixed, the characters try to maintain their initial pose in the face of an external force.

force. Even though the constraints driving the ragdoll to the target pose have infinite strength, the larger number of joints in the spine causes the left ragdoll to converge to the target pose very slowly, resulting in a noticeable deviation after 20 solver iterations. Replacing this chain of joints with our rod constraint allows forces to propagate through the right ragdoll in fewer iterations, leading to a greater ability to maintain the target pose.

It should be noted that our ball and socket joints are solved using simplified methods that are cheap but converge slowly, and thus the error behaviour is not directly comparable to other published methods that are more accurate. However, our implementation of the rod constraint also uses cheap approximations. It should always be possible to get significant benefits from reducing the number of joints in the spine if one replaces a group of ball and socket constraints with a single similarly accurate rod constraint.

It should also be mentioned that one could achieve the same improvements in convergence behaviour by simply reducing the number of ball and socket joints in the spine to one. However a single joint of this type has very different kinematics to a chain of joints, and the resulting motion looks unnatural[1]. Using our rod constraint allows the ragdoll to both converge quickly and look plausible.

[1]See Figure 7.4; the kinematics of a single joint are quite far from the cardioid we expect.

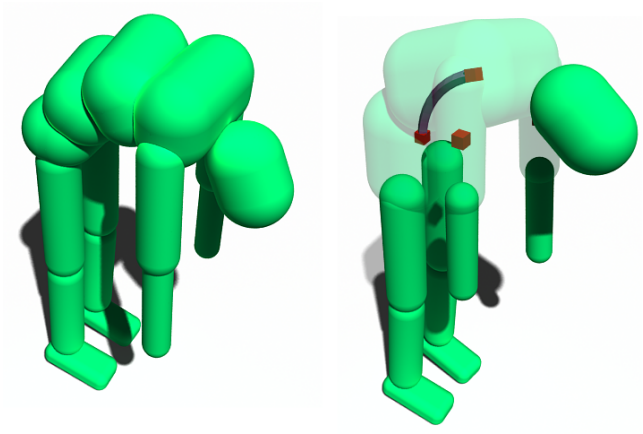

Figure 7.7: Comparison between a standard ragdoll with three joints in the spine (left) and a simplified ragdoll with a single rod constraint (right), both slumping forward under gravity.

7.8 Limitations and Future Work

The restriction of the constraint curve to a circular arc is useful for keeping control of our ragdolls, but it does not lend itself to a truly expressive model of Kirchhoff elastic rods. An interesting extension would be to follow [Bertails et al. 2006] more closely and use helical constraint curves. This would mean finding a set of constraint equations that could project the configuration of two arbitrary rigid bodies to the nearest configuration with a valid adjoining helix.

The other major candidate for simplification in ragdolls is the shoulder region. Unlike the spine, the shoulder does not have a simple kinematic approximation. We believe the problem of finding an expressive shoulder constraint could be best approached in a data-driven way, similar to that taken by Engell et al. [2012].

7.9 Concluding Remarks

In this chapter, we introduced a new type of constraint between two rigid bodies that connects them with a curved rod. We used a specific type of curve constraint to replace the spines of character ragdolls, allowing us to make significant performance savings. While this constraint does not perfectly follow the kinematics of the real spine (two rigid bodies clearly lack the degrees of freedom for this), we have found it to be a useful approximation. The circular arc curves we used for

