Accelerating Data-driven Simulations for Deformable Bodies and Fluids

Dissertation

Presented in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy in the Graduate School of The Ohio State University

By

Rajaditya Mukherjee, B.C.S.E, M.S

Graduate Program in Computer Science and Engineering

The Ohio State University

2018

Dissertation Committee:

Huamin Wang, Advisor

Tamal K Dey

Eric Fosler-Lussier

Nima Ghalichechian

Abstract

Subspace Simulation or Model reduction is a technique to simplify simulations of systems modeled by Ordinary Differential Equations. The essential idea is to project the high dimensional sparse equations on to lower dimensional dense equations and then re-projecting the solution back to the original space. Such a method has a much faster computation and lower memory footprint as compared to a full space simulation. Recalculating the subspace basis of a deformable body is a computationally expensive yet mandatory operation. We show that the subspace of a modified body can be efficiently obtained from the subspace of its original version if mesh changes are small. Our basic idea is to approximate the stiffness matrix by its low-frequency component, so we can calculate new linear deformation modes by solving an incremental eigenvalue decomposition problem. We also show a hybridized approach to calculate the modal derivatives and finally we demonstrate that the cubature samples trained for the original mesh can be reused in fast reduced force and stiffness matrix evaluation.

Next, we handle the problem of inverse simulation in deformable bodies. We present a novel system for interactive elastic shape design in both forward and inverse fashions. Using this system, the user can choose to edit the rest shape or the quasistatic shape of an elastic solid and obtain the other shape that matches under the quasistatic equilibrium condition at the same time. The development of this system is based on the discovery that inverse quasistatic simulation can be immediately solved by Newton's method with a direct solver.

To implement our simulator, we propose a Jacobian matrix evaluation scheme for the inverse elastic problem and we present step length and matrix evaluation techniques that improve the simulation performance. While our simulator is efficient, it is still not fast enough for the system to generate the result in real time. Our solution is a shape initialization method using the recent projective dynamics technique. Shape initialization not only works as a fast preview function during the user editing process, but also speeds up the convergence of quasistatic or inverse quasistatic simulation afterwards. The use of a heterogeneous algorithm structure allows the system to further reduce its preview cost, by utilizing the power of both the CPU and the GPU. Our experiment demonstrates that the whole system is fast, robust, and convenient for the designer to use in both forward and inverse elastic shape design. It can handle a variety of nonlinear elastic material models, and its run-time performance has space for more improvement.

Finally, we look at the problem of fluid simulation. Small-scale liquid flows on solid surfaces provide convincing details in liquid animation, but they are difficult to be simulated with efficiency and fidelity, mostly due to the complex nature of the surface tension at the contact front where liquid, air, and solid meet. In this section, we propose to simulate the dynamics of new liquid drops from captured real-world liquid flow data, using deep neural networks. To achieve this goal, we develop a data capture system that acquires liquid flow patterns from hundreds of real-world water drop experiments. We then convert raw data into compact data for training neural networks, in which water drops are represented by their contact fronts only. Based on LSTM, our neural networks serve three purposes in our system: predicting the contour of a contact front, predicting the color field gradient of a contact front, and finally predicting whether a contact front is going to break or not. Using these predictions, our simulator recovers the interior shape of a water drop at each time step

and handles merging and splitting events by geometric operations. The experiment shows that our trained neural networks are able to perform predictions well. The whole simulator is robust, convenient to use, and capable of generating realistic small-scale liquid effects in animation.

Dedicated to my parents, Tapas and Soma Mukherjee, and my sister, Koyel Kaushal

Acknowledgments

I would like to thank my advisor Dr. Huamin Wang for his constant guidance, encouragement, and inspiration. I would also like to thank my committee members Dr. Tamal K Dey and Dr. Eric Fosler-Lussier for their ideas and feedback. I'm grateful to my colleagues and collaborators at The Ohio State University for their valuable thoughts and frequent research discussions that made research a pleasant experience. I would like to thank my industry collaborators for bringing in new perspectives to my research. I'm grateful to my friends who made life at Columbus, Kolkata and Bangalore enjoyable. Finally, I'm indebted to my family for their constant encouragement, love, and support through out my doctoral studies. My work is supported in part by NSF grant CHS-1524992 and gifts from Adobe and NVIDIA. The opinions and findings of this thesis are those of the author, and collaborators (where ever applicable) and do not necessarily reflect the views of the National Science Foundation or The Ohio State University.

Vita

May 2018	Ph.D., Computer Science and Engineering, The Ohio State University, USA.
May 2017	M.S, Computer Science and Engineering, The Ohio State University, USA.
August 2011	B.C.S.E, Computer Science and Engineering, Jadavpur University, India.

Publications

Research Publications

Rajaditya Mukherjee, Longhua Wu and Huamin Wang. Interactive Two-Way Shape Design of Elastic Bodies In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (i3D 2018)*

Rajaditya Mukherjee, Xiaofeng Wu and Huamin Wang. Incremental Deformation Subspace Reconstruction In *Computer Graphics Forum (Pacific Graphics 2016)*

Xiaofeng Wu, Rajaditya Mukherjee and Huamin Wang. A Unified Approach for Subspace Simulation of Deformable Bodies in Multiple Domains In *ACM Transactions on Graphics (SIGGRAPH Asia 2015)*

Subhashis Hazarika, Tzu-Hsuan Wei, Rajaditya Mukherjee and Alexandru Barbur. Visualizing the life and anatomy of dark matter In *SciVis 2015 (Visualization Contest)*

Fields of Study

Major Field: Computer Science and Engineering

Studies in:

Computer Graphics	Prof. Huamin Wang
Theory and Algorithms	Prof. Tamal K Dey
Artificial Intelligence	Prof. Eric Fosler-Lussier

Table of Contents

Page

List of Tables

List of Figures

Chapter 1: Introduction

Physics based simulation is one of the most important subareas of research in computer graphics. Its contributions are ubiquitous, spanning the fields of digital movie-making, visual effects industry, virtual reality, medical and healthcare and more recently computational fabrication. It is responsible for making stunningly real clothing on animated characters with complex drapes and folds; modeling highly realistic smoke, fire and explosion effects in a live-action movie; helping future doctors practice their skills on surgery simulators with haptic feedback and making believable worlds in virtual reality headsets.

Pioneering this field was the work by Terzopoulos et al. [TPBF87] almost 20 years back who showed that continuum mechanics can be used for simulation of deformable bodies. Physics-based simulation in computer graphics sits at the intersection of mathematics and physics. Standard techniques in mechanics such as Finite Element Method [OH99], Boundary Element Method [JP99] and Smoothed Particle Hydrodynamics for fluid simulations [MCG03] has been incorporated into mainstream computer graphics. Backed by the mathematical foundations of the principles of linear algebra, this field incorporates the accuracy of solid mechanics with the adaptability of mathematical techniques to create the aforementioned stunning visuals.

