Rodrigo Castiel Reis de Souza

A REDUCED QUASI-NEWTON SOLVER FOR REAL-TIME SIMULATION OF HYPERELASTIC MATERIALS

B.Sc. Thesis

Federal University of Pernambuco
secgrad@cin.ufpe.br
www.cin.ufpe.br/~secgrad

Recife

November, 2017

Federal University of Pernambuco
Center for Informatics
Bachelor's in Computer Engineering

Rodrigo Castiel Reis de Souza

A REDUCED QUASI-NEWTON SOLVER FOR REAL-TIME SIMULATION OF HYPERELASTIC MATERIALS

A B.Sc. Thesis presented to the Center for Informatics of Federal University of Pernambuco in partial fulfillment of the requirements for the degree of Bachelor in Computer Engineering.

Advisors: *Jernej Barbic, Tsang Ing Ren*

Recife
November, 2017

Rodrigo Castiel Reis de Souza
A Reduced Quasi-Newton Solver for Real-time Simulation of Hyperelastic Materials/ Rodrigo Castiel Reis de Souza. – Recife, November, 2017-
46 p. : il. (algumas color.) ; 30 cm.

Advisors Jernej Barbic, Tsang Ing Ren

B.Sc. Thesis – Universidade Federal de Pernambuco, November, 2017.

1. Palavra-chave1. 2. Palavra-chave2. I. Orientador. II. Universidade xxx. III. Faculdade de xxx. IV. Título

CDU 02:141:005.7

Le grand finale.

A intuição é uma das maiores facetas humanas. Você intui,
e mesmo que não seja do nada, não se sabe de onde...
mas que intui, intui.

—CLYLTON GALAMBA

Abstract

Physics-based animation is an area of computer graphics that is concerned with generating realistic animation from the laws of dynamics. Typical applications are computer games, visual effects, animated movies and simulators; some of them must be interactive and real-time, which limits the computation budget to a few milliseconds. In this work, we present a new method for real-time simulation of deformable objects in reduced space. Recent methods such as the quasi-Newton solver presented by Liu *et al.* [13] are robust, general and simple, but can be expensive when dealing with large meshes. In order to fulfill high frame-rate requirements, we apply dimensionality reduction and optimal cubature to the standard quasi-Newton solver. As result, our implementation in CPU achieves a speed-up of hundred times, while preserving overall realistic animation qualities. Additionally, we provide a brief introduction to deformable object simulation and compare the classical implicit backward Euler method to the quasi-Newton solvers.

Keywords: Physics-based Animation, Deformable Object Simulation, Finite Element Method, Computer Graphics

List of Figures

2.1 System S with n particles. Each particle has a position $\boldsymbol{x}_i(t)$ and velocity $\boldsymbol{v}_i(t)$. They may interact with each other through internal forces that are position-dependent. 14

2.2 Motion of a bouncing elastic ball. The material point $\boldsymbol{X} \in \Omega$ is displaced to locations $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ at t_1 and t_2, respectively. (a) Reference configuration (undeformed shape). (b) Image of a deformation that horizontally stretches the object due to collision forces. (c) Image of a deformation that vertically stretches the object as result of oscillation caused by the collision. 18

2.3 Meshes for the classic bunny model. On the left, a surface triangulation typically used for rendering. On the right, a detailed tetrahedal mesh for discretizing the enclosed domain in FEM-based techniques. JIGSAW (2017). An unstructured mesh generator. [image] Available at: https://sites.google.com/site/dengwirda/jigsaw [Accessed 28 Oct. 2017]. . 21

2.4 Deformation of tetrahedron E_j. (a) Rest shape. (b) Deformed shape. . . . 22

5.1 Tetrahedral meshes at reference configuration. On the top-left corner, the *Letter A* model (small). On the top-right corner, the *Dragon* model (large). On the bottom, the *Bridge* model (medium size). 35

5.2 *Letter A* model, Experiment 1 (StVK), frame 155 (time-step $h = 10^{-3}s$). From the left to the right, the images depict the ground-truth, the implicit Euler, the full and reduced quasi-Newton, respectively. The ground-truth output animation (green) is more vivid and more locally detailed. However, we notice that the classical implicit integrator (in blue) becomes unstable (explosion). Furthermore, the motion of the quasi-Newton methods is similar, but with a higher damping and smaller deformation amplitudes. . 37

5.3 *Dragon* model, Experiment 1 (StVK), frames 189, 196, 204 (time-step $h = \frac{1}{30}s$). This vertical sequence of images shows how the full solver (in black, on the left) and the reduced solver (in red, on the right) react to a long impulse on the dragon's head. The animations are similar, but our reduced solver (with small cubature) runs about 402 times faster than the standard quasi-Newton. 38

5.4 Plot of the total kinect energy under constant upwards pulling forces on the bridge mesh (Experiment 3, frames 82-228). Curves are ordered from the largest energy (top) to the smallest (bottom). The oscillation in both quasi-Newton methods loses the amplitude more quickly than in the implicit backward Euler. The output motion of the reduced solver is similar to the full solver, but the oscillation frequency is lower. 39
5.5 Plot of kinetic energy peaks in *Letter A* model experiments. In Experiment 1, the tall peak of energy in blue around frame 150 occurs due to the explosion in the implicit Euler animation. 40
5.6 Plot of kinetic energy peaks in *Dragon* model experiments. 41
5.7 Plot of kinetic energy peaks in *Bridge* model experiments. Notice how the ground-truth motion is energetic. Though all methods do not approximate it well, they have close curves. 42

List of Acronyms

Contents

1 Introduction **9**

1.1 Physics-based Animation . . . 9
1.2 Deformable Object Simulation . . . 10
1.3 Proposal . . . 11
1.4 Structure of the Thesis . . . 12

2 Background **13**

2.1 General Particle Systems . . . 13
a. Formulation . . . 13
b. Integration . . . 15
2.2 Continuum Mechanics . . . 17
a. Deformation Map and Strain Energy . . . 17
b. Strain & Stress . . . 19
c. Materials . . . 20
2.3 Finite Element Method . . . 21
a. Volumetric Meshes . . . 21
b. Elastic Forces . . . 23
c. Governing Equations . . . 23
2.4 Model Reduction . . . 24

3 Related Work **25**

3.1 Classical Methods . . . 25
3.2 Projective Methods . . . 26
3.3 Quasi-Newton Solver . . . 26

4 Reduced Quasi-Newton Solver **29**

4.1 Derivation . . . 29
4.2 Optimal Cubature . . . 31
4.3 Algorithm . . . 32

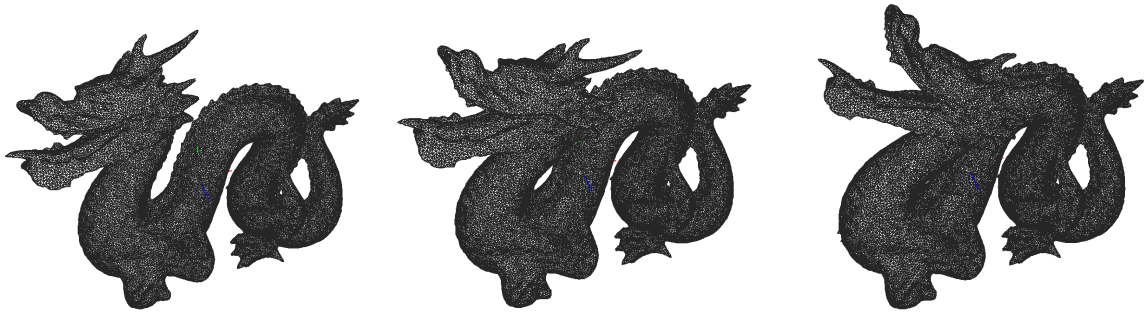
5 Experiments and Results **34**

5.1 Experiments . . . 34
5.2 Performance vs. Animation Quality . . . 35
5.3 Numerical Damping Analysis . . . 37

6 Conclusion **43**
6.1 Limitations 43
6.2 Future Work 44

References **45**

Chapter 1
Introduction



Sample frames of a deformable dragon model undergoing sudden pulling forces. The simulation is performed in real-time using the proposed method.

In this work, we present a novel technique for real-time simulation of hyperelastic materials. Before proceeding to further details, we shall first present an overview of physics simulation for computer animation, also known as *physics-based animation*, and briefly explain how its goals differ from classical scientific computing methods. Then, we shall introduce the branch of *deformable object simulation* and examine its recent works and applications.

1.1 Physics-based Animation

Physics simulation is the progressive numerical computation of physical quantities that fully describe a system over time, given their initial values, namely *boundary conditions*. The output of a simulation is a discrete-time sequence of state variables – frames – sampled at a constant rate. Phenomena are commonly modeled through differential equations whose solution is usually impossible to be calculated analytically, but certainly achievable by approximate simulations. The term *time-step* refers to the process of producing a new frame from previous computed data, but is often used as the duration Δt between two consecutive frames.

Historically, several simulation methods for computational dynamics of solids and fluids have been widely deployed in industry as a consequence of the well-established scientific

computing research in applied mechanics and mechanical engineering. One of the most widespread techniques is the *Finite Element Method* (FEM) for structural analysis, which calculates the accurate response of forces acting upon complex structures, such as a bridge loaded with cars or bent airplane wings during a take-off. When it comes to computer graphics, the research on physics-based animation is still increasing due to the recent demand of computer games, animated movies, visual effects and diverse simulators (*e.g.*, surgery and flight simulators). In this research field, most of techinques are derived from scientific computing methods, such as FEM, but have different constraints and goals.

Scientific computing applications primarily focus on the reproduction of phenomena with high accuracy rather than robustness, interactivity and performance [9]. In most cases, users may change parameters, such as the time-step, and perform the simulation until they achieve the desired stable results. Although fast simulations are valuable, the cost of a solution is not as important as its quality. On the other hand, physics-based animation is mainly concerned with visually appealing motion, interactivity, stability and freedom of modeling regardless of realism [9]. Moreover, in real-time applications, the simulation must be fast to fullfill high frame-rate requirements, and robust to avoid unexpected artifacts (resulting from numerical instability). For example, a typical rate of 60 Hz in games leaves about 16 ms to the overall frame computation, including rendering and physics. Thus, less than 10 ms remain to perform collision detection and dynamics update.

Physics-based animation methods are frequently classified into two main categories: *fluid dynamics* and *solid dynamics* [17]. Within solid dynamics, we may group objects as follows:

- *Rigid-bodies*, whose shape is preserved throughout the motion, meaning that only rotations and translations are allowed;
- *Cloth*, which are approximated to surfaces that have low reaction to bending and torsion, but high resistance to stretching and shearing;
- *Deformable objects*, general elastic bodies whose shape is disturbed under stimuli.

As mentioned, we focus on *deformable objects*.

1.2 Deformable Object Simulation

A deformable object in three dimensions is often modeled by some continuous domain contained in $\mathbb{R}^3$. During a deformation, each point of its domain is mapped to an arbitrary point and, as result, forces act to minimize the so-called *strain energy*. The relationship between stimuli (deformation) and response (forces and strain energy) is defined by its elastic material model. Materials whose strain energy depends only on the current

deformation are said to be *hyperelastic*. This class of materials is broadly investigated because it is able to model a large variety of real-world elastic properties.

Computer simulations require that we choose both a discrete representation for the deformable object domain and an elastic force model. Thus, we can apply Newton's laws to derive the governing differential equations of motion and use numerical integration to find the solution. Perhaps the simplest approach for 3D-elasticity may be *mass-spring systems*, where the domain is modeled as a group of particles connected by springs. However, this model does not always achieve the desired response because its behavior depends strongly on how the mesh is built [17]. For this reason, advanced approaches such as FEM are based on the field of *continuum mechanics*, which provides highly accurate models for elasticity.

Nonlinear FEM-based techniques are largely explored in computer graphics. In [23], the authors presented *VegaFEM*, an open-source library that implements several nonlinear material models and implicit integration methods. In practice, although all these classical methods achieve high quality animation, they may be overly expensive for the available computation time. The main reasons are: complex structures require detailed meshes (that might have thousands of vertices); to guarantee stability, integrators must be implicit, which requires solving high-order nonlinear systems at each time-step. Because of that, many applications use alternative approaches such as *Position-Based Dynamics* (PBD) [16].

PBD has been successfully implemented in game engines (*e.g.* PhysX) because of its general constraint-based formulation. Yet it lacks the support for more advanced elastic material models. Recently, based on PBD, Bouaziz *et al.* [8] proposed *projective dynamics*, a novel real-time solver that uses a continuum mechanics approach to deal with basic elastic models. Afterwards, Liu *et al.* [13] interpreted projective dynamics as a quasi-Newton method and improved its convergence through *line search* and *L-BFGS update* [12]. This extended the support to general material models, such as the St. Venant-Kirchhoff, Neo-Hookean, Mooney-Rivlin and the *spline-based* [26].

We implemented both projective dynamics with FEM potentials and the quasi-Newton solver described above; we noticed that they are in fact robust, general and simple. But due to the approximate energy gradients, their convergence is usually slow and, as consequence, the output animation becomes damped and not as accurate as the original implicit backward Euler integrator. Furthermore, their strain energy must be evaluted at each solver iteration, which requires performing either polar or singular-value decomposition of all tetrahedron deformation gradients.

1.3 Proposal

Inspired by dimensionality reduction, we propose a *reduced quasi-Newton solver*, a new technique to accelerate the simulation of large meshes of hyperelastic materials. In this work, we build a scheme that applies the concepts of model reduction [15] and optimal cubature [1] to the quasi-Newton solver of projective dynamics under hyperelastic constraints. Model reduction minimizes the time-step cost because it projects the state of an object and its differential equation from the original high-dimensional space onto a representative low-dimensional subspace. Optimal cubature increases the efficiency even more by selecting a subset of the mesh elements whose projected reduced internal forces best approximate the overall reduced force. Thus, all optimization steps become simpler and easier to compute, with the trade-off of decreasing the animation quality.