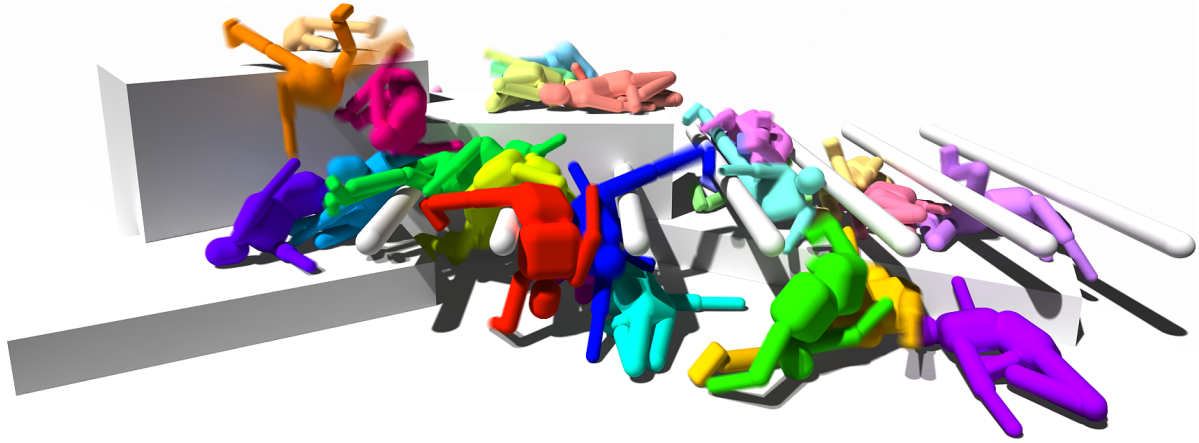

Figure 7.8: 30 simplified ragdolls fall together into a complex, compact physics world.

our spines are too constrained for general rod simulation in three dimensions; a sensible extension would be to use helical rather than circular segments for this purpose. While our current method is quite specific to the spine use case, this modification could make for a much more general Kirchhoff rod simulation using rigid bodies. This could be useful if game designers required inextensible rod-like structures, such as a car antenna or a coiled spring. The natural rod kinematics of our approach would still provide the significant advantage of requiring fewer degrees of freedom than a similar assembly of standard joints to model the required behaviour.

Chapter 8

Conclusion

This dissertation has presented some steps towards improving the realism of game physics simulation. We introduced a new approach for simulating soft bodies in real-time, which solved finite element strain energy equations using the Position Based Dynamics framework (Section 5.3). This led to a robust and reliable method in which we could incorporate hard contacts and other inequality constraints.

While we have found this method to be a useful workhorse, it lacks the global convergence of matrix-based methods like Projective Dynamics (PD). To try to capture this advantage without the expensive drawback of having to recalculate the system matrix in the presence of contacts, we modified the PD method into an iterative process that uses a constant system matrix as a highly effective preconditioner (Section 5.5). This allowed us to add and remove arbitrary numbers of constraints per time step without causing large spikes in processing cost, which is an important quality in a game simulation. While this method works well, the problems of rotation locking and the use of the penalty method mean we cannot use it in all situations. Complex collision environments and high stiffness values tend to exacerbate these problems, and PBD is superior when these qualities are unavoidable. Simulation of cloth and soft bodies on characters fits squarely into this category, and so we cannot currently rely on the fast global convergence of PD in this important case. Alleviating the rotation locking effect while keeping the advantage of a constant stiffness matrix is an interesting, untackled problem that might be approached using some kind of energy budgeting [Su, Sheth, and Fedkiw 2013]. Adding true inequality contacts while maintaining efficiency is also interesting; we must ask the question of whether we can gain any advantage from a large constant equality block when solving a quadratic program. Possibly a

nonsmooth Newton style solver [Erleben and Ortiz 2008] might be the answer.

While on the one hand we want to make characters more believable, there is also constant pressure to improve the efficiency of the technology we already have. Our rod constraint presented in Chapter 8 is a result of this pressure, designed specifically as a convenient simplification for one part of a character. This constraint enabled us to significantly decrease the number of rigid body solver iterations required to stably simulate ragdolls, making multiple-ragdoll simulation possible on console hardware. The idea of curved geometry constraints has much room left for exploration, however. Although we discounted the idea of using cubic splines for our curve representation for reasons of efficiency, it seems possible that such a representation could be viable if the right approximations were used. Our current rod constraint is highly specialised to the task of simulating the spine, and it does not allow one to represent other rod-like structures, like loose cables or strands of hair. It would be interesting to see if a polynomial model would prove superior in this case.

Appendix A

Sparse Linear Algebra

Much of numerical analysis is concerned with matrices: their formation and decomposition. In computer graphics we are mostly concerned with a particular flavour of matrix: the square, symmetric, sparse, positive definite kind. What exactly this means, and what are the best methods for dealing with these matrices, will be discussed in this chapter. This material is very well known but of such monumental importance in computer graphics that having a compact account of the relevant parts of the field is particularly useful.