From an artistic perspective, it is often beneficial to favor speed of simulations over accuracy. This is because in most production work-flows, any character or object animations

have to go through several iterations of prototyping before it is ready for final production. In such a prototyping stage, it is beneficial to see how animations would look and feel and interact with other assets in order to have a full understanding of its standing in the scene. Hence, a fast simulation at the cost of small accuracy is desirable in this situation.

While a myriad number of algorithms exists for accelerating simulations of deformable bodies and fluids, we focus on those that leverage past data as the primary insight for acceleration of such physics-based simulation. More specifically, we look at mathematical intuitions behind data-driven simulations and find ways to accelerate them. We leverage linear algebra insights, deep learning, and convex optimization to find novel ways to accelerate efficient simulations of deformable bodies and fluids. More specifically our contributions can be divided into three specific fields of physics based simulation of deformable bodies and fluids as listed below.

1.1 Subspace Simulations

For high precision mission-critical problems such as those in structural mechanics, the solution of object deformation is solving partial differential equations using high-quality offline solvers such as Ansys' Mechanical Solutions. Such a solution trades off speed in favor of accuracy. However, in computer graphics, we are more interested in speed of simulation as compared to accuracy. This motivation stems from the workflow requirements in animation. Deformable objects in computer graphics are highly transient and only persist for a short period in the observer's eyes. Also compared to traditional structural mechanics, casual observers are more forgiving of small inaccuracies in physical simulations provided there are no ungainly visual artifacts. Hence the problem of interactive object deformation becomes critical in computer graphics. Many possible solutions exist for the interactive deformation

problem such as linearization of the underlying models [BNC96] , simplified Co-Rotational Models [MG04] , Subspace simulations [BJ05], Projective Dynamics [BML$^+$14] , Multi-resolution simulations [GKS02] and Multi-Grid simulations [ZSTB10, SYBF06]. In this portion of the thesis, we will focus on Subspace Simulations.

The problem of subspace simulation (or *model reduction* as called in classical mechanics literature) can trace its root back to the works of [KLM01].Subspace simulation was pioneered in computer graphics by [BJ05]. It works by projecting the high dimensional ordinary differential equations on a smaller dimensional subspace thereby reducing the number of run-time computations. To compute a set of projection basis that doesn't produce ugly simulation artifacts is a non-trivial and time-consuming process. In the first part of my thesis, we show that if the parts of the deformable bodies are assumed to be connected by zero rest length springs, we can easily achieve speedups of 10-30x magnitude in the recomputation of this simulation basis. These speedups are achieved by exploiting the fact that addition of such zero-rest length springs only influences the low-frequency vibration modes of the body and if we are willing to sacrifice the high-frequency modes, the resulting animation is visually similar to a full-space simulation but with the added advantage of fast basis recomputation that greatly enhances its usability. We show a myriad number of uses for this method with applications spanning virtual surgery, multi-domain simulations, and multi-material simulations to name a few.

1.2 Inverse Shape Design on GPU

The field of computational fabrication has seen an explosive growth in recent times with the proliferation of 3D printers that span all price range and all dimensions. No longer an exotic prototyping and fabrication tool accessible to only the biggest research laboratories,

3D printers are available for hobbyist and home use by companies like MakerBot and Formlabs for less than $3000. The field of computational fabrication is not only innovating in terms of form and size but also in terms of materials that can be printed. Initially, 3D printers could only print hard plastic polymers such as ABS (Acrylonitrile Butadiene Styrene). However, the latest 3D printers are capable of printing thermoplastic elastomers that simulate rubber-like behavior which is much more compliant as compared to ABS.

Such soft materials are susceptible to the effects of gravity post printing when the supporting materials are removed. Under the influence of gravity, the printed object deforms and produces a shape which may be drastically different from the original mesh depending on the compliance of the materials. Hence it is beneficial to look into the problem of inverse shape design which tries to predict what rest shape can produce a target shape under quasi-static simulation in the presence of external loading such as gravity. One of the best-known approaches to this problem is using Asymptotic Numerical Methods [CZXZ14]. In this portion of the thesis, we solve this problem using Newton's method. We offset the costly Hessian calculation in Newton's method by an intelligent guess using Projective Dynamics which reduces the number of iterations needed. These guesses are computed in parallel using GPU during the preview stage so no extra processing cycles are spent. The result is a robust, interactive two-way shape design simulator that works on a wide range of non-linear material models.

1.3 Small Scale Fluid Simulations

Most of the techniques for fluid simulations in Computer Graphics are derived from the field of Computational Fluid Dynamics with earlier works such as those by [Sta99, EMF02]. Most of the approaches to simulating fluids are either grid-based (Eulerian) or

particle-based (Lagrangian) although in recent times, the computer graphics community favors hybrid approaches such as FLIP (Fluid Impact Particles) that takes the best of both worlds [JSS+15]. However, note that in such hybrid models, the transformation between different representations often creates discretization errors that impact the overall visual flow of the fluid simulation. This becomes even more complicated in small-scale liquid flows where interactions between the solid and the fluid become important. Contact angles and surface tension between the fluid-surface interface play an important role in the final result. Physics-based methods such as [WMT05] usually enforce this using level sets in addition to solving the shallow wave models.

In this portion of the thesis, we approach small-scale fluid simulation with a fresh approach. We model the non-linear nature of fluid front advection using a deep neural network. More specifically, we use LSTM (Long Short Term Memory) [HS97] to model the advection of the fluid front as a spatiotemporal function. We collected a large amount of real-world fluid flow data using an automated capture device, processed it using tracking and computer vision to extract the outline and infer the shape and volume finally and used it to train a deep neural network to accurately model small scale fluid flow. The result is a robust, fast, easy-to-use simulator that can be used in conjunction with other fluid simulation techniques such as FLIP.

Chapter 2: Incremental Deformation Subspace Reconstruction

2.1 Introduction

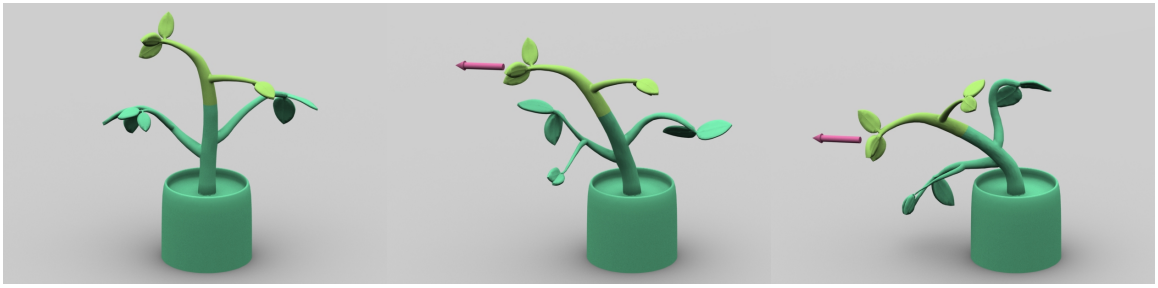
Many real-time simulation applications demand interactive editing on a deformable body. In virtual surgeries, surgeons perform cutting and stitching operations to modify the connectivity of a soft tissue. During computer-aided design, designers and artists adjust the stiffness and the size of a 3D model to produce its optimal shape. These topological and geometric editing operations should be efficiently handled on the fly, without noticeable lags that compromise user's experience.