Unlike methods such as the reduced implicit backward Euler and the reduced Newark, our technique does not require computing specific reduced force models or stiffness matrices. As it follows the quasi-Newton solver, only the energy function must be implemented according to the material model. Therefore, our method speeds up the full solver while preserving its simplicity and robustness. Although we do not investigate methods for basis generation or cubature estimation, the quality of the proposed method is determined by the precomputed basis and cubature.

1.4 Structure of the Thesis

This document is divided as follows. Chapter 2 introduces the underlying concepts of physics-based animation, focused on deformable object simulation. Chapter 3 reviews recent works in real-time simulation of hyperelastic materials, including classical and projective methods; it also explains the standard quasi-Newton method [13] on which the proposed method is built. In Chapter 4, we derive the proposed solver and provide a high-level algorithm for it. Chapter 5 reports the experiments and analysis of the proposed solver against the original quasi-Newton and the implicit backward Euler. In Chapter 6, we briefly present the benefits and limitations of our method, together with potential improvements and future work.

Chapter 2
Background

This chapter provides some of the basic concepts of real-time simulation of deformable objects. It is intended to be only a starting point for those who have not been introduced to this area; if you want to go further please consider studying the referenced works. We start by formulating a general model for simulating particle systems. Then, we present an overview of the elasticity theory from a continuum mechanics perspective. Afterwards, we explain how to apply the Finite Element Method (FEM) to continuum mechanics models to obtain a discrete system that we can integrate. Last, we describe the basic idea of how model reduction can be successfully used to accelerate the time-step computation.

2.1 General Particle Systems

a. Formulation

Let S be a system of n particles in 3D space (Figure 2.1). Each particle P_i has a mass m_i and, at a given moment t, a position $\boldsymbol{x}_i(t) \in \mathbb{R}^3$ and velocity $\boldsymbol{v}_i(t) \in \mathbb{R}^3$. Between any two particles P_i and P_j, there might be an internal force $\boldsymbol{f}_{i,j} \in \mathbb{R}^3$, such as elastic, gravitational or even magnetic. Generally, internal forces are functions of the particle positions and some of their derivatives. For example, while damping forces depend only on the particle velocities, electric forces depend only on the positions. We model external forces (*e.g.*, user interaction forces) acting on P_i using a single term $\boldsymbol{f}_{ext,i}$ which is constant within the time-step. Thus, the resulting force $\boldsymbol{f}_{\text{total},i}$ on P_i is:

$$\boldsymbol{f}_{\text{total},i} = \boldsymbol{f}_{ext,i} + \sum_{j \neq i} \boldsymbol{f}_{ij}$$

Notice that $\boldsymbol{f}_{\text{total},i}$ is a function of the configuration of all particles with which P_i interacts. Let Δt be the time-step duration. For the sake of simplification, we will use $\alpha^k = \alpha(k\Delta t)$ as a variable α at the k-th time-step (in the context of simulation). Also, we will use the notation $\boldsymbol{x} = \boldsymbol{x}(t)$ and $\dot{\boldsymbol{x}} = \frac{d\boldsymbol{x}(t)}{dt}$ except when its meaning can not be implicitly inferred.

In order to represent the current configuration of S, we concatenate the positions of all

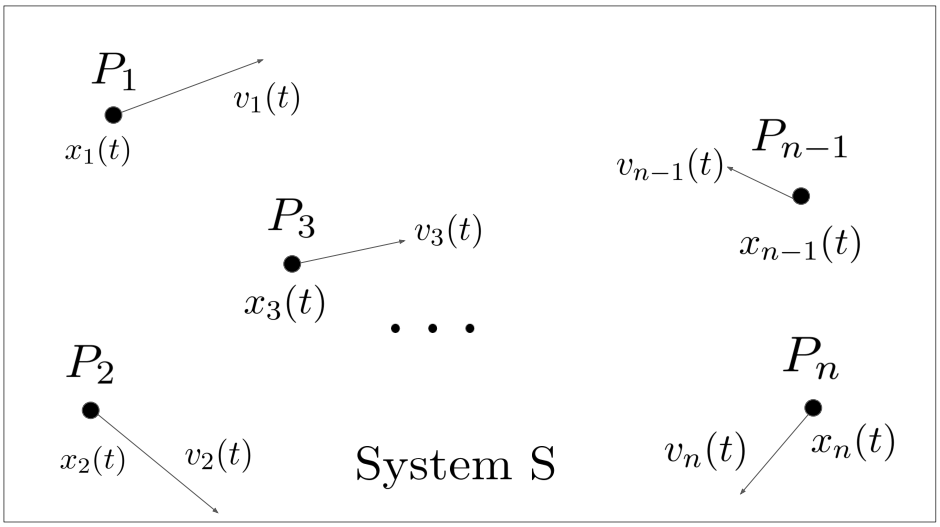


Figure 2.1: System S with n particles. Each particle has a position $\boldsymbol{x}_i(t)$ and velocity $\boldsymbol{v}_i(t)$. They may interact with each other through internal forces that are position-dependent.

particles vertically into a single vector $\boldsymbol{x} = (\boldsymbol{x}_0^T, \boldsymbol{x}_1^T, ... \, \boldsymbol{x}_{n-1}^T)^T \in \mathbb{R}^{3n}$. Similarly, the system velocities are $\boldsymbol{v} = (\boldsymbol{v}_0^T, \boldsymbol{v}_1^T, ... \, \boldsymbol{v}_{n-1}^T)^T \in \mathbb{R}^{3n}$. Considering that the mass of each particle is totally concentrated in its position, we can represent the system mass by a diagonal matrix $M = \text{diag}[m_0, m_1, ... \, m_{n-1}] \in \mathbb{R}^{3n \times 3n}$. Since hyperelastic forces are conservative, we assume that the internal forces can be calculated from the positions alone. Then, the total system forces become:

$$\boldsymbol{f_{total}} = \begin{bmatrix} \boldsymbol{f}_{\text{total},0} \\ \boldsymbol{f}_{\text{total},1} \\ \vdots \\ \boldsymbol{f}_{\text{total},n-1} \end{bmatrix} = \boldsymbol{f_{int}}(\boldsymbol{x}) + \boldsymbol{f_{ext}}(t)$$

Once the forces are defined, we apply Newton's Second Law to find the differential equation that describes the motion:

$$M\ddot{\boldsymbol{x}} = \boldsymbol{f_{int}}(\boldsymbol{x}) + \boldsymbol{f_{ext}}(t) \tag{2.1}$$

As most of forces do not depend linearly on the positions, this second-order Ordinary Differential Equation (ODE) is often nonlinear as well. To make numerical integration easier, we decouple it into two first-order ODEs:

$$\begin{cases} \dot{\boldsymbol{v}} = M^{-1} \left[\boldsymbol{f_{int}}(\boldsymbol{x}) + \boldsymbol{f_{ext}}(t) \right] \\ \dot{\boldsymbol{x}} = \boldsymbol{v} \end{cases} \tag{2.2}$$

The system is uniquely described by the positions and velocities, so we can write the state

variable as $\boldsymbol{z} = (\boldsymbol{v}^T,\ \boldsymbol{x}^T)$ and rewrite the governing equation as:

$$\dot{\boldsymbol{z}} = \boldsymbol{G}(\boldsymbol{z}) \tag{2.3}$$

b. Integration

Given the boundary conditions, that is, the state variable $\boldsymbol{z}_0 = \boldsymbol{z}(0)$, the solution of *Eq.* 2.3 gives us the system configuration over time. However, *Eq.* 2.3 is unlikely to have an analytical solution; let us see how to use numerical integration methods to find its discrete-time approximation. First, the exact solution is:

$$\boldsymbol{z}(t) = \boldsymbol{z}_0 + \int_0^t \dot{\boldsymbol{z}}(\tau)d\tau \tag{2.4}$$

whose integral term may not exist. As the simulation is performed, the states $\boldsymbol{z}(t)$ are output at each time-step Δt. Thus, we may rewrite the *Eq.* 2.4 in a recursive form:

$$\boldsymbol{z}(t+\Delta t) = \underbrace{\boldsymbol{z}(t)}_{\text{previous state}} + \underbrace{\int_t^{t+\Delta t} \dot{\boldsymbol{z}}(\tau)d\tau}_{\text{term to be approximated}} \tag{2.5}$$

There are several methods for approximating the above integral. *Explicit integrators* extrapolate the function $\boldsymbol{z}(t)$ within the interval $[t, t+\Delta t]$ using the states of previous frames only. *Implicit integrators* do not extrapolate $\boldsymbol{z}(t)$; instead, they build an approximatation scheme that leads to a nonlinear equation whose unknown is $\boldsymbol{z}(t+\Delta t)$. Then, using a numerical method, we solve for $\boldsymbol{z}(t+\Delta t)$. Examples of explicit methods are *explicit Euler*, *Runge-Kutta* and *Verlet integration*. Some of the well-known implicit methods are *implicit Euler* and *Newmark integration*. For more information about them, please refer to [9, 17, 4]. We will now look into the Euler's methods, the simplest integrators.

Explicit Euler's method. The basic idea is to approximate the time-derivative of $\boldsymbol{z}(\tau)$ to be constant within the integration interval and equal to the previous computed derivative. That is, $\dot{\boldsymbol{z}}(\tau) \approx \dot{\boldsymbol{z}}(t)$ for $\tau \in [t, t+\Delta t]$. Thus, applying into *Eq.* 2.5, we obtain:

$$\boldsymbol{z}(t+\Delta t) \approx \boldsymbol{z}(t) + \int_t^{t+\Delta t} \dot{\boldsymbol{z}}(t)d\tau = \boldsymbol{z}(t) + \Delta t\ \underbrace{\dot{\boldsymbol{z}}(t)}_{\boldsymbol{G}(\boldsymbol{z}(t))} \tag{2.6}$$

that can be written again with positions and velocities:

$$\begin{cases} \boldsymbol{v}(t+\Delta t) = \boldsymbol{v}(t) + \Delta t M^{-1}\left[\boldsymbol{f_{int}}(\boldsymbol{x}(t)) + \boldsymbol{f_{ext}}(t)\right] \\ \boldsymbol{x}(t+\Delta t) = \boldsymbol{x}(t) + \Delta t \boldsymbol{v}(t) \end{cases}$$

$$\Downarrow$$

$$\begin{cases} \boldsymbol{v}^{k+1} = \boldsymbol{v}^k + \Delta t M^{-1}\left[\boldsymbol{f_{int}}(\boldsymbol{x}^k) + \boldsymbol{f}^k_{\boldsymbol{ext}}\right] \\ \boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \Delta t \boldsymbol{v}^k \end{cases} \tag{2.7}$$

It is a simple direct computation from the previous frame state $\boldsymbol{v}^k, \boldsymbol{x}^k$.

Implicit Euler's method. It is very similar to the explicit Euler's method, but avoids extrapolation by assuming that the derivative depends on the next time-step state: $\dot{\boldsymbol{z}}(\tau) \approx \dot{\boldsymbol{z}}(t + \Delta t)$ for $\tau \in [t, t + \Delta t]$. This leads to:

$$\boldsymbol{z}(t + \Delta t) \approx \boldsymbol{z}(t) + \Delta t \underbrace{\dot{\boldsymbol{z}}(t + \Delta t)}_{\boldsymbol{G}(\boldsymbol{z}(t+\Delta t))} = \boldsymbol{z}(t) + \Delta t \boldsymbol{G}(\boldsymbol{z}(t + \Delta t))$$

a nonlinear system where the unknown is $\boldsymbol{z}(t + \Delta t)$. Writing the equations separately, we get:

$$\begin{cases} \boldsymbol{v}^{k+1} = \boldsymbol{v}^k + \Delta t M^{-1}\left[\boldsymbol{f_{int}}(\boldsymbol{x}^{k+1}) + \boldsymbol{f}^k_{\boldsymbol{ext}}\right] \\ \boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \Delta t \boldsymbol{v}^{k+1} \end{cases} \tag{2.8}$$

We can apply the second equation into the first one to remove the dependency of $\boldsymbol{x}^{k+1}$:

$$\boldsymbol{v}^{k+1} = \boldsymbol{v}^k + \Delta t M^{-1}\left[\boldsymbol{f_{int}}(\boldsymbol{x}^k + \Delta t \boldsymbol{v}^{k+1}) + \boldsymbol{f}^k_{\boldsymbol{ext}}\right]$$

$$M\boldsymbol{v}^{k+1} - M\boldsymbol{v}^k - \Delta t\left[\boldsymbol{f_{int}}(\boldsymbol{x}^k + \Delta t \boldsymbol{v}^{k+1}) + \boldsymbol{f}^k_{\boldsymbol{ext}}\right] = \boldsymbol{0}$$

$$\boldsymbol{F}(\boldsymbol{v}^{k+1}) = \boldsymbol{0} \tag{2.9}$$

Since the internal forces are nonlinear functions of positions, we must use an iterative numerical method such as Newton-Raphson to find the zero of $\boldsymbol{F}$. To simplify the notation, let $\boldsymbol{v} = \boldsymbol{v}^{k+1}$ be the unknown. Given an initial guess $\boldsymbol{v}_0$ (which can be the previous frame velocities), the iterations of the Newton-Raphson follow:

$$\boldsymbol{v}_{l+1} = \boldsymbol{v}_l - [\nabla \boldsymbol{F}(\boldsymbol{v}_l)]^{-1} \boldsymbol{F}(\boldsymbol{v}_l) \tag{2.10}$$

where $\nabla \boldsymbol{F}(\boldsymbol{v}_l)$ is the Jacobian of $\boldsymbol{F}$ evaluated using the previous iteration velocities. Please notice that *previous iteration* refers to the Newton's method, not the previous time-step. *Eq.* 2.10 is equivalent to the linear system below:

$$\nabla \boldsymbol{F}(\boldsymbol{v}_l)(\boldsymbol{v}_{l+1} - \boldsymbol{v}_l) = \boldsymbol{F}(\boldsymbol{v}_l) \tag{2.11}$$

Taking the the gradient of $\boldsymbol{F}(\boldsymbol{v}_l)$:

$$\nabla \boldsymbol{F}(\boldsymbol{v}_l) = M - \Delta t^2 \underbrace{\nabla \boldsymbol{f}_{in}(\boldsymbol{x}^k + \Delta t \boldsymbol{v}_l)}_{K(\boldsymbol{v}_l)} \Rightarrow \nabla \boldsymbol{F}(\boldsymbol{v}_l) = M - \Delta t^2 K(\boldsymbol{v}_l) \tag{2.12}$$

We must compute the derivative $K(\boldsymbol{v}_l)$ of the internal forces, also known as *tangent stiffness*

matrix, to solve the linear system (*Eq.* 2.11) at every iteration. Even though $K(\boldsymbol{v}_l)$ and M are sparse matrices, solving this system multiple times per time-step is overly expensive. Because of that, running only one or two Newton-Raphson iterations may already achieve a good result depending on the time-step or other simulation parameters.