A.1 On Matrices

Matrices are a compact notation for representing linear relationships between vectors. Note here the word *notation*. It should be emphasized from the outset that matrices are a notational convenience *only*, albeit an exceptionally useful one. For instance, the system of equations

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{A.1}$$

can be written just as well with no linear algebra at all using indices:

$$x_i = \sum_{j=1}^{n} A_{ij} b_j. \tag{A.2}$$

For systems that involve only relationships between vectors, matrices are the dominant notation. In fields where higher-order relationships between tensors are a reality, such as theoretical physics, indicial representations such as the summation convention are more prevalent. In many fields, computer graphics being one of them, people generally bend quite far to continue using matrices even when other representations might be more appropriate[1].

[1]The use of Voigt notation in some treatments of continuum mechanics is a prominent example.

Positive Definiteness

The concept of positive definiteness comes up with regularity in computer graphics. The easiest way to describe what this means is by using the *quadratic form* of the matrix. The quadratic form for a linear system $\mathbf{Ax} = \mathbf{b}$ is simply the function

$$\mathcal{F}_q(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{Ax} - \mathbf{b}^T\mathbf{x}. \tag{A.3}$$

This function is a quadratic scalar field; the critical point $\mathbf{x}_c$ of this field is the solution to the original equation (i.e. $\mathbf{x}_c = \mathbf{A}^{-1}\mathbf{b}$). We can easily see this by writing down the optimality conditions of this problem as previously discussed. If $\mathbf{A}$ is positive definite then the critical point is also the global minimum of f and we can find it purely by following f's gradient. It turns out that for this to be true we need the condition $\mathbf{x}^T\mathbf{Ax} \geq 0$ to be true for all $\mathbf{x}$. It's helpful to illustrate this with some figures:

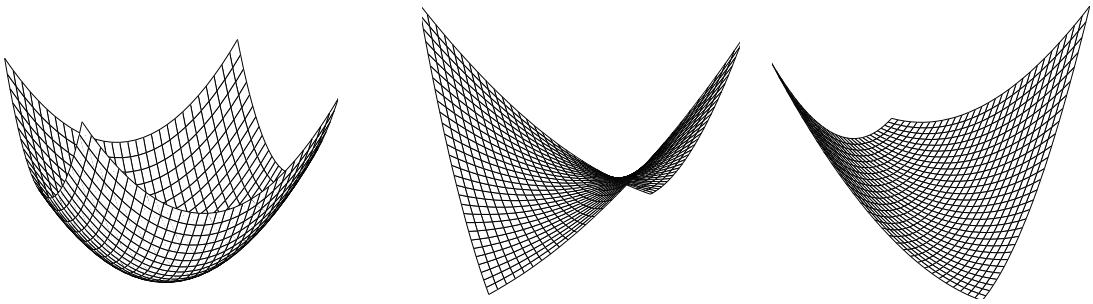

Figure A.1: Quadratic forms for 2×2 matrices. Left: positive definite. Middle: indefinite. Right: singular.

The positive definite matrix has one critical point, which is also the global minimum. The indefinite matrix has one critical point which is a saddle point (there is no global minimum). The singular matrix has an infinite number of critical points in a line. The practical importance of this is that we can find the minimum of a positive definite f (and thus the solution to the original linear system) purely by sliding down its gradient. Trying to do this for the indefinite case would cause us to slide off to infinity, and the singular matrix does not have a unique solution anyway. This idea of gradient descent is central to the more advanced Conjugate Gradient method, which we will discuss later.

So an interesting question to ask is: Why are the matrices generated by

Newton's method positive definite? Newton's method is used for minimization of nonlinear functions, by approximating that function at each step with a linear model. If the minimization is feasible using Newton's method (the function is sufficiently continuous and does not have inconvenient local minima) then *near the solution* the function must look like the positive definite quadratic form, i.e. have a single critical point that is also the global minimum. Thus the matrix associated with the linear model must also be positive definite. When we are further away from the global minimum, this assumption can break down and we may find that some matrices generated by Newton's method are not positive definite for large systems and large time steps.

Sparsity

Matrices encountered in regular 3D mathematics are generally dense and asymmetric. The large matrices resulting from the Finite Element method, however, are usually symmetric and sparse. Symmetry is fairly easy to understand - the bottom triangle of the matrix is the same as the top triangle. This symmetry in the matrix comes from symmetry in the underlying equations (i.e from conservation of energy and momentum). Clearly, knowledge of symmetry in a matrix allows us to roughly halve the amount of work involved in any algorithm. Sparsity is a little more complex. Consider a regular hexahedral finite element mesh where every node is involved in at most 8 hexahedral elements. Each node is connected to at most 26 other nodes, which means there will be at most 27 entries in its corresponding row in the FEM matrix. For a mesh of 1000 nodes, the number of nonzero entries n_{nz} in the matrix is at most 26000, which is only 2.6% of the total number of entries including zeros. This difference gets more pronounced as the number of nodes increases. This means that using dense linear algebra to store finite element matrices is incredibly inefficient, since almost all of the entries stored will be zeros.