Subspace deformation, also known as *dimensional model reduction* or *reduced-order deformation*, has been an important graphics research topic in recent years, thanks to its ability to significantly accelerate deformable body simulation. The basic strategy of this technique is to restrict the deformation into a subspace, so that the dynamical system is reduced from the full space to the subspace, in which the system solution can be found in real time. Although subspace simulation is fast, the computational cost spent on calculating the subspace is often non-trivial, which may take several seconds or even minutes for a high-resolution mesh. For applications that do not involve interactive editing on the rest shape of a deformable body, this is not an issue since the subspace can be calculated ahead of time as a pre-computation step. However, if an application requires to interactively

update the rest shape, the subspace recalculation cost cannot be ignored. Without subspace recalculation, the simulation result suffers from the locking issue as shown in Figure 3.1b, because the original subspace does not offer enough freedom for the model to deform.

For faster subspace recalculation, one natural idea is to divide the mesh into multiple domains and build the subspace of each domain separately. If the interface between two domains is small, each domain can be simulated in its own subspace with the interface implemented as a rigid constraint [BZ11]. If the interface is large, its deformation cannot be ignored and it can be incorporated into the subspace simulation of each domain as hard constraints by Lagrangian multipliers [HLB+06]. Since the subspaces may not be consistent, hard constraints often suppress subspace deformations. To address this issue, Kim and James [KJ11] replaced hard constraints by soft spring constraints, which allow two domains not to be strictly aligned at the interface. Yang and colleagues [YXG+13] developed a boundary-aware method to make the subspaces more consistent at the interface. To do so, their method incorporates more deformation modes into the subspace and it considers linear modes only. [HZ13] handles local deformation by supplementing subspace basis with additional basis functions. Multidomain subspace based simulations were also used in [ZB13] to demonstrate the tuning of stiffness in an arbitrary domain in a real-time setting. Recently, Wu and collaborators [WMW15] proposed to simulate rigid motions and subspace simulations of all of the domains under a unified framework, to better eliminate artifacts near the interface. In general, existing domain decomposition techniques still suffer from a variety of limitations, and there exists no effective way to fully address the interface issue yet.

Different from recent research [vTSSH13, YLX+15] on expediting the subspace construction process itself, our work tries to study the problem in an incremental fashion.



(a) Two domains (b) Stitched body simulated in the original subspace (c) Stitched body simulated in the new subspace

Figure 2.1: The plant example. Our system can quickly recalculate the subspace of a deformable body, after it gets modified by simple stitches, i.e., zero-length springs. The new subspace allows the two domains in (a) to be connected and simulated as shown in (c). In contrast, if the simulator uses the deformation subspace constructed for the original mesh before stitching happens, it will produce the locking issue due to the inconsistency of the subspace, as shown in (b).

Specifically, after a mesh gets modified by some small changes, can we quickly obtain its new subspace from the original subspace? Here the mesh can be a connected mesh, or a mesh with disjointed domains whose subspaces can be combined into a single subspace. By incorporating mesh changes into the recalculated subspace, we do not need to apply additional constraints in runtime simulation. Our work is based on the assumption that the subspace can be efficiently and incrementally calculated, after a small number of changes each time. To achieve this goal, we made the following technical contributions.

- **Recalculation of linear modes.** Given the existing linear subspace of a deformable body, we show how its linear deformation modes can be quickly updated through incremental eigenvalue decomposition, when the body is modified by a single stitch, i.e., a zero-length spring.

- **Recalculation of modal derivatives.** Barbič and James [BJ05] introduced the use of modal derivatives to handle nonlinear and large deformations. By avoiding the expensive recalculation of Hessian stiffness tensor and reducing the number of tensor-vector-vector evaluations, we present a fast approach for generating modal derivatives of the stitched body.

- **Cubature-based subspace simulation.** We demonstrate the use of our new deformation modes in subspace simulation. Since mesh changes are small, reduced forces and matrices can still be evaluated by original cubature approximation.

Figure 3.1c illustrates that the new subspace, generated by our techniques in two seconds, can realistically handle large and nonlinear deformations in simulation. Our techniques are especially fast when the number of stitches is small. In case the deformable body undergoes too many changes, we can still use other approaches [BJ05, YLX+15] and build the subspace from scratch.

We can envision a variety of usages for our methods as we have illustrated through examples : Changing Material Stiffness such as the Oball Example (Figure 2.12) ; stitching applications in virtual surgery simulations such as the Stomach Example (Figure 2.17); Multidomain simulations such as the Plant Example (Figure 3.1, 2.16) and complex deformation scenarios such as the Dinosaur Example (Figure 2.10c, 2.15).

2.2 Related Works

Subspace simulation. The history of subspace simulation in engineering and mathematics can be traced back to [Lum67]. In computer graphics, Pentland and Williams [PW89] used linear modal analysis to build a subspace and developed the subspace simulation of a dynamical system for the first time. Compared with linear deformation,

nonlinear deformation is much more difficult to handle in a subspace. A straightforward approach is to compute the deformation modes around several states of the physical system and use them to span the subspace. Specifically, this approach requires to solve several large eigenvalue problems, which are too expensive in practice. Researchers in engineering and graphics [IC85, BJ05, HSTP11, Tis11] have advocated the use of modal derivatives to enrich the subspace basis. The idea behind modal derivatives is to calculate the natural second-order system response to large deformation around the rest shape, so that they can approximate nonlinear deformation better when incorporated into the basis. Recently, von Tycowicz and collaborators [vTSSH13] proposed to widen the subspace by applying affine transformations to each linear mode. Their method is fast and easy to implement, but it creates a larger subspace that cannot be easily reduced afterwards. Subspace simulation of nonlinear deformation is relatively easier to handle, if the subspace is built from a set of deformed shape examples [KLM01]. Kim and James [KJ09] suggested to build the subspace on the fly, using previously simulated results as example data. In this work, we prefer not to recalculate the subspace in an example-based style, since we cannot afford running full-space simulation and we do not know user interaction ahead of time.