Although explicit methods are computationally cheap, they may not behavior correctly during high-frequency motion or under energetic user inputs (explosions). Therefore, the implicit methods are usually the best option for real-time dynamics, because even with the drawback of cost and numerical damping, they are more stable and do not accumulate energy [4].

2.2 Continuum Mechanics

So far, a general formulation for discrete systems was described but no specific force model was discussed. In this section, we analyze the continuous hyperelasticity theory in 3D to understand and use a more realistic force model for deformable objects. We follow the course notes presented by Sifakis and Barbic [22] and recommend it for further reading.

a. Deformation Map and Strain Energy

Suppose we want to simulate the dynamics of a deformable object. At its rest state, namely *reference configuration*, the object is represented by a continuous domain $\Omega \subset \mathbb{R}^3$. When it is deformed, each point $\boldsymbol{X}$ of its domain, called *material point*, is brought to another location. Thus, the *deformation* is modeled as a function $\boldsymbol{\phi} : \Omega \to \mathbb{R}^3$, known as *deformation map*, that assigns a new position $\boldsymbol{\phi}(\boldsymbol{X})$ to each domain point $\boldsymbol{X}$ (*i.e.*, a vector field over Ω). The deformation $\boldsymbol{\phi}$ is imposed by an external agent or just induced by previous elastic forces (*i.e.*, an object may be forced to change its shape). It is a mechanical stimulus that yields some response (forces). Perhaps an easy way to understand this theory is to see motion as a time-dependent vector field over the fixed domain Ω. At time t, an infinitesimal part of Ω is displaced to $\boldsymbol{\phi}(\boldsymbol{X}; t)$ (Figure 2.2).

The deformation map is able to handle all types of transformations, including rigid-body transformations. For example, $\boldsymbol{\phi}(\boldsymbol{X}) = \boldsymbol{X} + \boldsymbol{T}$ is a translation of all points by a displacement $\boldsymbol{T}$; $\boldsymbol{\phi}(\boldsymbol{X}) = s\boldsymbol{X}$ scales all points with respect to the origin by a factor s. The deformation map itself may not be a useful measure for calculating elastic forces, because in most of cases, such as a simple translation, its values are non-zero even when there is no shape change. Therefore, one important concept for developing elasticity models is the *deformation gradient*. The deformation gradient $F \in \mathbb{R}^{3\times3}$ is the second-order tensor

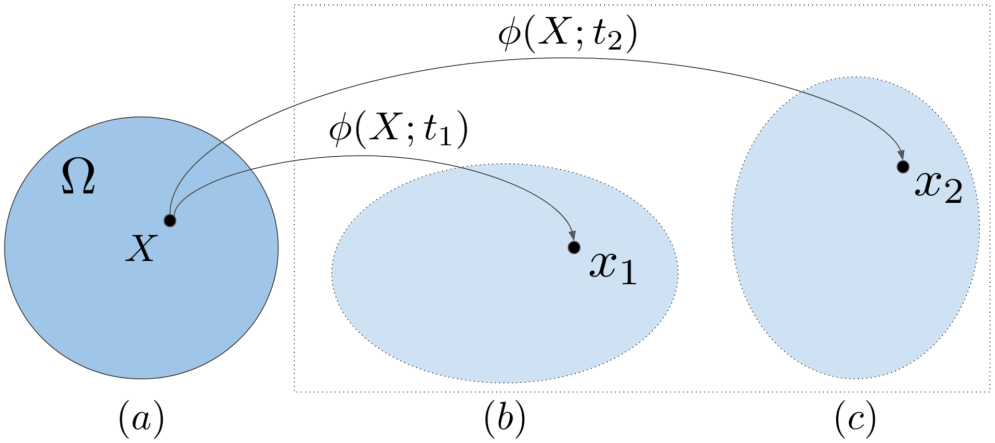


Figure 2.2: Motion of a bouncing elastic ball. The material point $\boldsymbol{X} \in \Omega$ is displaced to locations $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ at t_1 and t_2, respectively. (a) Reference configuration (undeformed shape). (b) Image of a deformation that horizontally stretches the object due to collision forces. (c) Image of a deformation that vertically stretches the object as result of oscillation caused by the collision.

derived from $\boldsymbol{\phi}$ as follows:

$$F(\boldsymbol{X}) = \boldsymbol{\nabla}\boldsymbol{\phi}(\boldsymbol{X}) = \begin{bmatrix} \frac{\partial \boldsymbol{\phi}_x}{\partial x} & \frac{\partial \boldsymbol{\phi}_x}{\partial y} & \frac{\partial \boldsymbol{\phi}_x}{\partial z} \\ \frac{\partial \boldsymbol{\phi}_y}{\partial x} & \frac{\partial \boldsymbol{\phi}_y}{\partial y} & \frac{\partial \boldsymbol{\phi}_y}{\partial z} \\ \frac{\partial \boldsymbol{\phi}_z}{\partial x} & \frac{\partial \boldsymbol{\phi}_z}{\partial y} & \frac{\partial \boldsymbol{\phi}_z}{\partial z} \end{bmatrix} \tag{2.13}$$

which is the Jacobian of $\boldsymbol{\phi}$. As $\boldsymbol{\phi}$, F may vary all over the domain Ω (*i.e.*, a tensor field). In spite of not being able to handle the effects of rotations directly, the deformation gradient is an underlying part of the projective dynamics-based methods that we will see in the next chapter.

Another fundamental concept is *strain energy*, the energy associated to an elastic deformation. In hyperelastic materials, the strain energy, here represented by g, depends only on the current configuration of the object, which means that it is a potential energy. It penalizes states that undergo intense shape change and is expected to be zero-valued when the object is at its reference configuration. Since every point of the domain is mapped to an arbitrary location, they may contribute to the total strain energy in different ways. The local contribution is modeled by the *energy density* Ψ, the energy per unit of volume. The energy density at a given point $X \in \Omega$ is invariant to translations, and is a function of the deformation gradient only [22]. We then calculate the total energy as:

$$g\{\boldsymbol{\phi}\} = \int_{\Omega} \Psi(F(\boldsymbol{X}))d\boldsymbol{X} \tag{2.14}$$

b. Strain & Stress

Elastic forces are the response of an object to a given deformation towards the decrease of potential energy. In general, the direct relationship between forces and deformation may be too complex or not intuitive for modeling real-world materials. Thus, most of material models use the concept of *stress tensor* as a force measure and *strain tensor* as deformation measure. The *constitutive laws* then relate stress and strain by equations.

The *1st Piola-Kirchhoff stress tensor* $P \in \mathbb{R}^{3\times3}$ is a quantity from which we can calculate the elastic forces acting both on the interior and on the surface of an object. By definition, it is the derivative of the strain energy density with respect to the deformation gradient:

$$P(F) = \frac{\partial \Psi}{\partial F} \tag{2.15}$$

From $P(F)$, the internal force density $\boldsymbol{f}_{den}$ (per unit of undeformed volume) and the surface traction $\boldsymbol{\tau}$ (per unit of undeformed area) can be respectively computed by:

$$\boldsymbol{f}_{den}(\boldsymbol{X}) = -\nabla_{\boldsymbol{X}} \cdot P(F(\boldsymbol{X})) \qquad \text{for } \boldsymbol{X} \text{ in the interior} \tag{2.16}$$

$$\boldsymbol{\tau}(\boldsymbol{X}) = -P(F(\boldsymbol{X})) \cdot \boldsymbol{N} \qquad \text{for } \boldsymbol{X} \text{ on the surface} \tag{2.17}$$

where $\nabla_{\boldsymbol{X}}\cdot$ is the divergence operator with respect to the positions and $\boldsymbol{N}$ is the outward normal of the surface at $\boldsymbol{X}$. Although there are other types of stress tensors, we will not go further because the quasi-Newton methods do not require calculating specific strain energy derivatives.

Strain tensors are quantities intended to describe the magnitude and properties of deformations more accuretely. A fundamental model for strain is the *green strain tensor* $E \in \mathbb{R}^{3\times3}$, defined by:

$$E(F) = \frac{1}{2}(F^T F - I) \tag{2.18}$$

a nonlinear function of F where $I \in \mathbb{R}^{3\times3}$ is the identity matrix. While the deformation gradient entries change under rigid-body rotations, $E(F)$ cancels the effect of orthogonal transformations (this can be seen from polar decomposition of F). Therefore, several constitutive laws are built on the top of the green strain tensor. Another useful strain model is the *small strain tensor*, which is calculated from the linear approximation of $E(F)$:

$$\varepsilon(F) = \frac{1}{2}(F + F^T) - I \tag{2.19}$$

It is only valid under small deformations, but due to its linearity, it fairly simplifies the stress-strain relation of some materials.

c. Materials

Important measures associated to deformations were defined above. Now, we will see a few elasticity models widely used in graphics and engineering. Materials may be classified into two categories: *isotropic materials*, whose properties are equal along any direction; *anisotropic materials*, whose properties vary according to the direction. That is, the strain energy of isotropic materials is rotation-invariant. In this work, we focus on the first group.

Linear elasticity. This is the simplest material model, in which the Piola stress tensor depends linearly on the small strain tensor and, therefore, the deformation gradient. Its constitutive law is defined by:

$$\Psi(\varepsilon) = \mu\varepsilon : \varepsilon + \frac{\lambda}{2}\mathrm{tr}^2(\varepsilon) \tag{2.20}$$

where μ, λ are the Lamé coefficients (related to *Young's modulus* and *Poisson's Ratio*) [22]. The operation $a : b$ denotes the tensor product between a and b, the summation of the element-wise multiplication of both tensors. This leads to:

$$P(F) = \mu(F + F^T - 2I) + \lambda\mathrm{tr}(F - I)I \tag{2.21}$$

The linear dependency of forces on positions makes this material computationally cheap, because any required derivatives are constant and can be precomputed. However, large deformations are poorly represented by such model.

St. Venant-Kirchhoff. This model is very similar to linear elasticity, but uses the Green strain tensor instead of the small strain tensor. Thus, the stress-strain relation becomes nonlinear:

$$\Psi(E) = \mu E : E + \frac{\lambda}{2}\mathrm{tr}^2(E) \tag{2.22}$$

$$P(F) = F(2\mu E + \lambda\mathrm{tr}(E)I) \tag{2.23}$$

One of the problems listed by [22] is its poor resistance to compressions, because the restorative elastic forces reach a maximum value after certain threshold of deformation along some axis. In practice, the consequence is that external forces may enforce the material to invert unexpectedly.

Principal Stretches. Some isotropic materials such as the *invertible St. Venant-Kirchhoff* and *spline-based* are described by orientation-independent energy functions. In this case, the strain energy density is expressed in terms of the *principal stretches*, which are singular values of the deformation gradient:

$$\Psi(F) = \Psi(\underbrace{U\Sigma V^T}_{\text{SVD}}) = \Psi(\Sigma) \tag{2.24}$$

where $\Sigma = \text{diag}[\sigma_1, \sigma_2, \sigma_3]$ is the diagonal matrix of principal stretches. The physical meaning is that $\sigma_1, \sigma_2, \sigma_3$ are the scales of the most representative deformation directions; and directions make up an orthogonal basis specified by the columns of U. Therefore, since the stretches are always defined with respect to this basis, Ψ becomes an orientation-invariant formula from which we can easily detect and solve the inversion problem.

2.3 Finite Element Method

In the previous section, we described the theory of continuum mechanics for hyperelasticity. Its mathematical formulation yields a partial differential equation whose exact solution is the ideal motion of the continuous object, that is, the displacement of each material point over time. Nevertheless, the deformation map is a vector field that needs an infinite number of functions to be fully described. In practice, we must use a discretization scheme that allows us to accurately represent the object motion by a finite number of functions or parameters. This achieved by the *Finite Element Method* (FEM).

In the FEM, the domain Ω is first discretized into small regions called *elements*, which can be either tetrahedra or cubes (voxels). Then, variables such as the deformation map and forces are approximated as piecewise functions whose values are measured at the vertices and interpolated within the elements. This makes possible computing elastic forces resulting from element deformations for a given configuration of vertices. This procedure converts partial differential equations into ordinary differential equations that we can integrate using the methods presented previously.

a. Volumetric Meshes

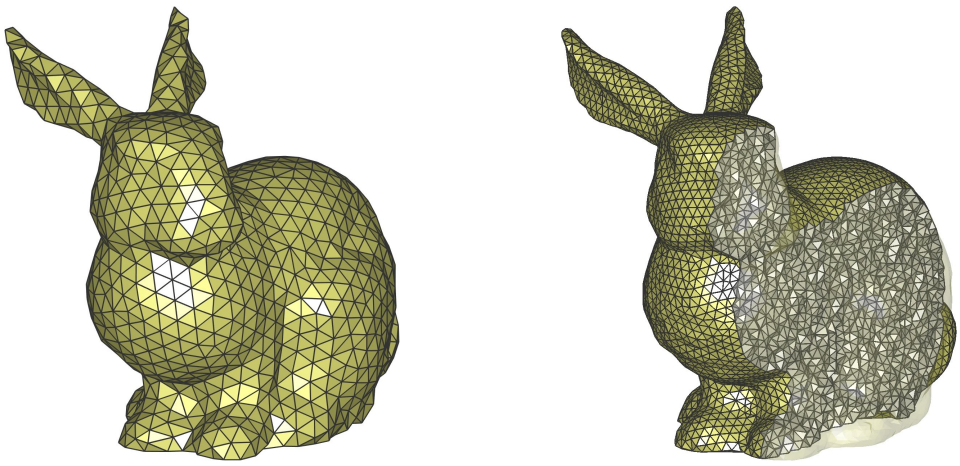


Figure 2.3: Meshes for the classic bunny model. On the left, a surface triangulation typically used for rendering. On the right, a detailed tetrahedal mesh for discretizing the enclosed domain in FEM-based techniques. JIGSAW (2017). An unstructured mesh generator. [image] Available at: https://sites.google.com/site/dengwirda/jigsaw [Accessed 28 Oct. 2017].