A.1.1 Data Structures

Symmetry and sparsity can be useful qualities when constructing efficient algorithms. Symmetry is easy to exploit: we simply store only the lower triangle of

the matrix and infer the rest when implementing our algorithms. We can roughly halve the amount of data required to store a symmetric matrix in this way. Sparsity is harder to exploit; we must store the matrix in a completely different way. The two most useful data structures for sparse matrices are the *triplet array* and *compressed column* structures.

Triplets: A triplet is simply a trio specifying a single entry in a matrix: a row index i, column index j and value at that location x. A sample definition in C++ might be:

```
struct Triplet
{
    int i, j;
    float x;
};

typedef std::vector<Triplet> TripletArray;
```

Entries can simply be appended as the contributions of different elements are added to the matrix. Duplicate entries that reference the same row and column are simply added together when the list is traversed. Triplet storage is very useful for *constructing* matrices but not great for *processing* their contents. For that it is better to use compressed storage.

Compressed Column Storage (CCS): This is the data structure of choice for working with matrices which we have already defined. For a general sparse matrix without special structure, CCS is the most optimal storage solution. In this scheme, we order the entries in such a way that we do not have to store as many indices as the triplet array. Each entry still has an individual row index, but the column indices are made implicit using another array. A definition for a matrix with size n and n_{nz} nonzero entries might be:

```
struct SparseMatrix
{
   int* colBeginIndices; // size n
   int* is; // size nnz
   float* xs; // size nnz
```

```
};
```

The `colBeginIndices` array contains the index of the first entry in the `is` array and the first entry in the `xs` array corresponding to each column. While we can very efficiently iterate over the entries of a matrix stored in this way, adding or removing entries essentially requires us to reallocate the entire structure. Searching for an arbitrary entry is also inefficient. Fortunately, it turns out that common matrix operations such as multiplication only require us to iterate over the entries in order:

```
// Multiplication of vector b by sparse symmetric matrix A
// giving result r (r is initialized to zeroes).
// First iterate over columns:
for(int j = 0; j < n; ++j) {
    // Iterate over the entries in this column:
    int rowBeginEntry = A.colbeginIndices[j];
    int rowEndEntry = j == n-1 ? nnz : A.colBeginIndices[j+1];
    for(int entry = rowBeginEntry; entry < rowEndEntry; ++entry) {
        int i = A.is[entry];
        float x = A.xs[entry];
        if(j == i) {
            // Diagonal entries are only considered once.
            r[i] += x*b[j];
        } else {
            // Off-diagonals have a symmetric pair.
            r[i] += x*b[j];
            r[i] += x*b[i];
        }
    }
}
```

As we will see shortly, solving triangular systems of the form $\mathbf{Lx} = \mathbf{b}$ is an important building block. This is very similar to the multiplication operation:

```
// Multiplication of vector b by sparse symmetric matrix A
```

```
// giving result r (r is initialized to zeroes).
// First iterate over columns:
for(int j = 0; j < n; ++j) {
    // Iterate over the entries in this column:
    int rowBeginEntry = A.colbeginIndices[j];
    int rowEndEntry = j == n-1 ? nnz : A.colBeginIndices[j+1];
    for(int entry = rowBeginEntry; entry < rowEndEntry; ++entry) {
        int i = A.is[entry];
        float x = A.xs[entry];
        if(j == i) {
            // Diagonal entries are only considered once.
            r[i] += x*b[j];
        } else {
            // Off-diagonals have a symmetric pair.
            r[i] += x*b[j];
            r[i] += x*b[i];
        }
    }
}
```