Subspace simulation can be several orders of magnitude faster than full-space simulation. But if the mesh resolution is high, the evaluation of reduced forces and Jacobian matrices is still costly and can be the bottleneck of the system. Barbič and James [BJ05] found that the internal forces of a geometrically nonlinear material can be expressed as cubic polynomial functions of reduced space coordinates. So they proposed to precompute the polynomial coefficients and quickly evaluate reduced forces and matrices. Instead of performing the evaluation exactly, An and colleagues [AKJ08] used cubature approximation to estimate

reduced forces and matrices. Their method effectively reduced the evaluation cost to $O(r^3)$, when the number of deformation modes is r and the number of cubature points is $O(r)$.

Other problems. When simulating a deformable body in the subspace, an interesting question is whether we can handle its self collisions in the subspace as well. Assuming that an object cannot be in self collision if it does not deform much, Barbič and James [BJ10] developed “certificates" to quickly avoid unnecessary collision tests using reduced subspace coordinates. Teng and colleagues [TOK14] proposed a pose-based cubature scheme, so they can detect self contacts of an articulated body, without explicitly checking the collision of two primitives. Recently, Sheth and colleagues [SLYF15] presented a skinning-based framework to ensure momentum conservation in subspace simulation, even in the presence of collisions and contacts.

Graphics researchers have also explored the use of subspace simulation in other fields. Hahn and colleagues [HTC+14] used a pose-varying subspace basis to simulate detailed clothes dressed on a human body. Xu and collaborators [XLCB15] developed an interactive tool to design heterogeneous material properties of a 3D model, based on subspace simulation techniques. Researchers [TLP06, WST09, KD13, ATW15] have also studied subspace fluid simulation.

2.3 Background

2.3.1 Deformable Object Simulation

Substantial research has been done by the engineering community in the field of simulation of continuum behavior of deformable objects. Most of this research takes place in the field of Finite Element Methods (FEM) which form the backbone of a number of commercial simulators such as those by ANSYS which is primarily used in product design as well as in

structural and engineering applications. However the extreme precision afforded by these commercial products usually have a large simulation runtime associated with them and hence are unsuitable for real-time graphics applications where we can trade some of the precision for speed.

This section is structured as follows : We first offer an abbreviated primer of the mechanics of materials and then offer a brief introduction to Finite Element Methods and their usage in Computer Graphics. We will then introduce *Subspace Simulation* - a technique that can reduce the runtime cost of simulations by orders of magnitude. We then explore how subspace simulation can be applied in more complex situations by coupling it with rigid motions. Finally, we explore multi-domains subspace simulations which breaks up the entire mesh into non-intersecting domains, performs subspace simulation on each individual domains and returns the geometrically contiguous result.

2.3.1.1 Mechanics of Materials

When any deformable object is subjected to an external force, internal elastic forces will appear which will try to restore the object back to its rest shape. In order to simplify our material models, we make a locality assumption about the nature of deformations. More specifically, the deformation at a point is only dependent on the deformations in some infinitesimal small local neighbourhood of the point.

We now define some terms commonly found in any solid mechanics literature such as [Sha90] . All these terms pertain to quantify the local deformation neighborhood at a point. Define the deformation vector $u = u(X) \in \mathbb{R}^3$ as the displacement of a point in the body from its rest position X under the influence of an external force. More specifically the deformed position X is captured by a deformation function $\phi(X)$. The mapping between rest position and the deformed position as a function of the deformation function is shown in Fig. 2.2.

However this is not a good measure of the deformation field since it is not invariant to affine transforms. Define the gradient of the deformation function (called *Deformation Gradient* $\mathbf{F}$) as $\mathbf{F} = \frac{\delta \phi(X)}{\delta X}$ or alternatively $\mathbf{F} = \mathbf{I} + \frac{\delta u}{\delta X}$. Finally we define the *Green-Lagrange Strain* as $\mathbf{E} = \frac{1}{2}(\mathbf{F}^T\mathbf{F} - \mathbf{I})$. Note that the Green-Lagrange is a non-linear measure of strain compared to its Linear Counterpart which is known as *Cauchy Strain* defined as $\varepsilon = \frac{1}{2}(\mathbf{F}^T + \mathbf{F}) - \mathbf{I}$. This distinction is particularly important to observe since the choice of the linearity/non-linearity of the stress tensor is crucial to the simulation behavior. Obviously $\mathbf{E}$ (or ε) $\in \mathbb{R}^{3\times3}$. Finally, note that strain is a purely geometrical property and not a material property.

Now we will attempt to quantify the internal restoring force acting on the body. More specifically we try to define the *stress* acting in the body. Consider the internal force acting on an infinitesimally small region around the material point X. The force acting per unit volume (also called *force density*) is quantified as $f^{int} = \nabla \cdot \mathbf{P}$. Here the quantity $\mathbf{P}$ is the aforementioned stress, more specifically *First Piola-Kirchoff Stress*.

Having defined both stress and strain from first principles, we are now in a position to describe the relationship between them. The relationship between stress and strain of a material is a function of the material properties and is known as *Constitutive Law* for a particular material. The choice of the Constitutive law defines the behavior of the object when subjected to external deformation forces. There exists a varied number of such laws. Interested readers are referred to [BW08] for a concise description on the some of the relevant constitutive laws in mechanics. However each constitutive law has their own advantages and disadvantages depending on the simulation scenario in question. For instance Linear Elasticity is advantageous for its simplicity but it is unable to model large deformations. The computer graphics community is particularly interested in two non-linear models - *St.Venant-Kirchoff* and *Neohookean* elasticity. The characteristic curves for both