The set of vertices and elements originated from the discretization of $\Omega \in \mathbb{R}^3$ is called *volumetric mesh*. In this work, we deal with tetrahedral meshes only, which are the equivalent to triangulations in 2D. Tetrahedral meshes are unstructured grids whose level of detail can be tuned to achieve the desired accuracy. In computer graphics, we usually have available 3D models that represent the surface of an object. However, to simulate deformable objects, we must distribute vertices and elements over the enclosed volume, the interior (Figure 2.3). There are several algorithms to construct tetrahedral meshes [20, 21]. Since this is not the scope of our research, we use the open-source library VegaFEM [23] to generate such models.

Let n be the number of vertices and m be the number of elements in our mesh. Each tetrahedron E_j is specified by four vertices: (V_a, V_b, V_c, V_d) with $1 \leq a, b, c, d \leq n$ and a vertex V_i may be shared by multiple tetrahedra. When we work with just one tetrahedron, we may use local indices $k = 0..3$ to address its vertices. The mass distribution of the object is defined by the matrix $M \in \mathbb{R}^{3n \times 3n}$ which is diagonal if the masses are concentrated at vertices (*lumped mass*). At reference configuration, the vertex V_i has a position $\boldsymbol{X}_i$; during a deformation, it is displaced to a new position $\boldsymbol{x}_i$. FEM approximates the deformation map $\boldsymbol{\phi}$ to a piecewise function divided into components $\boldsymbol{\phi}_j$ that hold within the corresponding element E_j. To avoid increasing the nonlinearity of equations, most of authors use *linear tetrahedral meshes*, where $\boldsymbol{\phi}_j$ is modeled as an independent affine transformation with 12 degrees of freedom. Analyzing tetrahedra separately (Figure 2.4), we notice that deformations yield four point correspondences.

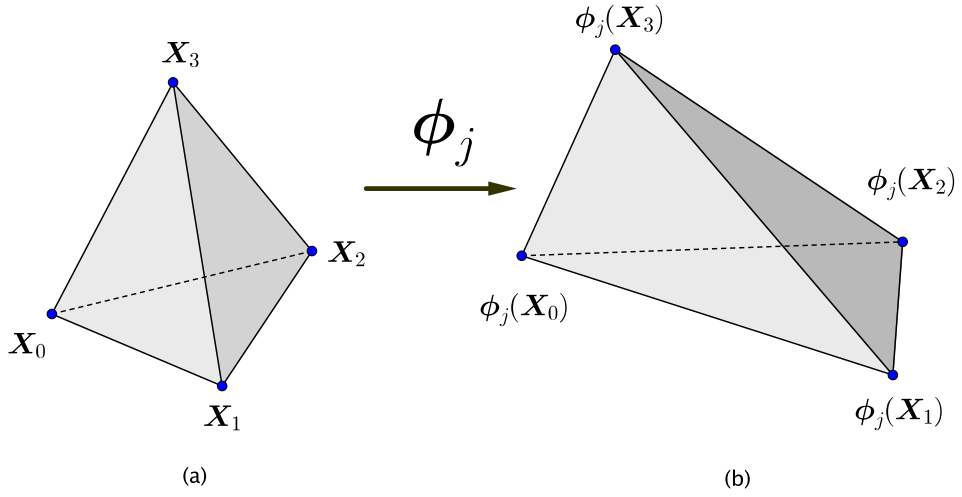


Figure 2.4: Deformation of tetrahedron E_j. (a) Rest shape. (b) Deformed shape.

With these four correspondences, the coefficients of the general affine transformation $\boldsymbol{\phi}_j(\boldsymbol{X}) = A_j\boldsymbol{X} + \boldsymbol{b}$ for all $\boldsymbol{X} \in E_j$ may be found and the gradient $F_j = \nabla\boldsymbol{\phi} = A_j$ may be derived. F_j is essentially the matrix that transforms the three directed outgoing edges of

a reference vertex to their deformed configuration. Thus,

$$F_j \underbrace{\begin{bmatrix}(\boldsymbol{X}_1 - \boldsymbol{X}_0) & (\boldsymbol{X}_2 - \boldsymbol{X}_0) & (\boldsymbol{X}_3 - \boldsymbol{X}_0)\end{bmatrix}}_{D_{m,j}\ \text{(constant)}} = \underbrace{\begin{bmatrix}(\boldsymbol{x}_1 - \boldsymbol{x}_0) & (\boldsymbol{x}_2 - \boldsymbol{x}_0) & (\boldsymbol{x}_3 - \boldsymbol{x}_0)\end{bmatrix}}_{D_{s,j}\ \text{(variable)}}$$

$$\Downarrow$$

$$F_j = D_{s,j} D_{m,j}^{-1} \tag{2.25}$$

where $D_{s,j}$ depends on the current deformed shape and $D_{m,j}^{-1}$ is constant matrix that we can precompute.

b. Elastic Forces

Recall that the strain energy density Ψ is a function of the deformation gradient only. Therefore, in linear tetrahedral meshes, Ψ becomes a constant within each element E_j. Let $\Psi_j(F_j)$ be the strain energy density of E_j. Then, we may simplify the total energy (*Eq.* 2.15) of a single tetrahedron to:

$$g_j = g\{\boldsymbol{\phi}_j\} = \int_{E_j} \Psi(F_j) d\boldsymbol{X} = \Psi_j(F_j)\text{vol}(E_j) \tag{2.26}$$

where $\text{vol}(E_j)$ denotes the volume of E_j at rest shape. A tetrahedron can be interpreted as a *hyper-spring*, because it exerts elastic forces on all its vertices to different directions. Applying the conservation law of the total energy, we can calculate these forces from the spatial derivative of the potential energy g_j [22]. Thus, this yields a closed-form expression for the contribution of E_j to the elastic forces:

$$H_j = \begin{bmatrix}\boldsymbol{f}_0 & \boldsymbol{f}_1 & \boldsymbol{f}_2\end{bmatrix} = -\text{vol}(E_j) P(F) D_{m,j}^{-T} \tag{2.27}$$

where $\boldsymbol{f}_0, \boldsymbol{f}_1, \boldsymbol{f}_2$ are the forces acting on the first three vertices of E_j. The force on the last vertex is just $\boldsymbol{f}_3 = -(\boldsymbol{f}_0 + \boldsymbol{f}_1 + \boldsymbol{f}_2)$. Notice that $P(F) = \partial\Psi(F)/\partial\boldsymbol{X}$ is determined by the corresponding constitutive law (Section 2.2 c.).

c. Governing Equations

At this point, we have formulated the discretization of deformation measures and elastic force models for linear tetrahedral meshes. Meshes can be seen as discrete systems where particles are vertices and the internal forces arise from the contribution of individual elements. Therefore, similarly to *Eq.* 2.1, we may write down a differential equation that describes the overall system motion from Newton's second law. Instead of the actual vertex positions $\boldsymbol{x} = \boldsymbol{\phi}(\boldsymbol{X})$, most of authors use the vector of displacements $\boldsymbol{u} = \boldsymbol{x} - \boldsymbol{X}$

as the unknown (*e.g.*, $\boldsymbol{u} = 0$ indicates reference configuration). This yields:

$$M\ddot{\boldsymbol{u}} = \boldsymbol{f}_{int}(\boldsymbol{u}) \tag{2.28}$$

We integrate the above equation using the methods presented in Section 2.1 b. In fact, this procedure is analogous to what was described, but most of force models are higly nonlinear, which makes the equations to me more complex. Furthermore, implicit integration envolves calculating force differentials, an expensive computation for real-time simulation. For more information on FEM and also damping models, please refer to [22].

2.4 Model Reduction

Model reduction for simulating general dynamic systems [15] is a classical technique which substantially simplifies the time-step computation by projecting the original high-dimensional space of solutions to a precomputed low-dimensional subspace. As a result of this projection, the simulation loses local details of elasticity and dynamics because the configurations tend to follow displacements with the smallest increase of energy. Nonetheless, the computation in reduced space becomes less expensive and proper for real-time applications, as the reduction usually decreases the number of degrees of freedom hundreds of times.

In deformable object simulation, this subspace is defined by the column-space of a basis matrix $U \in \mathbb{R}^{3n \times r}$, where each one of the r columns represents a mode of vibration. The vector of displacements in full-space $\boldsymbol{u} \in \mathbb{R}^{3n}$ can then be calculated as a linear combination of all modes, which is described by $\boldsymbol{u} = U\boldsymbol{z}$, where $\boldsymbol{z} \in \mathbb{R}^r$ is the vector of reduced coordinates. Also, notice that at rest configuration $\boldsymbol{x_0}$ in full space, the reduced coordinates are just $\boldsymbol{z} = 0$. For more details on how to generate the basis or how model reduction works, please refer to [6]. This is the last concept needed for understanding the most recent works and the presented method.

Chapter 3
Related Work

In this chapter, we group techniques for real-time simulation of deformable objects into two branches: *classical methods* and *projective methods*. While classical methods numerically integrate the governing differential equations of motion to compute the next time-step state, projective methods minimize an objective function that properly models constraints and potential energies. The work presented here belongs to the latter group, which may be considered a less conservative approach.

3.1 Classical Methods

Several State-of-Art techniques for simulating deformable bodies are based on the classical *implicit backward Euler* integrator [5]. As seen previously, the most fundamental step of this method is a nonlinear system solve, which is computationally expensive because the system matrix changes at every iteration. Thus, one typically runs the Newton's method for just one iteration. The implicit Euler is able to simulate any object given the constitutive law of its material; that is, the strain energy, the elastic forces and the spatial derivative of elastic forces (tangent stiffness matrix). VegaFEM [23] is an open-source library that implements a large variety of material models that can be used with different integrators, including the implicit Euler and the implicit Newmark [18].

Although these classical integrators are accurate, they may be costly for the available computation time in real-time applications. Therefore, early works investigated model reduction to decrease the differential equation order [19, 11]. In 2005, Barbič *et al.* [7] proposed an efficient technique for integrating StVK deformable objects in a low-dimensional subspace, where the elastic forces are shown to be cubic polynomials. This work also introduced *modal derivatives*, a new algorithm for generating bases for such subspaces. Within the scope of model reduction, An *et al.* presented in the optimal cubature [1], which accelerates the subspace integration by approximating reduced forces using a small subset of the mesh elements. Later, further algorithms were developed to construct cubatures faster [25]. Reduced methods run at high frame-rates, but with a significant loss in animation quality.

3.2 Projective Methods

In 2007, Müller *et al.* proposed in [16] *Position-Based Dynamics* (PBD), a totally different way of integrating general systems. This cheap technique allows us to simulate particle systems with custom constraints while achieving a real-time performance. It does not use velocities directly and may be easily coupled with collision detection. Martin *et al.* went further and formulated a technique to simulate objects from examples using a projective approach [14]. Other authors develop methods to address the numerical damping problem present in traditional integrators by imposing energy conservation [10, 24].

More recently, Bouaziz *et al.* [8] presented a novel solver called *Projective Dynamics* (PD), which generalizes the PBD constraints to potential energies. This approach is able to model not only several types of forces and constraints at once, but also nonlinear elasticity of volumetric tetrahedral meshes. Though its formulation is robust and general, the objective energy is limited to quadratic expressions. For this reason, Liu *et al.* built a quasi-Newton solver that minimizes arbritary elastic energies to support more material models [13]. Its formulation is similar to PD, but additional steps are added to accelerate the convergence. As needed to derive our method, we will provide an overview of this quasi-Newton method in the next section.

3.3 Quasi-Newton Solver

Notation. Let n be the number of vertices and m be the number of tetrahedra in the object mesh. Let h be the time-step duration. The vertices at rest configuration are represented by the vector $\boldsymbol{x}_0 \in \mathbb{R}^{3n}$ and the mass is distributed according to the positive-definite matrix $M \in \mathbb{R}^{3n\times 3n}$, which is diagonal when the mass is lumped. At time-step $t = l$, the state of this object is described by a vector of vertex positions $\boldsymbol{x}^{(l)}$ and its velocity $\dot{\boldsymbol{x}}^{(l)}$. At iteration k, the variables will be specified with the subscript k, such as $\boldsymbol{x}_k$. The vector of displacements is generically defined as $\boldsymbol{u} = \boldsymbol{x} - \boldsymbol{x_0}$. Even though x, y and z coordinates can be easily decoupled for a faster system solve, we write them down in the $3n \times 1$ form for clarity of equations.