A.2 Cholesky Decomposition

The most common task involving matrices is solving linear systems $\mathbf{Ax} = \mathbf{b}$. When the matrices we are using are only 3×3 or 4×4, we can get away with computing the inverse $\mathbf{A}^{-1}$ by Cramer's rule and simply finding $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$. However, this approach becomes completely untenable as the size of the system increases. A wiser approach is either to use an iterative technique to solve the problem, such as the conjugate gradient method discussed later, or to compute one of the many *matrix decompositions* available. For positive definite matrices the appropriate tool is the Cholesky decomposition, which allows us to compute a lower triangular matrix $\mathbf{L}$ whose square is the original matrix $\mathbf{A}$:

$$\mathbf{L}\mathbf{L}^T = \mathbf{A}. \tag{A.4}$$

It is very easy to solve triangular linear systems, so using this equivalence we can compute the solution $\mathbf{x} = \mathbf{L}^{-T}(\mathbf{L}^{-1}\mathbf{b})$ efficiently. The Cholesky decomposition only exists for positive definite matrices, and is in some sense the 'square root' of the matrix, since multiplying it with itself will yield the original.

A.2.1 Computing the decomposition

Equation A.4 actually gives us enough information on its own to compute the matrix $\mathbf{L}$ for a given $\mathbf{A}$. The best explanation I have found is given by [Davis 2006]: Consider the block decomposition of $\mathbf{L}\mathbf{L}^T = \mathbf{A}$:

$$\begin{bmatrix} \mathbf{L}_{00} & \mathbf{0} \\ \mathbf{l}_{01}^T & l_{11} \end{bmatrix} \begin{bmatrix} \mathbf{L}_{00}^T & \mathbf{l}_{01} \\ \mathbf{0} & l_{11} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_{00} & \mathbf{a}_{01} \\ \mathbf{a}_{01}^T & a_{11} \end{bmatrix}, \tag{A.5}$$

where $\mathbf{L}_{00}$ is an $n-1 \times n-1$ matrix, $\mathbf{l}_{01}$ an $n \times 1$ vector and l_{11} a scalar. We can expand this out to three equations:

$$\begin{aligned} \mathbf{L}_{00}\mathbf{L}_{00}^T &= \mathbf{A}_{00}, \\ \mathbf{L}_{00}\mathbf{l}_{01} &= \mathbf{a}_{01}, \\ \mathbf{l}_{01}^T\mathbf{l}_{01} + l_{11}^2 &= a_{11}. \end{aligned} \tag{A.6}$$

Notice that the first equation is simply an $(n-1)$-sized Cholesky decomposition! We can apply the same technique again recursively to calculate $\mathbf{L}_{00}$, then perform a triangular solve to calculate $\mathbf{l}_{01}$, then finally do a dot product and square root[2] to calculate l_{11}. In the dense case, this is all we really require to construct an efficient algorithm. However, we are interested in exploiting the sparsity of $\mathbf{A}$. The practical consequence of this is that in the second equation $\mathbf{L}_{00}\mathbf{l}_{01} = \mathbf{a}_{01}$, all three terms are sparse. Solving this sparse triangular system (with sparse right hand side) is the major difficulty in the sparse Cholesky factorisation.

Sparse Triangular Solve

To efficiently solve triangular systems $\mathbf{L}\mathbf{x} = \mathbf{b}$ where $\mathbf{L}$, $\mathbf{x}$ and $\mathbf{b}$ are all sparse, we need to know the sparsity pattern of $\mathbf{x}$. We can represent this pattern with

[2]Note that $a_{11} - \mathbf{l}_{01}^T\mathbf{l}_{01}$ is guaranteed to be positive (and the square root guaranteed to exist) as long as the matrix is positive definite. In fact, attempting to compute the Cholesky factorisation is one of the most efficient ways to test if a matrix is indeed positive definite.

the set $\boldsymbol{\chi}$, which contains the indices of the nonzero entries of $\mathbf{x}$. As long as $\boldsymbol{\chi}$'s entries are sorted in ascending order, we can compute $\mathbf{x}$ with Algorithm 4. For

Algorithm 4 Sparse Triangular Solve [Davis 2006]

$\mathbf{x} \leftarrow \mathbf{b}$

for all $j \in \boldsymbol{\chi}$ **do**

  for all $i > j$ for which $L_{ij} \neq 0$ **do**

    $x_i \leftarrow x_i - L_{ij} x_j$

  end for

end for

the systems obtained when doing sparse Cholesky factorisation we can compute the $\boldsymbol{\chi}$ of each $\mathbf{l}_{01}$ from a structure called the *elimination tree* of $\mathbf{A}$. Computing this tree requires some graph theory, discussed compactly in [Davis 2006].