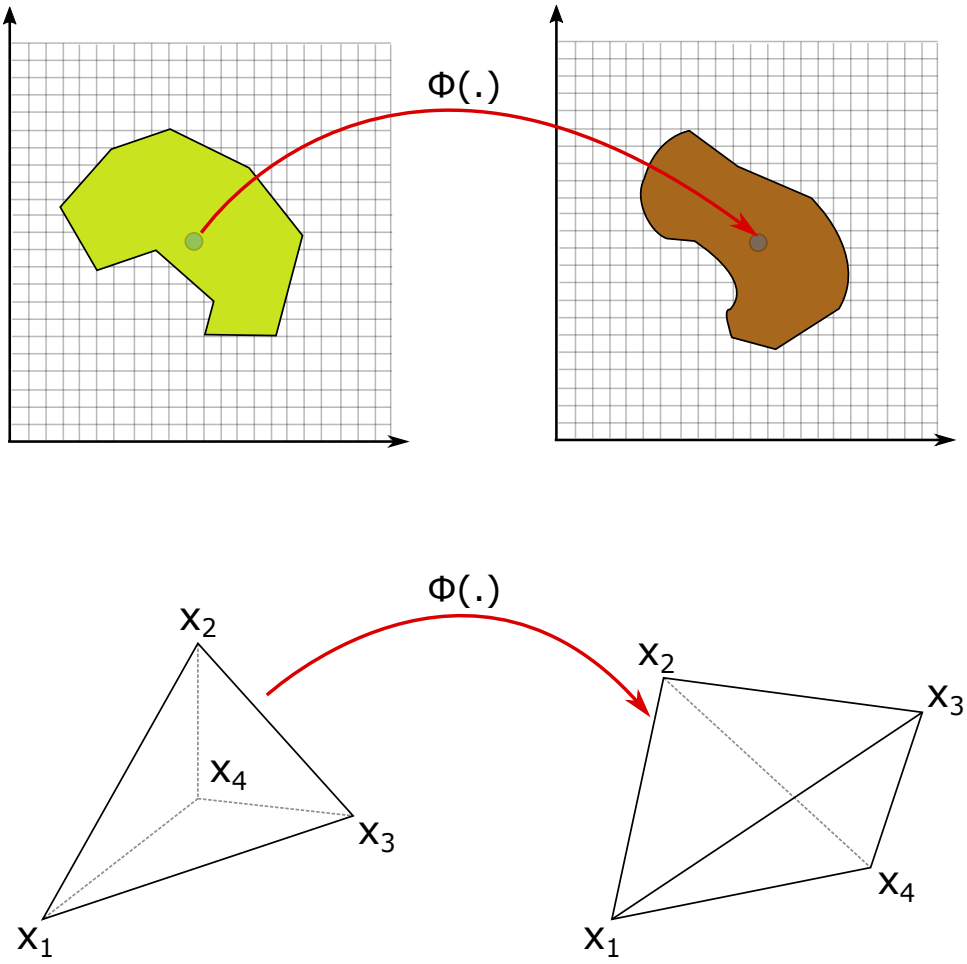


Figure 2.2: The definition of deformation mapping as applied to a arbitrary domain in $\mathbb{R}^2$ and then shown in a deformed linear tetrahedron in $\mathbb{R}^2$. The figures on the left denote the rest shapes and the figures on the right denote the deformed state under the influence of the deformation mapping ϕ.

of them is shown in Fig. 2.3. Both of these models can model large deformations however they are complicated to implement and substantial work [ITF04] has been done to simplify the computations of the matrices needed to implement them.

Finally we define some additional terms which are indispensable to the study of mechanical models. Under the application of an external force, work is done on a deformable body. While portions of this work is dissipated as material damping or converted into deformable kinetic energy, a major portion of it is converted into the *internal strain energy* of the deformable body. A class of materials for which both the deformation gradient as well as the internal strain energy are dependent only on the present configuration (that is, the deformation history is immaterial when quantifying these values) are known as *Hyperelastic* Materials. A hyperelastic material is *isotropic* if the material properties at any arbitrary location in the body is same regardless of its local rotational orientation. The computer graphics community is particularly interested in Isotropic Hyperelastic Materials owing to their simplicity and ability to accurately represent large deformations. We investigate one such model in the next section.

Neohookean Elasticity Isotropic Neohookean Elasticity is a non-linear constitutive model commonly used in deformable object simulation. The Isotropic Neohookean model provides the following relationship between the stress $\mathbf{P}$ (more specifically *First Piola Stress*) and the Deformation Gradient $\mathbf{F}$ as follows:

$$\mathbf{P} = \mu(\mathbf{F} - \mu\mathbf{F}^{-T}) + \lambda \log(J)\mathbf{F}^{-T} \tag{2.1}$$

Note that while this relationship doesn't explicitly model between stress and strain, it is equivalent since as we discussed, the strain is basically a local geometric function of the deformation gradient.

Now we discuss the meaning of some of the symbols used here. μ and λ are the *Lamé coefficients*, which relate to the material properties of the object . J here refers to the determinant of the Deformation Gradient $\mathbf{F}$.

The main characteristics of Neohookean solids that distinguish it from similair other models is the fact that due to the logarithmic term, Neohookean solids are extremely resiliant to extreme compression. That is by construction, the model disallows $J = 0$. However in practice, such a situation can arise owing to inaccuracies in numerical solver or instability of time integration schemes and so special methods such as [TSIF05], [ITF04] deal with such scenarios.

2.3.1.2 Finite Element Method

Continuum mechanics is a branch of mechanics that describes the kinematics and the mechanical response of a deformable body when subjected to an external force. Usually this behavior is governed by a set of Partial Differential Equations (PDE). Except for extremely simple geometry, closed form solutions generally don't exists for these PDE. Hence the usual workflow is to convert these PDEs in continuum into the so called "semi-discretized" Ordinary Differential Equations in time by applications of techniques like Finite Element Method, Finite Difference Method or Finite Volume Method. Then these ODEs are fully discretized in the time domain by using some kind of numerical time-stepping scheme (such as Forward or Backward Euler) and the resulting discrete solution in time and space. In this tutorial, we will focus on Finite Element Method for Semi-discretization of the continuum PDE and Backward Euler for fully discretization of the resulting ODE.

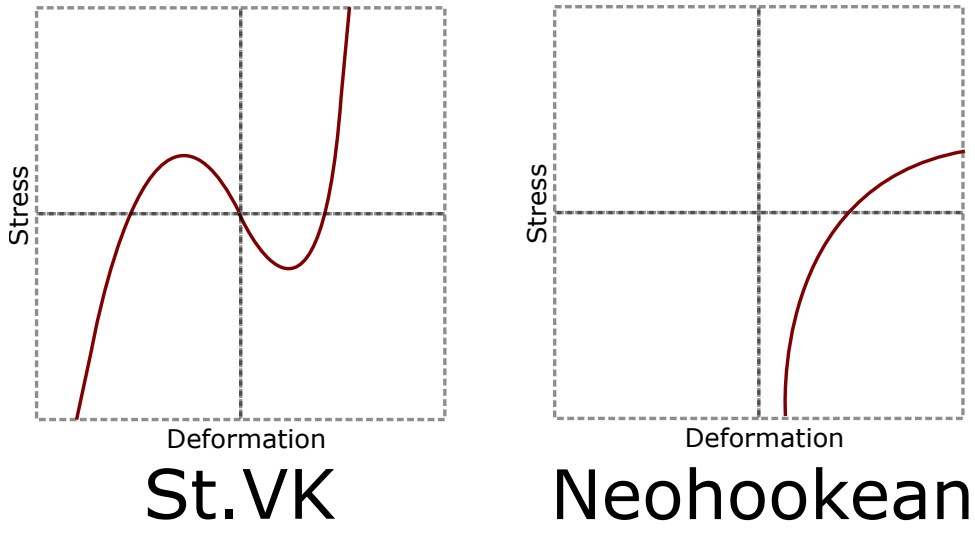


Figure 2.3: In St.VK materials, as the stress increases,it becomes weaker and weaker till it becomces zero as the object is completely flatteend. Beyond that point, inversion occurs and it remains that way. Compare that to the Neohookean model where there is no inversion (no part goes to the left hand side of the deformation axes which represents negative deformation or object inversion).