Quasi-Newton Solver. This section is an overview of the solver proposed in [13], which extends the projective dynamics technique to handle more general hyperelastic materials. As a quasi-Newton method, it is computationally cheaper than the original Newton's method because it does not require the Hessian evaluation at every iteration and, therefore, does not add a significant overhead of computation to the original projective dynamics optimization. The main idea is to find the next time-step state by minimizing the energy function $g(\boldsymbol{x})$. It deals with different constitutive models of materials by including in $g(\boldsymbol{x})$ an inertia term and an elasticity term $E(\boldsymbol{x})$, which is a sum of all individual

tetrahedron elastic energies $E_i(\boldsymbol{x})$:

$$g(\boldsymbol{x}) = \underbrace{\frac{1}{2h^2}(\boldsymbol{x}-\boldsymbol{y})^T M (\boldsymbol{x}-\boldsymbol{y})}_{\textbf{Inertial Energy}} + \underbrace{\sum_{i=1}^{m} w_i E_i(\boldsymbol{x})}_{\textbf{Elasticity Energy} E(\boldsymbol{x})} \tag{3.1}$$

where w_i are weights proportional to the tetrahedra volumes and stiffness, and the constant $\boldsymbol{y} \in \mathbb{R}^{3n}$ is the current time-step prediction of the state based only on inertia and external forces. It is evaluated using the previous time-step data and current external forces $\boldsymbol{f}_{\boldsymbol{ext}}^{(l)}$,

$$\boldsymbol{y}^{(l)} = \boldsymbol{x}^{(l-1)} + h\dot{\boldsymbol{x}}^{(l-1)} + h^2 M^{-1} \boldsymbol{f}_{\boldsymbol{ext}}^{(l)} \tag{3.2}$$

For isotropic materials (independent of orientation), the energy density Ψ_i can be written as function of the principal stretches of the tetrahedron i, which are the singular values of the deformation gradient $F_i \in \mathbb{R}^{3\times3}$ [22]. Assuming that each element is linear and therefore the deformation gradient F_i is constant within it, the energy per element is just $E_i(\boldsymbol{x}) = V_i \Psi_i(\boldsymbol{\sigma}_i)$, where V_i is the tetrahedron volume at rest pose and $\boldsymbol{\sigma}_i \in \mathbb{R}^3$ is the vector of principal stretches.

The algorithm searches for $\boldsymbol{x}_{opt} = \underset{x}{\operatorname{argmin}}\ g(\boldsymbol{x})$, which is equivalent to finding the solution for $\nabla g(\boldsymbol{x}) = 0$. This method is iterative and usually takes about 10 to 20 iterations to reach the minimum. At iteration k, the state is updated towards the descent direction $\boldsymbol{d}(\boldsymbol{x})$ computed by the Limited BFGS algorithm (L-BFGS) [12] followed by a *line search* [3]. L-BFGS is a method to find $\boldsymbol{d}(\boldsymbol{x})$ by updating the Hessian matrix approximation $A \approx \nabla^2 g(\boldsymbol{x})$ according to curvature data from previous iterations without building it in memory. Because of its improved descent direction, the convergence is accelerated even for large deformations. The simulation starts with the solution of $A_0\ \boldsymbol{d}(\boldsymbol{x}) = -\nabla g(\boldsymbol{x})$, where A_0 is the initial Hessian approximation whose Cholesky decomposition can be precomputed (positive-definite matrix). Afterwards, $\boldsymbol{x}$ and $\nabla g(\boldsymbol{x})$ from the m previous iterations are used to correct $\boldsymbol{d}(\boldsymbol{x})$ as it was the solution of $A_k\ \boldsymbol{d}(\boldsymbol{x}) = -\nabla g(\boldsymbol{x})$, where $A_k \approx \nabla^2 g(\boldsymbol{x}_k)$ [12].

In order to compute the gradient of the strain energy in each tetrahedron, projective dynamics requires auxiliary variables $\boldsymbol{p}_i \in \mathbb{R}^9$ for $i = 1..m$ that correspond to the local projection of the deformation gradient F_i onto the zero strain energy manifold. Mathematically, $\boldsymbol{p}_i$ represents the entries of the closest orthogonal matrix R_i to F_i, which can be obtained through *polar decomposition* or *Singular-Value Decomposition* (SVD). All projective variables can be combined together into a $9m$-dimensional vector $\boldsymbol{p}$. Let $G_i \in \mathbb{R}^{9\times12}$ be the gradient operator which maps the displacements of a single tetrahedron $\boldsymbol{u}_i \in \mathbb{R}^{12}$ to a 9-vector whose elements are entries of the gradient F_i. Also, according to the volumetric mesh topology, for each tetrahedron i, we can define two selector matrices C_i $(12 \times 3n)$ and D_i $(9 \times 9m)$ that map $\boldsymbol{u}$ into $\boldsymbol{u}_i$ and $\boldsymbol{p}$ into $\boldsymbol{p}_i$, respectively. Thus, we

obtain

$$\nabla g(\boldsymbol{x}) = \frac{1}{h^2} M(\boldsymbol{x} - \boldsymbol{y}) + L\boldsymbol{x} - J\boldsymbol{p} \tag{3.3}$$

where $L = \sum_{i=1}^{m} w_i C_i^T G_i^T G_i C_i$ and $J = \sum_{i=1}^{m} w_i C_i^T G_i^T D_i$ are constant sparse matrices. Once the gradient is evaluated, L-BFGS is called to return $\boldsymbol{d}(\boldsymbol{x})$. Projective dynamics defines the system matrix as

$$A_0 = (\frac{1}{h^2} M + L) \approx \nabla^2 g(\boldsymbol{x_0}), \tag{3.4}$$

which reasonably approximates the initial Hessian matrix.

One of the most important contributions of [13] is to use a line search as a way to handle complex nonlinear materials, such as spline-based [27], St. Venant-Kirchoff and Neo-Hookean [22], avoiding energy accumulation. This is necessary because taking the full descent direction does not always lead to the minimal energy state for any type of material. So, the algorithm finds α_{opt} such that $\min_{\alpha} g(\boldsymbol{x}_{k-1} + \alpha \boldsymbol{d}(\boldsymbol{x}_k))$ for $\alpha \in [0, 1]$ and then updates $\boldsymbol{x}_k = \boldsymbol{x}_{k-1} + \alpha_{opt} \boldsymbol{d}(\boldsymbol{x}_k)$. However, for complex meshes (large m), the energy evaluation becomes a time-consuming part of the solver, since it has to perform SVD of all m deformation gradients at line search iteration. Notice that this quasi-Newton solver minimizes the ground-truth nonlinear elastic energy but uses the projective dynamics approximate gradient to perform the optimization.

Chapter 4

Reduced Quasi-Newton Solver

In this chapter, we will develop the *reduced quasi-Newton solver*. We build this method by applying model reduction and cubature techniques on the original quasi-Newton solver presented in the previous section. To simplify notation, the quasi-Newton solver will be called *full-solver* and the reduced quasi-Newton will be called *reduced-solver*. Unlike classical methods of integration, these iterative techniques are formulated as an optimization problem whose solution is the next time-step state. Therefore, we first derive the reduced expressions – initial guess, energy (cost function), forces (gradient) and system matrix (Hessian). Secondly, we show how to construct a cubature subset using the combinatorial optimization method proposed in [2] with the projective dynamics force model. In the last section, we provide the complete algorithm for our reduced solver and explain its behavior.

4.1 Derivation

For the sake of consistency, we will keep the notation used in the Section 3.3. Since this work is not concerned with basis generation, we assume that the modal matrix $U \in \mathbb{R}^{3n\times r}$ with r modes is computed in advance through modal derivatives [7] and has been mass-orthonormalized, which yields $\tilde{M} = U^T M U = I$. An implementation of modal derivatives is also available on VegaFEM [23]. After the projection, the state variable and the velocity become $\boldsymbol{z}, \dot{\boldsymbol{z}} \in \mathbb{R}^r$ respectively, where r is the number of modes of vibration. Except for the displacements, we express variables in reduced space with a tilde, such as $\tilde{g}$ (the reduced energy).

Reduced Initial Guess. The first step of an optimization algorithm is the initial estimation of the unknown variable. In projective dynamics, the prediction $\boldsymbol{y}^l \in \mathbb{R}^{3n}$ is dependent on the previous step state $\boldsymbol{x}^{(l-1)}$, $\dot{\boldsymbol{x}}^{(l-1)}$ and the current external forces $\boldsymbol{f}^l_{\boldsymbol{ext}}$. The reduced initial guess can be obtained by left-multiplying *Eq.* 3.2 by $U^T M$ and writing

the full variables as a function of the reduced variables:

$$U^T M \underbrace{\boldsymbol{y}^l}_{U\tilde{\boldsymbol{y}}^l+\boldsymbol{x}_0} = U^T M \underbrace{\boldsymbol{x}^{(l-1)}}_{U\boldsymbol{z}^{(l-1)}+\boldsymbol{x}_0} + hU^T M \underbrace{\dot{\boldsymbol{x}}^{(l-1)}}_{U\dot{\boldsymbol{z}}^{(l-1)}} + h^2 U^T \boldsymbol{f}_{\boldsymbol{ext}}^{(l)}$$

$$U^T MU\tilde{\boldsymbol{y}}^l + U^T M\boldsymbol{x}_0 = U^T MU\boldsymbol{z}^{(l-1)} + U^T M\boldsymbol{x}_0 + hU^T MU\dot{\boldsymbol{z}}^{(l-1)} + h^2 U^T \boldsymbol{f}_{\boldsymbol{ext}}^{(l)}$$

$$\Downarrow \text{ simplifying } U^T MU = I$$

$$\tilde{\boldsymbol{y}}^l = \boldsymbol{z}^{(\boldsymbol{l-1})} + \underbrace{h\dot{\boldsymbol{z}}^{(\boldsymbol{l-1})}}_{\textbf{Inertia term}} + \underbrace{h^2 U^T \boldsymbol{f}_{\boldsymbol{ext}}^{\boldsymbol{l}}}_{\textbf{External forces term}} \tag{4.1}$$

For simplicity, $\tilde{\boldsymbol{y}}^l$ will be written as $\tilde{\boldsymbol{y}}$ in the following equations.

Reduced Energy. As seen in the previous section, for a given state $\boldsymbol{x} \in \mathbb{R}^{3n}$, the full-solver energy is a function $g(\boldsymbol{x})$ (*Eq.* 3.1). In the reduced-solver, we want to find $\tilde{g}(\boldsymbol{z})$ for a given $\boldsymbol{z}$. Thus, applying $\boldsymbol{x} = U\boldsymbol{z} + \boldsymbol{x_0}$ into *Eq.* 3.1, the objective function can be derived as:

$$\begin{aligned}
\tilde{g}(\boldsymbol{z}) &= g(U\boldsymbol{z} + \boldsymbol{x_0}) \\
&= \frac{1}{2h^2}(U\boldsymbol{z} - U\tilde{\boldsymbol{y}})^T M(U\boldsymbol{z} - U\tilde{\boldsymbol{y}}) + \sum_{i=1}^{m} w_i E_i(U\boldsymbol{z} + \boldsymbol{x_0}) \\
&= \frac{1}{2h^2}(\boldsymbol{z} - \tilde{\boldsymbol{y}})^T \underbrace{U^T MU}_{I}(\boldsymbol{z} - \tilde{\boldsymbol{y}}) + \sum_{i=1}^{m} w_i E_i(U\boldsymbol{z} + \boldsymbol{x_0}) \\
&= \frac{1}{2h^2}\|\boldsymbol{z} - \tilde{\boldsymbol{y}}\|^2 + \sum_{i=1}^{m} w_i \underbrace{E_i(U\boldsymbol{z} + \boldsymbol{x_0})}_{\text{isotropy: } V_i\Psi(\boldsymbol{\sigma_i})}
\end{aligned} \tag{4.2}$$

Reduced Gradient. At each solver iteration, the reduced gradient is evaluated by projecting the full gradient (*Eq.* 3.3) down onto the basis:

$$\begin{aligned}
\nabla\tilde{g}(\boldsymbol{z}) &= U^T \nabla g(U\boldsymbol{z} + \boldsymbol{x_0}) \\
&= U^T \left[\frac{1}{h^2} M(U\boldsymbol{z} - U\tilde{\boldsymbol{y}}) + LU\boldsymbol{z} + L\boldsymbol{x_0} - J\boldsymbol{p}\right] \\
&= \frac{1}{h^2} \underbrace{U^T MU}_{I}(\boldsymbol{z} - \tilde{\boldsymbol{y}}) + U^T(LU\boldsymbol{z} - J\boldsymbol{p}) \\
&= \frac{1}{h^2}(\boldsymbol{z} - \tilde{\boldsymbol{y}}) + \underbrace{(U^T LU)}_{constant}\boldsymbol{z} - \underbrace{(U^T J)}_{constant}\boldsymbol{p} + \underbrace{U^T L\boldsymbol{x_0}}_{constant}
\end{aligned} \tag{4.3}$$

Reduced L-BFGS and Line Search. Similarly to the full-solver, L-BFGS algorithm will compute a reduced descent direction by using m previous solver iterations. The core idea is to initialize this direction by solving a linear system with the initial reduced Hessian approximation $\tilde{A}_0 \in \mathbb{R}^{r\times r}$ before the update using previous data.

$$\tilde{A}_0 = U^T A_0 U \implies \boldsymbol{d}(\boldsymbol{z_k}) = -\tilde{A}_0^{-1}\nabla\tilde{g}(\boldsymbol{z_k}) \tag{4.4}$$

If the dense inverse $\tilde{A}_0^{-1}$ is precomputed, the above linear system solve becomes a simple matrix multiplication. Notice that in reduced space all the steps (except for the system solve) are just $O(r)$, which makes L-BFGS a small fraction of the overall time-step computation (approximately 2%, according to the experiments).

The reduced line search iterations proceed in the same way as in the full solver, but whenever the straing energy is evaluated, the state $\boldsymbol{x}$ in full coordinates has to be retrieved by $\boldsymbol{x} = U\boldsymbol{z} + \boldsymbol{x_0}$ to compute the principal stretches $\boldsymbol{\sigma_i}$.