A.2.2 Fill Reducing Ordering

We expect that the Cholesky factor $\mathbf{L}$ of a matrix $\mathbf{A}$ will have some *fill-in*, which are entries introduced by the factorisation that were not present in $\mathbf{A}$. Naturally, we would like to minimize the amount of fill-in to make computing and working with the Cholesky factor more efficient. It turns out that the amount of fill-in is determined by the *ordering* of the system. We are free to order the vector and matrix representing our system any way we wish without changing the answer, but the choice of ordering changes the elimination tree and can introduce more or less fill-in. Thus an important step of sparse Cholesky decomposition is computing a *fill reducing ordering*. This is a hard combinatorial optimization problem which we will not delve into here. Fortunately, there is favourably licensed software such as METIS [Karypis and Kumar 1998] available to compute approximately-optimal orderings.

Bibliography

Anitescu, Mihai and F. A. Potra (2001). "A Time-Stepping Method for Stiff Multibody Dynamics with Contact and Friction". In: *International Journal for Numerical Methods in Engineering*.

Baraff, David and Andrew Witkin (1998). "Large steps in cloth simulation". In: *ACM SIGGRAPH 1998 papers*. SIGGRAPH '98. New York, NY, USA: ACM, pp. 43–54. ISBN: 0-89791-999-8.

Bargteil, Adam W. and Elaine Cohen (2014). "Animation of Deformable Bodies with Quadratic Bezier Finite Elements". In: *ACM Trans. Graph.* 33.3, 27:1–27:10. ISSN: 0730-0301.

Belytschko, Ted and Lee P. Bindeman (1993). "Assumed strain stabilization of the eight node hexahedral element". In: *Computer Methods in Applied Mechanics and Engineering* 105.2, pp. 225 –260. ISSN: 0045-7825.

Belytschko, Ted, W.K. Liu, and B. Moran (2000). *Nonlinear Finite Elements for Continua and Structures*. Wiley. ISBN: 9780471987734.

Bender, Jan and Alfred Schmitt (2006). "Fast Dynamic Simulation of Multi-Body Systems Using Impulses". In: *Virtual Reality Interactions and Physical Simulations (VRIPhys)*. Madrid (Spain), pp. 81–90.

Bender, Jan et al. (2012). "Interactive Simulation of Rigid Body Dynamics in Computer Graphics". In: *EUROGRAPHICS 2012 State of the Art Reports*. Cagliari, Sardinia, Italy: Eurographics Association.

Bergou, Miklós et al. (2008). "Discrete Elastic Rods". In: *ACM SIGGRAPH 2008 papers*. Vol. 27. 3. New York, NY, USA: ACM, 63:1–63:12.

Bertails, Florence et al. (2006). "Super-Helices for Predicting the Dynamics of Natural Hair". In: *ACM SIGGRAPH 2006 papers*. New York, NY, USA: ACM.

Boggs, Paul T. and Jon W. Tolle (1995). *Sequential Quadratic Programming*.

Bouaziz, Sofien et al. (2014). "Projective Dynamics: Fusing Constraint Projections for Fast Simulation". In: *ACM Trans. Graph.* 33.4, 154:1–154:11. ISSN: 0730-0301.

Catmull, E. and J. Clark (1978). "Recursively generated B-spline surfaces on arbitrary topological meshes". In: *Computer-Aided Design* 10.6, pp. 350 –355. ISSN: 0010-4485.

Catto, Erin (2014). *Box2D*.

Chao, Isaac et al. (2010). "A Simple Geometric Model for Elastic Deformations". In: *ACM SIGGRAPH 2010 Papers*. SIGGRAPH 2010. Los Angeles, California: ACM, 38:1–38:6. ISBN: 978-1-4503-0210-4.

Cirak, Fehmi, Michael Ortiz, and Peter Schröder (2000). "Subdivision surfaces: a new paradigm for thin-shell finite-element analysis". In: *International Journal for Numerical Methods in Engineering* 47.12, pp. 2039–2072. ISSN: 1097-0207.

Coumans, Erwin (2005). *Bullet Physics*. URL: `http://bulletphysics.org/`.

— (2014). "Exploring MLCP solvers and Featherstone". In: *Game Developers' Conference 2014*.

Davis, T. (2006). *Direct Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics.

Engell-Norregard, Morten, Sarah Niebe, and Kenny Erleben (2012). "A joint-constraint model for human joints using signed distance-fields". English. In: *Multibody System Dynamics* 28 (1-2), pp. 69–81. ISSN: 1384-5640.