Discretization Consider any physical process governed by a PDE (in this thesis we consider the particular case of Elasticity) but the discussion remains relevant for any physical process governed by a PDE. The basic idea is to subdivide the continuous domain into simpler discrete elements and then obtain the solution over the whole domain by piecing together the individual solutions.

The elements to be subdivided can be of varied types such as hexahedral, cubic or tetrahedral as shown in Fig. 2.4. In this thesis, we will mostly focus on tetrahedral elements. The degree of deformation is controlled by the displacements at the vertices of the spatially discretized mesh. Also we need a specific interpolation function that can interpolate the individual discrete domain displacements along any point in a continuous domain. We call such an interpolation function in the context of FEM, a *shape function*.

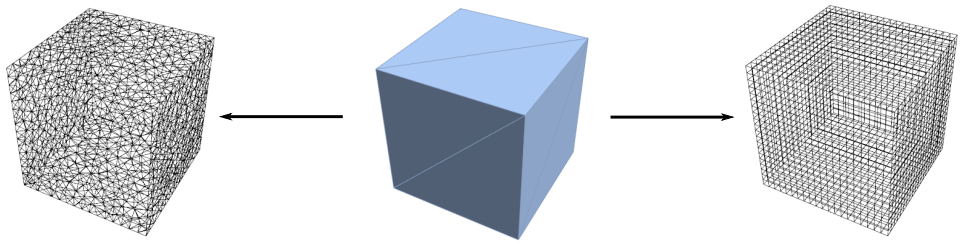


Figure 2.4: This figure shows two possible discretization of a simple cube. On the left is a tetrahedral discretization while on the right we have an equivalent hexahedral discretization.

After addition of the shape functions into the PDE system, we obtain a system of ODEs. At this point, we have successfully discretized the function in space but not in time. Note that should the discretization be such that the resulting mesh has N vertices, the resulting system of ODE has the following form :

$$\mathbf{Ma} + \mathbf{Dv} + \mathbf{f}^{\text{int}}(\mathbf{u}) = \mathbf{f}^{\text{ext}}, \tag{2.2}$$

where the symbols have the following meaning : $\mathbf{u} \in \mathbb{R}^{3N}$ is the stacked vertex displacement vector, $\mathbf{a} \in \mathbb{R}^{3N}$ is the corresponding stacked acceleration vector, $\mathbf{v} \in \mathbb{R}^{3N}$ is the corresponding stacked velocity vector, $\mathbf{M} \in \mathbb{R}^{3N \times 3N}$ and $\mathbf{D} \in \mathbb{R}^{3N \times 3N}$ are the mass and damping matrices, and $\mathbf{f}^{\text{int}} \in \mathbb{R}^{3N}$ and $\mathbf{f}^{\text{ext}} \in \mathbb{R}^{3N}$ are the stacked internal and external force vectors. In the language of mechanics, these equations are known as *Euler-Lagrange* equations of motion.

Eq.2.2 obtained above can be time-stepped to obtain the displacements of the vertices under the influence of external force $\mathbf{f}$. However note that in general such an equation is non-linear primarily because the underlying material models are non-linear (Neohookean models using Green-Lagrange Strains). Non-linear models have the advantage of being able to model large deformations however time-stepping them is more complex.

Such non-linear equations can be solved by Newton-Raphson method. In this particular case, we are interested to advance the displacement and velocity vectors $(\mathbf{u}^n, \mathbf{v}^n)$ to their next time-step position $(\mathbf{u}^{n+1}, \mathbf{v}^{n+1})$.

$$\begin{aligned} \mathbf{Ma} + \mathbf{Dv} + \mathbf{f}^{\text{int}}(\mathbf{u}^{n+1}) &= \mathbf{f}^{\text{ext}} \\ \mathbf{v}^{n+1} = \mathbf{v}^n + \Delta t \mathbf{M}^{-1}(\mathbf{f}^{\text{ext}} - \mathbf{f}^{\text{int}}(\mathbf{u}^{n+1}) - \mathbf{D}\mathbf{v}^{n+1}) \end{aligned} \tag{2.3}$$

Newton-Raphson method iteratively defines a series of approximations $\mathbf{u}^{n+1}_{(0)}$,$\mathbf{u}^{n+1}_{(1)}$,...,$\mathbf{u}^{n+1}_{(k)}$, and $\mathbf{v}^{n+1}_{(0)}$,$\mathbf{v}^{n+1}_{(1)}$,...,$\mathbf{v}^{n+1}_{(k)}$ such that $\mathbf{u}^{n+1}_{(k)} \xrightarrow{k\to\infty} \mathbf{u}^{n+1}$ and $\mathbf{v}^{n+1}_{(k)} \xrightarrow{k\to\infty} \mathbf{v}^{n+1}$. Also note that in this case, we start the approximation as $\mathbf{u}^{n+1}_{(0)} = \mathbf{u}^n$ and $\mathbf{v}^{n+1}_{(0)} = \mathbf{v}^n$.

At each step of Newton-Raphson Iteration, we update the values of the variables $\mathbf{u}^{n+1}_{(k)}, \mathbf{v}^{n+1}_{(k)}$ till some satisfactory convergence criterion is reached. The difference of the values between successive approximations are referred to as *correction variables* defined as $\Delta u = \mathbf{u}^{n+1}_{(k+1)} - \mathbf{u}^{n+1}_{(k)}$ and $\Delta v = \mathbf{v}^{n+1}_{(k+1)} - \mathbf{v}^{n+1}_{(k)}$.

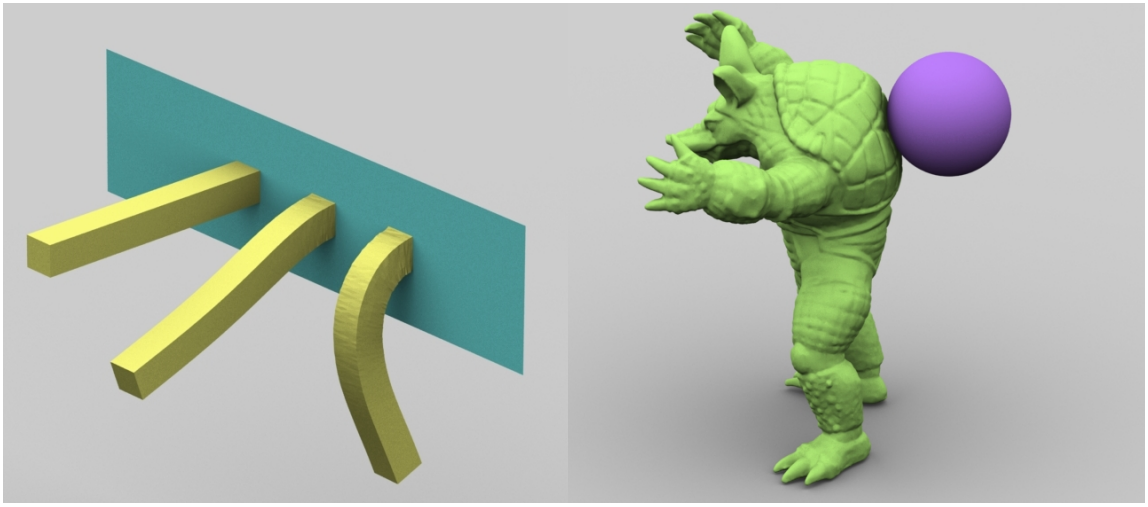
Finally we need to linearize Eq. 2.3 to obtain the values of correction variables. This is simply done by Taylor Expansion of $\mathbf{f}^{\text{int}}(\mathbf{u}^{n+1}_{(k+1)})$ as