4.2 Optimal Cubature

So far, we defined a solver that searches for the new state in a subspace, but if compared to model deduction on the classic implicit backward Euler method, it does not speed up the time-step computation considerably. Although its descent direction computation is fast (using the reduced L-BFGS), it turns out that the gradient and energy computations are still the bottleneck of the entire optimization because they require assembling the vertices, performing the SVD decomposition of all tetrahedra and also large matrices multiplication. Therefore, to avoid this heavy computation, we use the idea of optimal cubature conceived in [2] and the training algorithm developed in [25]. In reduced FEM, optimal cubature is the limited-size subset S of the set E of tetrahedra such that the total reduced internal forces $\tilde{F}_{total}$ can be best approximated as a linear combination of the reduced internal forces generated by each individual element. In other words,

$$\tilde{F}_{total}(U\boldsymbol{z}) = \sum_{i \in E} \tilde{\boldsymbol{f}}_i(U\boldsymbol{z}) \approx \sum_{i \in S} \alpha_i \tilde{\boldsymbol{f}}_i(U\boldsymbol{z}) \tag{4.5}$$

where α_i is the optimal weight of the i-th tetrahedron computed by the training algorithm. As cubature exploits the correlation between single tetrahedron forces when projected onto the basis, we can extend this idea to more general functions (after they are projected onto the subspace), such as the term $L(U\boldsymbol{z}) - J\boldsymbol{p}$ in the reduced gradient (*Eq.* 4.3) and also the strain energy (*Eq.* 4.2). For our solver, we train the data set to approximate the internal force model of the gradient (the elasticity term). Once we find the optimal subset S, we perform SVD on the selected deformation gradients $F_i \ \forall i \in S$ instead of all of them. At this point, we assume that S and the corresponding set of weights $W = \{\alpha_i\}, \ \forall i \in S$ are known and precomputed (using the above methods). Then, we can rewrite the reduced equations for the energy as:

$$\tilde{g}(\boldsymbol{z}) = \frac{1}{2h^2}\|\boldsymbol{z} - \tilde{\boldsymbol{y}}\|^2 + \sum_{i \in S} \alpha_i w_i E_i(U\boldsymbol{z} + \boldsymbol{x_0}) \tag{4.6}$$

Note that the state $\boldsymbol{x} = U\boldsymbol{z} + \boldsymbol{x_0}$ no longer needs be fully calculated; only the cubature vertices $i \in S$, that is, the dot product between each row i of U and $\boldsymbol{z}$. Recall $\boldsymbol{p}$ is a

$9m$-dimensional vector whose blocks of 9 entries correspond to the rotation matrix elements of each tetrahedron. Then, by using cubature, only $9 * |S|$ elements of $\boldsymbol{p}$ are required to be computed. This is efficiently done in parallel as well as the sparse $U^T J\boldsymbol{p}$ product. Additionally, we also precompute the dense matrix $U^T LU$ using the cubature elements only. Let $J_i = w_i C_i^T G_i^T D_i$. Thus, the reduced gradient is computed as:

$$\nabla \tilde{g}(\boldsymbol{z}) = \frac{1}{h^2}(\boldsymbol{z} - \tilde{\boldsymbol{y}}) + \underbrace{(U^T LU)}_{r\times r}\boldsymbol{z} - \sum_{i\in S} \alpha_i \underbrace{(U^T J_i)}_{r\times 9}\boldsymbol{p}_i + \underbrace{U^T L\boldsymbol{x_0}}_{r\times 1}. \tag{4.7}$$

Since all terms are now in reduced space, the gradient evaluation becomes a cheap part of the technique.

4.3 Algorithm

Data: cubature S and basis U
input : previous time-step state $\boldsymbol{z}^{(l-1)}$, $\dot{\boldsymbol{z}}^{(l-1)}$ and current external forces $\boldsymbol{f}_{ext}^{l}$
output : current time-step state $\boldsymbol{z}^{l}$, $\dot{\boldsymbol{z}}^{l}$

$\boldsymbol{z}_0 = \tilde{\boldsymbol{y}}$ = ComputeInitialGuess($\boldsymbol{z}^{(l-1)}$, $\dot{\boldsymbol{z}}^{(l-1)}$, $\boldsymbol{f}_{ext}^{l}$) // Eq. 4.1
$\boldsymbol{x}_S$ = AssembleCubatureVertices($\boldsymbol{z}_0$)
$\tilde{g}(\boldsymbol{z}_0), \boldsymbol{p}_S$ = EvaluateObjectiveEnergy($\boldsymbol{z}_0$, $\tilde{\boldsymbol{y}}$, $\boldsymbol{x}_S$) // Eq. 4.6
for k = 1..N **do**
    $\nabla \tilde{g}(\boldsymbol{z}_k)$ = ComputeGradient($\boldsymbol{z}_{k-1}$, $\boldsymbol{p}_S$) // Eq. 4.7
    $\boldsymbol{dz}_k$ = L-BFGS($\nabla \tilde{g}(\boldsymbol{z}_{k-1})$, $\tilde{A}_0^{-1}$)
    $\alpha = 2$
    repeat
        $\alpha = \alpha/2$
        $\boldsymbol{z}_k = \boldsymbol{z}_{k-1} + \alpha \boldsymbol{dz}_k$
        $\boldsymbol{x}_S$ = AssembleCubatureVertices($\boldsymbol{z}_k$)
        $\tilde{g}(\boldsymbol{z}_k), \boldsymbol{p}_S$ = EvaluateObjectiveEnergy($\boldsymbol{z}_k$, $\tilde{\boldsymbol{y}}$, $\boldsymbol{x}_S$) // Eq. 4.6
        `// Armijo Condition`
    until $\tilde{g}(\boldsymbol{z}_k) \leq \tilde{g}(\boldsymbol{z}_{k-1}) + \gamma\alpha(\nabla g(\boldsymbol{z}_{k-1}))^T \boldsymbol{dz}_k$;
end
$\boldsymbol{z}^{l} = \boldsymbol{z}_N$, $\dot{\boldsymbol{z}}^{l} = (\boldsymbol{z}^{l} - \boldsymbol{z}^{l-1})/h$

Algorithm 1: Reduced quasi-Newton solver

The solver takes the previous frame (time-step) data as input and returns the current frame data. We write only the most essential functions and variables in the Algorithm 1 to keep notation simple. Although the optimization is done in reduced space, assembling the full state $\boldsymbol{x}^l = U\boldsymbol{z}^l + \boldsymbol{x}_0$ at the end is needed whenever the volumetric mesh must be updated. Partially assembled variables such as $\boldsymbol{x}_S \in \mathbb{R}^{3n}$ and $\boldsymbol{p}_S \in \mathbb{R}^{9m}$ are sparse vectors that contain only the cubature data; the remaining entries are zero.

Our reduced quasi-Newton solver works as follows:

- Line 1. Direct reduced initial guess computation from *Eq.* 4.1.
- Lines 2-3. Two steps for initial energy evaluation. First, assemble cubature vertices only ($\boldsymbol{x}_S$). Then, build cubature deformation gradients and decompose them to find projective variables $\boldsymbol{p}_S$. Depending on the elastic model, extracting the principal stretches of the deformation gradients may be required to calculate the energy. In this case, SVD is recommended because it provides not only the stretches but also the projective variables. With the computed data, evaluate objective function from *Eq.* 4.6.
- Lines 5-6. Descent direction computation using projective variables. The first line is the basic gradient evaluation (*Eq.* 4.7) with several constant terms. As mentioned, L-BFGS corrects the direction as if the Hessian matrix is updated for the current vertex positions. For more details on how L-BFGS works, please refer to [13].
- Lines 8-14. Reduced line search with Armijo condition [3], which searches for α that minimizes the objective function along the descent direction. At each line search iteration, to evaluate the energy, we need recompute the projective variables or even the principal stretches for cubature elements. This is similar to lines 2-3.
- Line 16. At the end, the velocities are updated and variables are output.

This algorithm may be parallelized in several steps: partial or complete assembly of vertices, gradient evaluation, deformation gradient calculation and decomposition. However, due to the line search, both reduced and full quasi-Newton methods rely heavily on energy evaluation, which is performed multiple times per time-step. In fact, line search is the bottleneck of the entire algorithm. While larger cubature subsets approximate projective dynamics elastic forces more accurately, they certainly make the energy evaluation expensive as more deformation gradients must be decomposed. Yet, it is also very favorable to GPU implementations (typically, over 100 simultaneous SVDs may be executed at once). For these reasons, the basis size r has small impact on the overall computational cost of CPU implementations.

Chapter 5

Experiments and Results

5.1 Experiments

In this chapter, we compare the *implicit backward Euler*, the *quasi-Newton* and the *reduced quasi-Newton* methods. We perform a series of experiments on meshes with different materials and geometries, and record the simulation data such as deformations, kinetic energy, average frame rate and performance profile. External forces are pre-recorded to feed 600-frame animations that run the different solvers under the same conditions (time-step, material, fixed vertices). Then, we analyze the behavior and features of the three methods against the ground-truth, which is generated by running the implicit backward Euler $5-10$ sub-steps per time-step (*i.e.*, executed at a higher sample-rate).

To assess the methods, we use three tetrahedral meshes (Figure 5.1): *Letter A* model with 575 vertices and $2,144$ elements, *Bridge* with $4,000$ vertices and $12,827$ elements, and *Dragon* with $46,736$ vertices and $160,553$ elements (all available online). Also, two material models are examined: *invertible St. Venant-Kirchoff* material (StVK) and *Projective Dynamics* material (PD). Due to the limited time for this research, we did not explore more materials and models; but for a given combination of material and mesh, we cover a large variety of dynamics by performing sets of 3 experiments including high-amplitude jerk, high-frequency vibration, large stretching, and strong inversion.

In this research, we implemented both quasi-Newton solvers, but we use the implicit backward Euler available on VegaFEM [23]. All the implementations are done in CPU (C++) and run on a MacBook Pro with Intel i5 2.7GHz processor, Intel Iris Graphics 6100 (1536 MB) and 8GB-DDR3 memory (1867 MHz). Several steps of all methods may be parallelized in different ways, that is, with different number of threads; empirically, we choose the multi-threading parameters that best decrease the per-frame computation time.

Finding metrics for evaluating a new physics-based animation method may be a difficult task, because even standard error measures such as the per-vertex distance might not indicate how realistic or plausible the output animation is for humans. Therefore, we first assess the trade-off between animation quality and computational cost in all solvers using a qualitative approach with the ground-truth as reference. Then, in the latter section, we

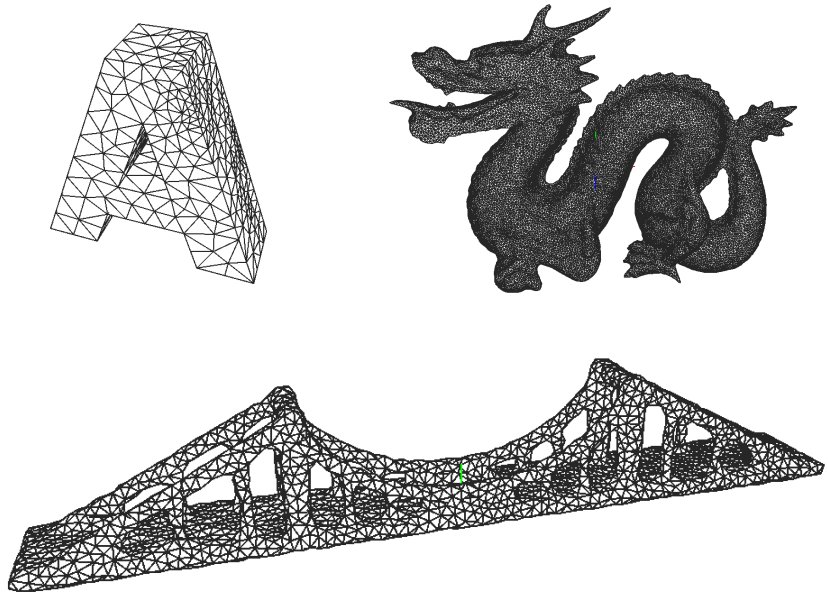


Figure 5.1: Tetrahedral meshes at reference configuration. On the top-left corner, the *Letter A* model (small). On the top-right corner, the *Dragon* model (large). On the bottom, the *Bridge* model (medium size).

analyze the numerical damping over the experiments from the kinect energy data.

5.2 Performance vs. Animation Quality

In the *Letter A* model experiments, we use time-step $h = 10^{-3}s$. While the full solver converges with 10 iterations, our reduced solver needs 20, otherwise it becomes unstable. We try model reduction with two cubature sets, one with 60 tetrahedra and the other with 120. Similarly, we simulate the *Bridge* model with $h = 10^{-3}s$, 10 full iterations, 20 reduced iterations and two cubature sets with 50 and 100 elements. Due to different elastic and geometrical properties, the *Dragon* is simulated with $h = \frac{1}{30}s$ and 10 iterations for both quasi-Newton methods. Also, we use two cubature sets with 60 and 120 elements. In all cases, we run only one iteration of the implicit backward Euler method.

Table 5.1 shows the average computational cost per **solver iteration** in each 600-frame long experiment for different solvers and material models. In our reduced method, we do not take into account the full vertices assembly $\boldsymbol{x} = U\boldsymbol{z} + \boldsymbol{x}_0$ that updates the deformable mesh, because it is a rendering issue which may be solved directly on GPU. Vega [23] provides a class that performs this heavy matrix multiplication directly on the *fragment shader*, though our research prototype does not use it (check SceneObjectReducedGPU).