Erleben, Kenny and Ricardo Ortiz (2008). "A non-smooth newton method for multibody dynamics". In: *Numerical Analysis and Applied Mathematics*. Ed. by Theodore E Simos, George Psihoyios, and Ch. Tsitouras. AIP Conference Proceedings. Springer, pp. 178–181. ISBN: 9780735405769.

Hadap, Sunil and Nadia Magnenat-Thalmann (2001). "Modeling Dynamic Hair as a Continuum". In: *Computer Graphics Forum* 20.3, pp. 329–338. ISSN: 1467-8659.

Hauth, Michael and Olaf Etzmuss (2001). "A High Performance Solver for the Animation of Deformable Objects using Advanced Numerical Methods". In: *Computer Graphics Forum* 20.3, pp. 319–328. ISSN: 1467-8659.

Hecht, Florian et al. (2012). "Updated Sparse Cholesky Factors for Corotational Elastodynamics". In: *ACM Trans. Graph.* 31.5, 123:1–123:13. ISSN: 0730-0301.

Irving, G., J. Teran, and R. Fedkiw (2004). "Invertible Finite Elements for Robust Simulation of Large Deformation". In: *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA 2004. Grenoble, France: Eurographics Association, pp. 131–140. ISBN: 3-905673-14-2.

Jakobsen, Thomas (2001). "Advanced Character Physics". In: *Game Developer's Conference 2001*.

Kaldor, Jonathan M., Doug L. James, and Steve Marschner (2008). "Simulating knitted cloth at the yarn level". In: *ACM SIGGRAPH 2008 papers*. SIGGRAPH '08. Los Angeles, California: ACM, 65:1–65:9. ISBN: 978-1-4503-0112-1.

Karypis, George and Vipin Kumar (1998). "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs". In: *SIAM J. Sci. Comput.* 20.1, pp. 359–392. ISSN: 1064-8275.

Kim, Tae-Yong, Nuttapong Chentanez, and Matthias Müller-Fischer (2012). "Long range attachments - a method to simulate inextensible clothing in computer games". In: *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '12. Lausanne, Switzerland: Eurographics Association, pp. 305–310. ISBN: 978-1-4502-1909-6.

Langer, Joel and David A. Singer (1996). "Lagrangian Aspects of the Kirchhoff Elastic Rod". In: *SIAM Rev.* 38.4, pp. 605–618. ISSN: 0036-1445.

Lee, Sung-Hee and Demetri Terzopoulos (2008). "Spline joints for multibody dynamics". In: *ACM SIGGRAPH 2008 papers*. SIGGRAPH '08. Los Angeles, California: ACM, 22:1–22:8. ISBN: 978-1-4503-0112-1.

Lewin, Chris et al. (2013). "Rod Constraints for Simplified Ragdolls". In: *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '13. Anaheim, California: ACM, pp. 79–84. ISBN: 978-1-4503-2132-7.

McAdams, Aleka et al. (2011). "Efficient elasticity for character skinning with contact and collisions". In: *ACM SIGGRAPH 2011 papers*. SIGGRAPH '11.

Vancouver, British Columbia, Canada: ACM, 37:1–37:12. ISBN: 978-1-4503-0943-1.

Michels, Dominik L., Gerrit Sobottka, and Andreas Weber (2014). "Exponential Integrators for Stiff Elastodynamic Problems". In: *ACM Trans. Graph.* 33.1.

Müller, Matthias and Nuttapong Chentanez (2010). "Wrinkle Meshes". In: *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '10. Madrid, Spain: Eurographics Association, pp. 85–92.

Müller, Matthias and Markus Gross (2004). "Interactive Virtual Materials". In: *Proceedings of Graphics Interface 2004*. GI 2004. London, Ontario, Canada: Canadian Human-Computer Communications Society, pp. 239–246. ISBN: 1-56881-227-2.

Müller, Matthias et al. (2005). "Meshless Deformations Based on Shape Matching". In: *ACM SIGGRAPH 2005 Papers*. SIGGRAPH 2005. Los Angeles, California: ACM, pp. 471–478.

Müller, Matthias et al. (2007). "Position based dynamics". In: *J. Vis. Comun. Image Represent.* 18.2, pp. 109–118. ISSN: 1047-3203.