$$\mathbf{f}^{\text{int}}(\mathbf{u}^{n+1}_{(k+1)}) = \mathbf{f}^{\text{int}}(\mathbf{u}^{n+1}_{(k)}) + \mathbf{K}(\mathbf{u}^{n+1}_{(k)})\Delta u.$$

Here $\mathbf{K}(\mathbf{u}^{n+1}_{(k)}) = \frac{\delta \mathbf{f}^{\text{int}}}{\delta u}|_{\mathbf{u}^{n+1}_{(k)}}$. This is the jacobian of the internal deformation force which is essential for linearization often known as *Tangent Stiffness Matrix*.

Linearization of Eq. 2.3 using the Taylor Expansion and subsequent algebraic manipulation leads to the final form as below :

$$\left(\mathbf{K}(\mathbf{u}^{n+1}_{(k)}) + \frac{1}{(\Delta t)^2}\mathbf{M} + \frac{1}{\Delta t}\mathbf{D}\right)\Delta u = \frac{1}{\Delta t}\mathbf{M}(\mathbf{v}^n - \mathbf{v}^{n+1}_{(k)}) + \left(\mathbf{f}^{\text{ext}} - \mathbf{f}^{\text{int}}(\mathbf{u}^{n+1}_{(k)}) + \mathbf{f}^{\text{damping}}(\mathbf{v}^{n+1}_{(k)})\right) \tag{2.4}$$



(a) Cantilever Beam (b) Armadillo

Figure 2.5: Some common models undergoing deformation either because of gravity as in the beam or collision forces as in the back of the armadillo.

Eq. 2.4 can be solved by any sparse solver such as Conjugate Gradient [She94] to obtain the values of the correction variables at each stage of the iteration. Typically it takes no more than 4-5 iterations for the system to converge satisfactorily.

Finite Element Method (FEM) provides a convenient way to calculate the values of the elements of the stiffness matrices as well as the forces. The choice of the elements in FEM (tetrahedral, cubic, hexahedral), the order of the elements and the constitutive models chosen result in expressions for the *Internal Forces* as well as the *Tangent Stiffness Matrices.* For a more detailed overview as to how we can obtain these expressions in code, refer to [ITF04], [TSIF05] and [XSZB15].

2.3.2 Subspace Simulation

As mentioned in the previous section, several natural phenomena (including elasticity which is the focus of this thesis) can be "semi-discretized" into a high-dimensional Ordinary

Differential Equation (ODE). The idea of *Subspace Simulation* is to reduce such a high-dimensional ODE and project it into some suitably chosen lower-dimensional subspace. Such an approximation results in a much simplified system that can be solved faster and then the result is reprojected back to the original high dimensional subspace. .These projection based techniques are also called Principal Orthogonal Directions (POD) Method by some authors and they appear frequently in engineering.

2.3.2.1 Model Reduction Techniques

Recall Eq. 2.2 from the previous section. Instead of solving the full ODE using Non-Linear Newton Raphson methods as described in the previous section to obtain the value of $\mathbf{u}$, constrain the displacement vector $\mathbf{u}$ into a subspace spanned by r deformation modes: $\{\phi_1, \phi_2, ..., \phi_r\}$. These modes can be assembled together into a $3N \times r$ matrix $\mathbf{U}$, known as the *subspace basis*. Since $\mathbf{u}$ is in the subspace defined by $\mathbf{U}$, we can define it as: $\mathbf{u} = \mathbf{U}\mathbf{q}$, in which $\mathbf{q} \in \mathbb{R}^r$ contains the reduced coordinates of $\mathbf{u}$ in the subspace. Assuming that $\mathbf{U}$ is mass-orthogonal: $\mathbf{U}^\mathsf{T}\mathbf{M}\mathbf{U} = \mathbf{I}$, we now obtain the governing equation in the subspace:

$$\ddot{\mathbf{q}} + \mathbf{U}^\mathsf{T}\mathbf{D}\mathbf{U}\dot{\mathbf{q}} + \mathbf{U}^\mathsf{T}\mathbf{f}^{\text{int}}(\mathbf{U}\mathbf{q}) = \mathbf{U}^\mathsf{T}\mathbf{f}^{\text{ext}}. \tag{2.5}$$

We introduce some new terminology now. In Eq. 2.5, define the quantity $\mathbf{U}^\mathsf{T}\mathbf{f}^{\text{int}}(\mathbf{U}\mathbf{q})$ as $\tilde{\mathbf{f}}^{\text{int}}$ referred to as *Reduced Internal Forces*. In the same vein as full space simulation, we also need to compute the derivative of $\tilde{\mathbf{f}}^{\text{int}}$, defined as below,

$$\widetilde{\mathbf{K}}(\mathbf{q}) = \frac{d\tilde{\mathbf{f}}^{\text{int}}}{d\mathbf{q}} = \mathbf{U}^\mathsf{T}\mathbf{K}(\mathbf{U}\mathbf{q})\mathbf{U} \in \mathbb{R}^{r\times r} \tag{2.6}$$

$\widetilde{\mathbf{K}}(\mathbf{q})$ is referred to as *Reduced Tangent Stiffness Matrix*.