From the invertible StVK data, we see that the reduced solver with a small cubature size achieves an average simulation frame-rate of 124Hz for the *Letter A* model, 141Hz for the *Bridge* and 142Hz for the *Dragon*, against 24.09Hz, 4.54Hz and 0.34Hz of the full solver (respectively). The reduced solver frame-rate is roughly constant because the sizes of the basis and the cubature are preserved. Thus, if we compare to the standard

PD Material		per-iteration time		
model	exp.	full	red. (small cub.)	red. (large cub.)
A	1	1.68 ms	0.21 ms	0.42 ms
A	2	1.60 ms	0.21 ms	0.39 ms
A	3	1.61 ms	0.22 ms	0.40 ms
Bridge	1	10.64 ms	0.17 ms	0.37 ms
Bridge	2	10.67 ms	0.17 ms	0.35 ms
Bridge	3	10.74 ms	0.18 ms	0.37 ms
Bridge	4	11.01 ms	0.18 ms	0.38 ms
Dragon	1	141.87 ms	0.58 ms	0.92 ms
Dragon	2	138.72 ms	0.55 ms	0.84 ms
Dragon	3	133.64 ms	0.48 ms	0.82 ms

Inv. StVK		per-iteration time			
model	exp.	Euler	full	red. (small cub.)	red. (large cub.)
A	1	15.03 ms	4.57 ms	0.41 ms	0.80 ms
A	2	14.67 ms	3.73 ms	0.37 ms	0.81 ms
A	3	14.72 ms	4.25 ms	0.43 ms	0.79 ms
Bridge	1	91.92 ms	23.12 ms	0.42 ms	0.70 ms
Bridge	2	92.18 ms	20.80 ms	0.33 ms	0.64 ms
Bridge	3	91.93 ms	21.87 ms	0.32 ms	0.67 ms
Bridge	4	92.55 ms	22.56 ms	0.35 ms	0.75 ms
Dragon	1	1698.66 ms	298.81 ms	0.74 ms	1.27 ms
Dragon	2	1668.06 ms	291.61 ms	0.71 ms	1.34 ms
Dragon	3	1680.05 ms	293.32 ms	0.67 ms	1.19 ms

Table 5.1: Average per-iteration computation time in different solvers. *Euler* stands for one implicit backward Euler iteration, which is not available for PD materials. *Full* and *red.* correspond to the full and reduced quasi-Newton solvers, respectively. The size of small and large cubatures vary in different models. As the mesh size increases, the methods in full space become highly expensive, while our reduced solver keeps roughly the same cost. This happens because regardless of mesh, we always use a low size basis ($r = 20$ or $r = 40$) and the same cubature sizes. As expected, by doubling the cubature size, the average cost of the full solver increases accordingly.

full solver, the reduced simulation of larger meshes might be accelerated up to hundred times (*e.g*, the Dragon becomes 417 times faster). Yet, one important drawback of both quasi-Newton methods is that the frame-rate is not stable, mainly because the number of line search iterations may change under different conditions (*i.e.*, it increases whenever true energy gradient becomes poorly approximated by the PD gradient).

Regarding quality and realism, although the reduced solver output is generally less locally detailed than the full solver output, their overall motion is similar. Notwithstanding, the reduced simulation can be inaccurate in a few cases. For example, our solver is not able to handle the strong inversion of the *Letter A* in Experiment 3. Also, its oscillation frequency may be lower than the frequency in the other methods. Analyzing all solvers, we notice that both quasi-Newton methods are not as vivid and natural as the implicit backward Euler, but they are less expensive and clearly more robust. Figure 5.2 depicts a

sample frame of the first *Letter A* experiment, where artifacts occur in the implicit Euler animation, while the other methods remain stable, but with low amplitude motion.

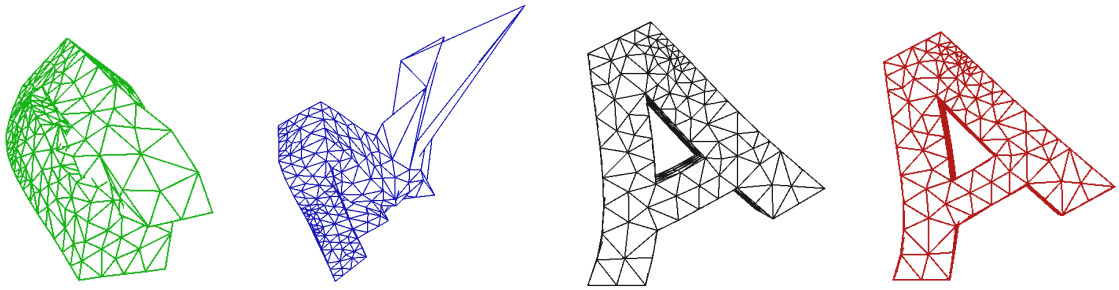


Figure 5.2: *Letter A* model, Experiment 1 (StVK), frame 155 (time-step $h = 10^{-3}s$). From the left to the right, the images depict the ground-truth, the implicit Euler, the full and reduced quasi-Newton, respectively. The ground-truth output animation (green) is more vivid and more locally detailed. However, we notice that the classical implicit integrator (in blue) becomes unstable (explosion). Furthermore, the motion of the quasi-Newton methods is similar, but with a higher damping and smaller deformation amplitudes.

The advantages of the reduced quasi-Newton method arise when simulating the *Bridge* and the *Dragon*. In Figure 5.3, we show a comparative sequence of frames sampled from both quasi-Newton methods running the first dragon experiment. The reduced solver allows us to simulate large deformable objects in real-time even with a CPU implementation only, achieving qualitatively similar results. Therefore, one may say that the loss in visual quality is small for such gain in performance, which the makes trade-off between motion quality and cost balanced. One of its limitations is certainly the required precomputation of a basis and cubature, but in applications such as games, the reduced simulation may be the only approach to fulfill the high frame-rate constraints.

5.3 Numerical Damping Analysis

Given the system velocities $\boldsymbol{v} \in \mathbb{R}^{3n}$ in full-space and the mass-matrix $M \in \mathbb{R}^{3n \times 3n}$, we may calculate the total kinetic energy by $E_k = \frac{1}{2}\boldsymbol{v}^T M \boldsymbol{v}$. External forces exerted on the system vertices increase the strain energy; then, the total energy oscillates between strain and kinetic energy. Ideally, without a specific model for damping, no energy loss should be expected when simulating hyperelastic materials. However, some solvers introduce numerical damping as consequence of the approximation error, which ceases the system motion after a certain time. By measuring the kinetic energy over the animation frames, we can compare the influence of damping in all solvers. Important behaviors to consider are the instability and potential explosions, which lead to sudden peaks of energy even when no stimuli is caused. Apart from these cases, a low rate of kinetic energy loss is a good feature, because it suggests that the method has a low numerical damping.

Since the implicit backward Euler method does not handle projective dynamics materials, we analyze the experiments with invertible StVK materials only. For each 600-frame

Figure 5.3: *Dragon* model, Experiment 1 (StVK), frames 189, 196, 204 (time-step $h = \frac{1}{30}s$). This vertical sequence of images shows how the full solver (in black, on the left) and the reduced solver (in red, on the right) react to a long impulse on the dragon's head. The animations are similar, but our reduced solver (with small cubature) runs about 402 times faster than the standard quasi-Newton.

simulation, we plot the total kinect energy of the system as computed by different solvers under the same input forces. In both quasi-Newton methods, we choose the number of iterations that makes the solver to converge. In our experiments, the full solver required 10 iterations, while the reduced solver required 10 iterations for the *Dragon* model and 20 for the other models.

Figure 5.4 shows the kinetic energy during a typical response of the *Bridge* model to pulling forces in part of the Experiment 3. According to the generated plot, we see that the implicit backward Euler with one iteration achieves the lowest damping, followed by the full solver and then the reduced solver. This is repeatedly observed throughout the experiments. Indeed both quasi-Newton methods have similar behaviors due to their descent direction scheme. However, approximate reduced forces may increase the energy loss because of the cubature error.

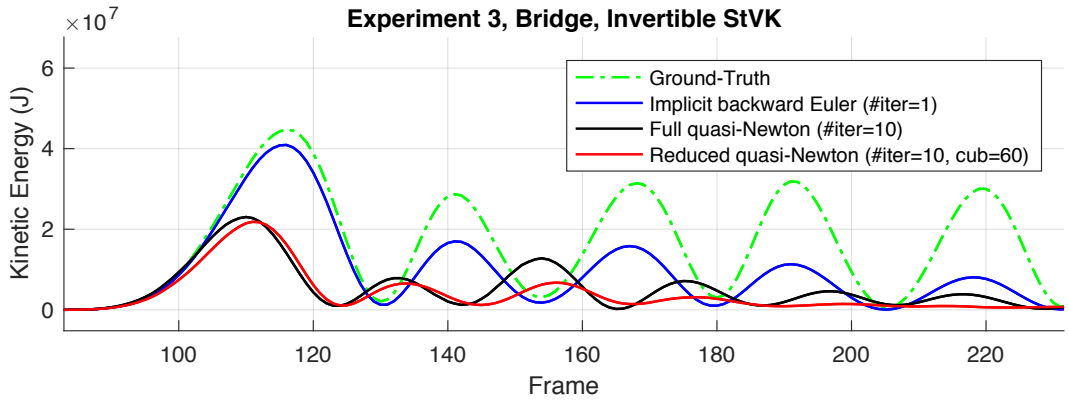


Figure 5.4: Plot of the total kinect energy under constant upwards pulling forces on the bridge mesh (Experiment 3, frames 82-228). Curves are ordered from the largest energy (top) to the smallest (bottom). The oscillation in both quasi-Newton methods loses the amplitude more quickly than in the implicit backward Euler. The output motion of the reduced solver is similar to the full solver, but the oscillation frequency is lower.

In general, every peak on the kinetic energy curve is followed by a local minimum, which indicates the state of maximum deformation amplitude (largest stretching). Perhaps, a good way to visualize the effects of damping is by plotting the peaks of energy (Figures 5.5, 5.6 and 5.7). In all plots, the curves are always ordered from the less damped motion to the most damped. Notice that the ground-truth curve (in green) presents several ripples in its peaks because of its energetic motion (Figure 5.7). As in the example above, the energy curves of our reduced technique are always close to the full quasi-Newton curves. The implicit Euler does not perform well in *Letter A* experiments as consequence of its instability (after explosions, the motion becomes incoherent with the ground-truth). In the following experiments though, it mostly outperforms the quasi-Newton methods in terms of damping.

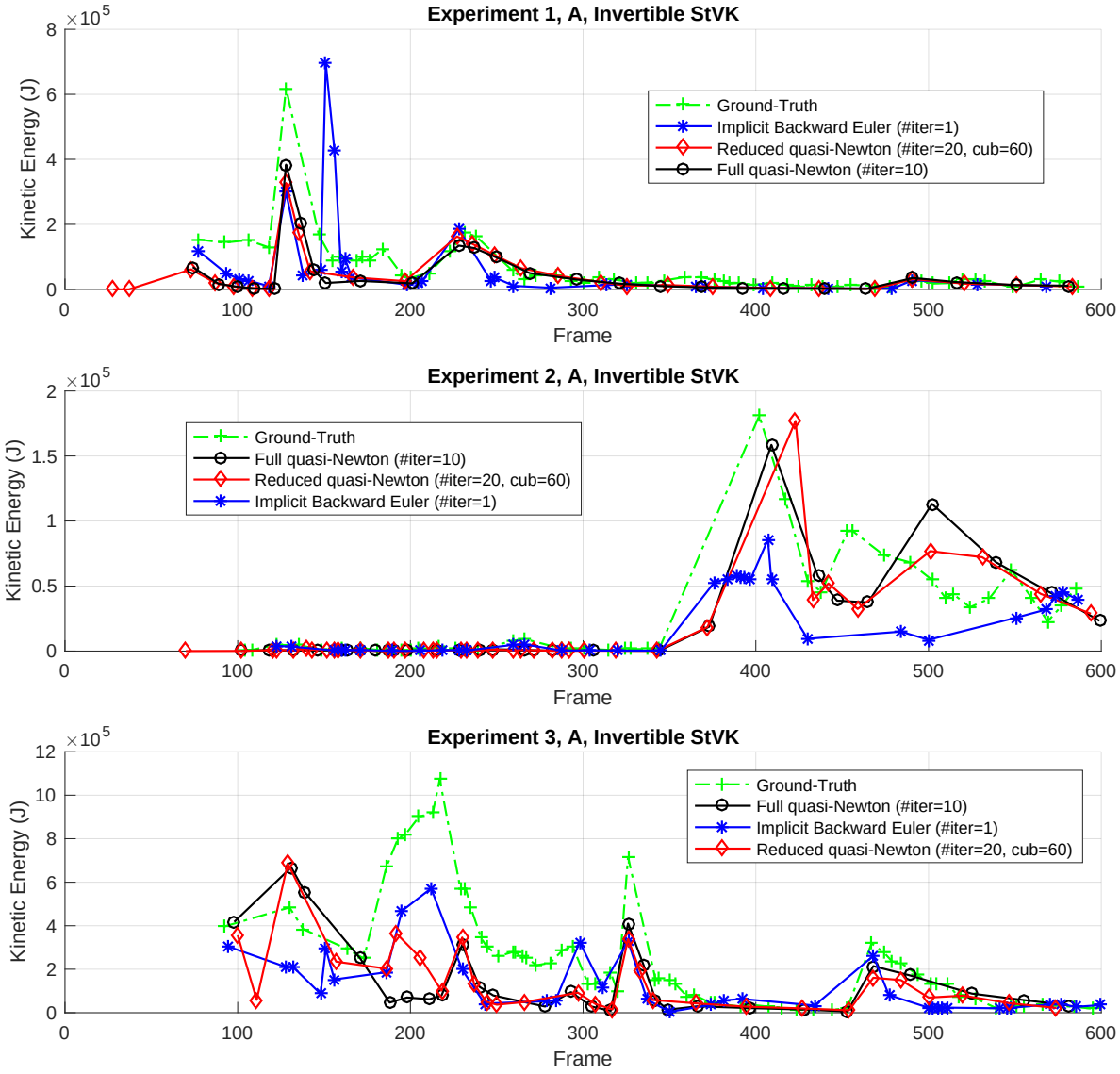


Figure 5.5: Plot of kinetic energy peaks in *Letter A* model experiments. In Experiment 1, the tall peak of energy in blue around frame 150 occurs due to the explosion in the implicit Euler animation.