Nocedal, J. and S. Wright (2006). *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer New York. ISBN: 9780387303031.

Nocent, O. and Y. Remion (2001). "Continuous deformation energy for dynamic material splines subject to finite displacements". In: *Proceedings of the Eurographic workshop on Computer animation and simulation*. Manchester, UK: Springer-Verlag New York, Inc., pp. 88–97. ISBN: 3-211-83711-6.

Parker, Eric G. and James F. O'Brien (2009). "Real-time Deformation and Fracture in a Game Environment". In: *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '09. New Orleans, Louisiana: ACM, pp. 165–175. ISBN: 978-1-60558-610-6.

Press, William H. et al. (2007). *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. 3rd ed. New York, NY, USA: Cambridge University Press. ISBN: 0521880688, 9780521880688.

Remion, Yannick, Jean-Michel Nourrit, and Didier Gillard (1999). "Dynamic Animation Of Spline Like Objects". In: *Proc. WSCG'99*. Ed. by V. Skala.

Rivers, Alec R. and Doug L. James (2007). "FastLSM: fast lattice shape matching for robust real-time deformation". In: *ACM SIGGRAPH 2007 papers*. SIGGRAPH '07. San Diego, California: ACM.

Rungjiratananon, Witawat et al. (2012). "Animating strings with twisting, tearing and flicking effects". In: *Comput. Animat. Virtual Worlds* 23.2, pp. 113–124. ISSN: 1546-4261.

Selle, Andrew, Michael Lentine, and Ronald Fedkiw (2008). "A mass spring model for hair simulation". In: *ACM SIGGRAPH 2008 papers*. SIGGRAPH '08. Los Angeles, California: ACM, 64:1–64:11. ISBN: 978-1-4503-0112-1.

Sifakis, E. and J. Barbič (2012). "FEM Simulation of 3D Deformable Solids: A practitioner's guide to theory, discretization and model reduction". In: *SIGGRAPH 2012 Courses*.

Silcowitz-Hansen, Morten, Sarah Niebe, and Kenny Erleben (2009). "Nonsmooth Newton Method for Fischer Function Reformulation of Contact Force Problems for Interactive Rigid Body Simulation". In: *VRIPHYS'09*, pp. 105–114.

Stam, J. (2009). "Nucleus: Towards a unified dynamics solver for computer graphics". In: *Computer-Aided Design and Computer Graphics, 2009. CAD/Graphics '09. 11th IEEE International Conference on*, pp. 1–11.

Stewart, David and J. C. Trinkle (1996). "An Implicit Time-Stepping Scheme for Rigid Body Dynamics with Coulomb Friction". In: *International Journal for Numerical Methods in Engineering* 39, pp. 2673–2691.

Su, J., R. Sheth, and R. Fedkiw (2013). "Energy Conservation for the Simulation of Deformable Bodies". In: *Visualization and Computer Graphics, IEEE Transactions on* 19.2, pp. 189–200. ISSN: 1077-2626.

Theetten, A. et al. (2008). "Geometrically exact dynamic splines". In: *Comput. Aided Des.* 40.1, pp. 35–48. ISSN: 0010-4485.

Tonge, Richard, Feodor Benevolenski, and Andrey Voroshilov (2012). "Mass splitting for jitter-free parallel rigid body simulation". In: *ACM Trans. Graph.* 31.4, 105:1–105:8. ISSN: 0730-0301.

Ward, Kelly et al. (2007). "A Survey on Hair Modeling: Styling, Simulation, and Rendering". In: *IEEE Transactions on Visualization and Computer Graphics* 13.2, pp. 213–234. ISSN: 1077-2626.

Witkin, Andrew, Michael Gleicher, and William Welch (1990). "Interactive Dynamics". In: *Proceedings of the 1990 Symposium on Interactive 3D Graphics*. I3D '90. Snowbird, Utah, USA: ACM, pp. 11–21. ISBN: 0-89791-351-5.

Yin, KangKang, Kevin Loken, and Michiel van de Panne (2007). "SIMBICON: Simple Biped Locomotion Control". In: *ACM SIGGRAPH 2007 papers*. Vol. 26. SIGGRAPH '07 3.

Zhu, Yongning et al. (2010). "An Efficient Multigrid Method for the Simulation of High-resolution Elastic Solids". In: *ACM Trans. Graph.* 29.2, 16:1–16:18. ISSN: 0730-0301.