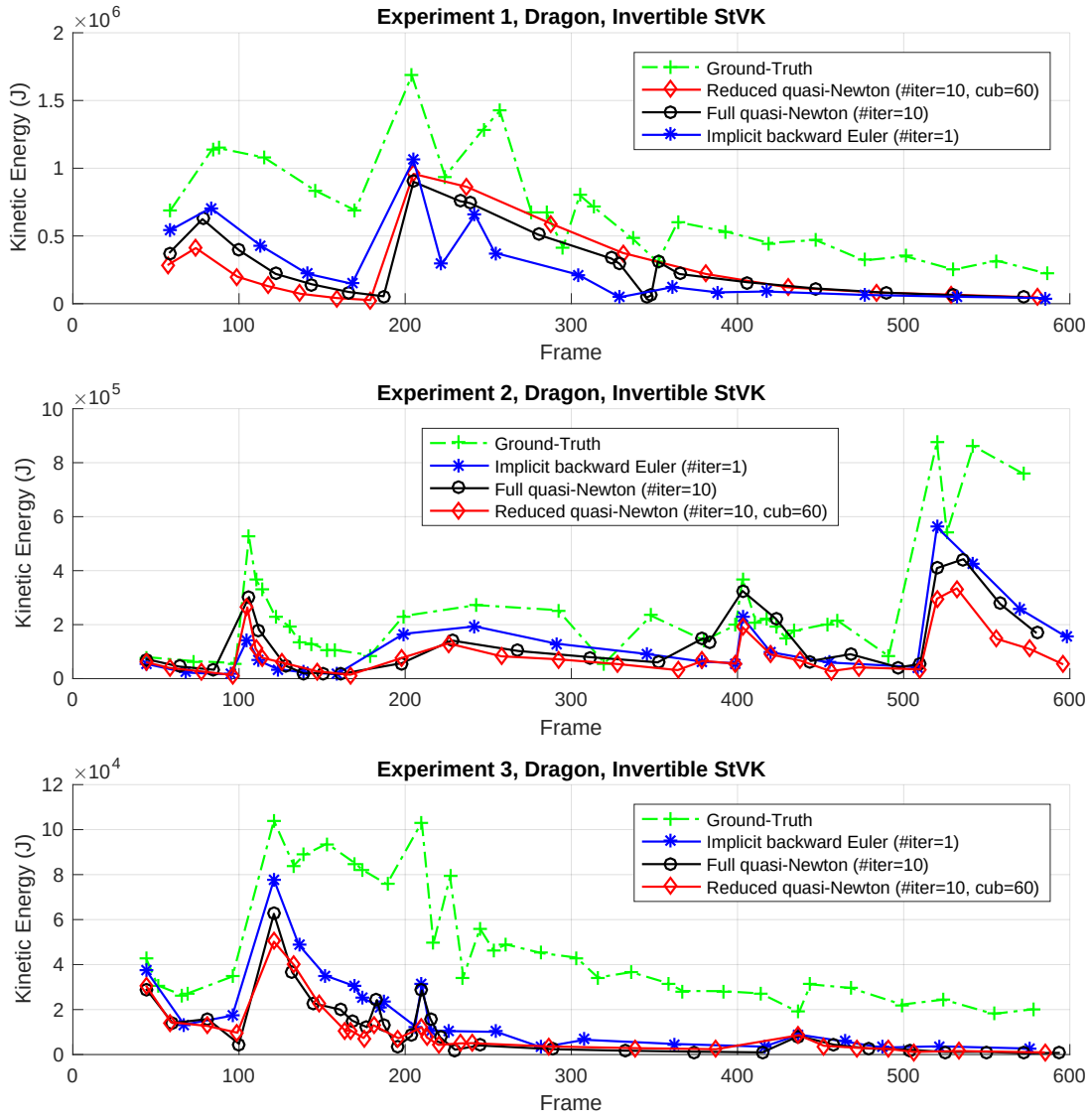


Figure 5.6: Plot of kinetic energy peaks in *Dragon* model experiments.

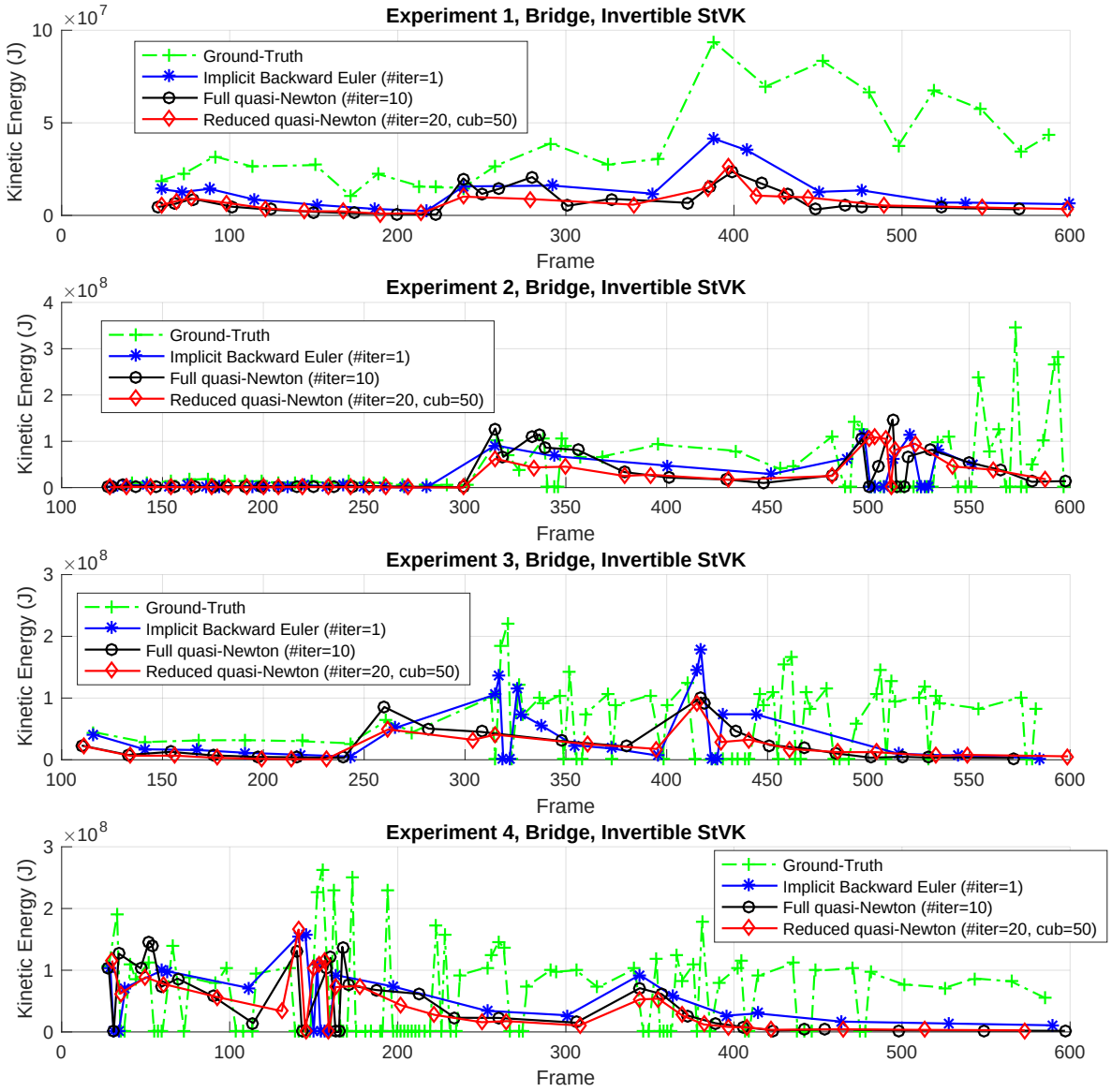


Figure 5.7: Plot of kinetic energy peaks in *Bridge* model experiments. Notice how the ground-truth motion is energetic. Though all methods do not approximate it well, they have close curves.

Chapter 6
Conclusion

In this work, we proposed a new method for deformable object simulation in low-dimensional spaces. The presented solver allows simulating in real-time large meshes that were too expensive for the standard quasi-Newton method, while achieving a balanced trade-off between cost and quality. It may not be able to handle extreme conditions, such as inversions, or its output animation may be more damped than the other methods. However, it can be the most suitable option under high frame-rate constraints. Since our method is built on the top of the standard quasi-Newton, extending the support for model reduction is easy, and no specific elastic force models or tangent stiffness matrices are needed.

Regarding the comparative analyis, we notice that the implicit backward Euler is clearly more accurate than the other methods. Yet it is also more costly and likely to present artifacts under strong stimuli. The projective dynamics gradient is what makes the quasi-Newton method [13] general and simple to implement, because it approximates the true energy gradient of any material model. On the other hand, 10 quasi-Newton iterations are expensive and still far in quality from a single iteration of the classical implicit Euler. Nevertheless, the robustness and simplicity to handle new materials are the best advantages of the full quasi-Newton solver.

6.1 Limitations

Like other reduced methods, the proposed technique requires a basis and a cubature, an additional data that must be precomputed using specific algorithms. Generating such data for large meshes is a process that may take several minutes, and is certainly an undesirable additional step for many applications. Furthermore, the cubature introduces an approximation error in the elastic forces, which slows down the convergence. For this reason, we run twice as much iterations in both *Bridge* and *Letter A* experiments. If too few cubature elements are used, the energy gradient may not be zero-evaluated even at rest configuration. This causes the output motion to present sudden and quick deformations.

6.2 Future Work

As a future work, we would like to compare our solver against the reduced implicit backward Euler. For now, we expect that the reduced quasi-Newton would be outperformed by the reduced Euler in terms of computational cost, because the quasi-Newton methods rely on line search, which requires multiple energy evaluations per time-step. In fact, from our data, we can see that roughly 80% of the total simulation time is spent in line search. A GPU implementation of our solver would explore its highest potential of performance and would also make it fairly comparable to the reduced Euler. For instance, parallelizing the cubature vertices assembly and the deformation gradient decompositions would already provide an additional speed-up of almost 5 times. We estimate that a total GPU implementation can accelerate our technique more than 10 times.

References

[1] Steven S. An, Theodore Kim, and Doug L. James. Optimizing cubature for efficient integration of subspace deformations. *ACM Trans. on Graphics*, 27(5):165:1–165:10, 2008.

[2] Steven S An, Theodore Kim, and Doug L James. Optimizing cubature for efficient integration of subspace deformations. In *ACM Transactions on Graphics (TOG)*, volume 27, page 165. ACM, 2008.

[3] Larry Armijo. Minimization of functions having lipschitz continuous first partial derivatives. *Pacific Journal of mathematics*, 16(1):1–3, 1966.

[4] Uri M. Ascher, Steven J. Ruuth, and Raymond J. Spiteri. Implicit-explicit runge-kutta methods for time-dependent partial differential equations. *Applied Numerical Mathematics*, 25(2):151 – 167, 1997. Special Issue on Time Integration.

[5] David Baraff and Andrew Witkin. Large steps in cloth simulation. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, pages 43–54, New York, NY, USA, 1998. ACM.

[6] Jernej Barbic. *Real-time reduced large-deformation models and distributed contact for computer graphics and haptics*. PhD thesis, Intel, 2007.

[7] Jernej Barbič and Doug L. James. Real-time subspace integration for St. Venant-Kirchhoff deformable models. *ACM Trans. on Graphics*, 24(3):982–990, 2005.

[8] Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly. Projective dynamics: Fusing constraint projections for fast simulation. *ACM Trans. Graph.*, 33(4):154:1–154:11, July 2014.

[9] K. Erleben. *Physics-based Animation*. Charles River Media graphics. Charles River Media, 2005.

[10] Ernst Hairer, Christian Lubich, and Gerhard Wanner. *Geometric numerical integration: structure-preserving algorithms for ordinary differential equations*, volume 31. Springer Science & Business Media, 2006.

[11] Kris K. Hauser, Chen Shen, and James F. O'Brien. Interactive deformation using modal analysis with constraints, June 2003.

[12] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

[13] Tiantian Liu, Sofien Bouaziz, and Ladislav Kavan. Quasi-newton methods for real-time simulation of hyperelastic materials. *ACM Trans. Graph.*, 36(3):23:1–23:16, May 2017.

[14] Sebastian Martin, Bernhard Thomaszewski, Eitan Grinspun, and Markus Gross. Example-based elastic materials. In *ACM SIGGRAPH 2011 Papers*, SIGGRAPH '11, pages 72:1–72:8, New York, NY, USA, 2011. ACM.

[15] Bruce Moore. Principal component analysis in linear systems: Controllability, observability, and model reduction. *IEEE transactions on automatic control*, 26(1):17–32, 1981.

[16] Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. Position based dynamics. *J. Vis. Comun. Image Represent.*, 18(2):109–118, April 2007.

[17] Matthias Müller, Jos Stam, Doug James, and Nils Thürey. Real time physics: Class notes. In *ACM SIGGRAPH 2008 Classes*, SIGGRAPH '08, pages 88:1–88:90, New York, NY, USA, 2008. ACM.

[18] N.M. Newmark. *A Method of Computation for Structural Dynamics*. Number nos. 179-181 in A Method of Computation for Structural Dynamics. American Society of Civil Engineers, 1959.

[19] A. Pentland and J. Williams. Good vibrations: Modal dynamics for graphics and animation. In *Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '89, pages 215–222, New York, NY, USA, 1989. ACM.

[20] Hang Si. Adaptive tetrahedral mesh generation by constrained delaunay refinement. 75:856 – 880, 08 2008.

[21] Hang Si and K Gärtner. 3d boundary recovery by constrained delaunay tetrahedralization. 85:1341 – 1364, 03 2011.

[22] Eftychios Sifakis and Jernej Barbič. Finite element method simulation of 3d deformable solids. *Synthesis Lectures on Visual Computing: Computer Graphics, Animation, Computational Photography, and Imaging*, 1(1):1–69, 2015.

[23] Fun Shing Sin, Daniel Schroeder, and Jernej Barbič. Vega: Non-linear fem deformable object simulator. In *Computer Graphics Forum*, volume 32, pages 36–48. Wiley Online Library, 2013.

[24] Jonathan Su, Rahul Sheth, and Ronald Fedkiw. Energy conservation for the simulation of deformable bodies. *IEEE Transactions on Visualization and Computer Graphics*, 19(2):189–200, February 2013.

[25] Christoph von Tycowicz, Christian Schulz, Hans-Peter Seidel, and Klaus Hildebrandt. An efficient construction of reduced deformable objects. *ACM Transactions on Graphics (TOG)*, 32(6):213, 2013.

[26] Hongyi Xu and Jernej Barbič. Pose-space subspace dynamics. *ACM Trans. on Graphics (SIGGRAPH 2016)*, 35(4), 2016.

[27] Hongyi Xu, Funshing Sin, Yufeng Zhu, and Jernej Barbič. Nonlinear material design using principal stretches. *ACM Trans. on Graphics (SIGGRAPH 2015)*, 34(4), 2015.