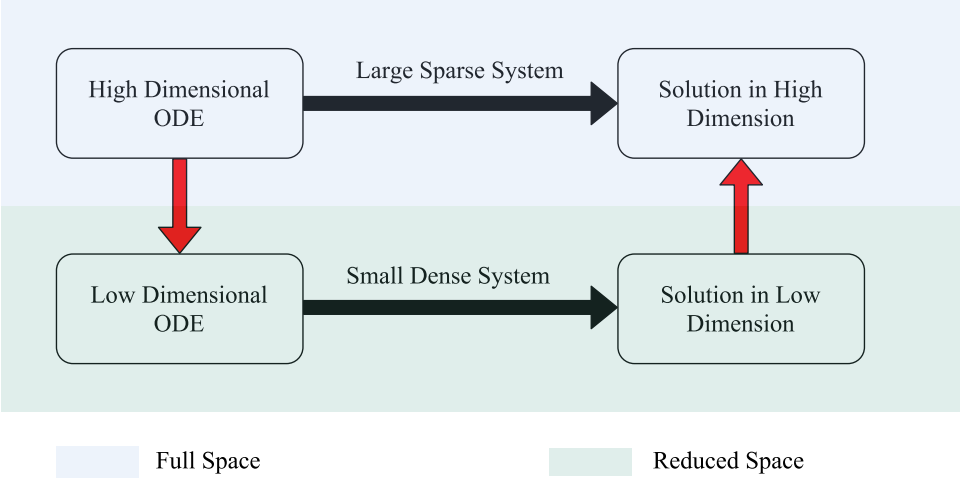


Figure 2.6: Overview of subspace simulation or model reduction

2.3.2.2 Computation of Basis Vectors

The success of subspace simulation is dependent on the choice of the basis vectors **U**. **U** is essentially a r-dimensional linear subspace of $\mathbb{R}^{3n}$. First we list some desirable properties of such a simulation basis :

1. The basis should well approximate the non-linear deformation spaces for the given geometry. Usually an ill-chosen basis will suffer from locking issues for large deformations and unwieldy artifacts during simulations.

2. The basis must be mass-orthogonal i.e $\mathbf{U}^{\mathsf{T}}\mathbf{M}\mathbf{U}$ where **M** is the mass matrix. This is easily enforced as any basis can be made to have this property byusing mass-weighted Gramm-Schmidt process.

There are two principal methods in Computer Graphics to generate the mass-orthogonal basis for deformable object simulation.

Example Driven Basis Generation : In this approach , we generate a basis by running the full-space simulation on the geometry under a variety of external forces to generate a variety of deformation values $\mathbf{u_1}, \mathbf{u_2}, ..., \mathbf{u_n}$.

Next we perform Singular Value Decomposition on the system $\mathbf{A} = [\mathbf{u_1}, \mathbf{u_2}, ..., \mathbf{u_n}]$,

$$\mathbf{A} = \mathbf{U}\blacksquare\mathbf{V}$$

The first *r* columns of **U** with the largest Singular Values provide us with a basis vector for simulation.

While this method is simple, in practice it suffers from two major drawbacks : Firstly it needs a large amount of example simulation data to accurately represent any random deformation. Secondly, if any deformation is modeled which has not been accurately represented in the example space, the simulation suffers from ungainly artifacts as it doesn't have any "memories" of such a deformation. The next method described can avoid such artifacts.

Modal Analysis Basis : As mentioned in the previous paragraph, the problem with example driven basis is the necessity for excessive amount of examples needed to simulate plausible deformation. Hence, a more methodical way to generate the basis is needed. An answer to this problem comes from the natural frequency of vibrations of a material subject to certain spatial constraints such as fixed vertices. More specifically, under boundary conditions such as fixed vertices, there are certain specific shapes that offer the least resistance to deformation. Such shapes are known as *modes*. It has been observed that a vector of such modes offer an efficient way to compute the basis for subspace deformations. [BJ05].

Next we come the question of how to compute such *modes*. Consider the stiffness matrix at rest position $\mathbf{K}(\mathbf{0} \in \mathbb{R}^{\mathbf{3n}})$ and the mass matrix $\mathbf{M}$ for a body at a given position. As it has been mentioned, the modes of a system are dependent on the boundary conditions such as fixed vertices in the system. If no such boundary conditions are set, we can simply use the matrices as is. If however, there are fixed vertices, we remove the rows and columns corresponding to those fixed degrees of freedom from both $\mathbf{K}(\mathbf{0} \in \mathbb{R}^{\mathbf{3n}})$ and $\mathbf{M}$ to obtain the matrices $\bar{\mathbf{K}}$ and $\bar{\mathbf{M}}$. Then we solve the *generalized Eigenvalue system*

$$\bar{\mathbf{K}}\phi_i = {\omega_i}^2 \bar{\mathbf{M}}\phi_i \tag{2.7}$$

to obtain the set of linear modes of the body ϕ_i and the corresponding vibration frequencies as ω_i. The set of r modes corresponding to the r smallest generalized eigenvalues can be formed into a $3N \times r$ linear modal matrix for subspace simulation: $\mathbf{U} = \{\phi_1, \phi_2, ..., \phi_r\}$.

2.3.2.3 Timestepping the equations

After applying the projection operations, we are left with the reduced equations of motion as demonstrated in Eq.2.5 with a dimensionality of r instead of n. Assuming that $r << n$, we can solve the implicit equations of motions much faster than the full-space equations. It is important to note that unlike Eqn. 2.2 which usually involve sparse systems, the reduced space equations are dense and hence we will need to use dense Cholesky decomposition to solve them.

However the bottleneck in subspace simulation is that if we have to evaluate the full step stiffness matrix $\widetilde{\mathbf{K}}(\mathbf{q})$ and full-space internal force $\tilde{\mathbf{f}}^{\text{int}}$ before we can apply the projection operations, all the computational savings that occur essentially boils down to the solve time of a large sparse system vs a small dense system. These savings, while substantial may

dwarf the actual simulation time and may be problematic for real-time simulations. In the next section, we discuss some methods to compute the reduced forces and reduced stiffness matrices directly without resorting to computation of full-space force and stiffness matrices.

2.3.2.4 Evaluation of reduced forces and stiffness matrices

Trivially, we can evaluate the reduced internal force and tangent stiffness matrix using projection from full-space internal force and tangent stiffness matrix. However, such a method has a runtime cost of $\Theta(nr)$ and $\Theta(nr^2)$ respectively (assuming as per previous conventions that n is the number of unreduced degrees of freedom and r is the number of modal basis). In this section, we explore some alternative methods that eliminates the dependence of the computation time on the number of unreduced degrees of freedom.

Precomputed cubic and quadratic polynomials Observe that the elastic strain energy as defined in the previous section can be shown to be a fourth-order polynomials in the components of the deformation $\mathbf{u}$ for linear and StVK materials owing to their underlying formulations. (This is not true for Isotropic materials). Hence the following discussions are only applicable to these classes of materials.

Since the unreduced internal forces are simply the derivatives of the internal strain energy, it is straightforward to show that the (unreduced) internal forces are cubic multi-variate polynomials in the deformation vector $\mathbf{u}$ of the vertex in question and all its immediate mesh neighbors. From this we can conclude that the reduced internal forces are also cubic multi-variate polynomials in the reduced deformation vector $\mathbf{q}$. For geometrically non-linear materials, we can precompute the coefficients of this cubic polynomials which expedites the process of computation of reduced forces and reduced stiffness matrices.

$$\begin{aligned}\tilde{\mathbf{f}}^{\text{int}}(\mathbf{q}) &= \mathbf{U}^{\mathsf{T}}\mathbf{f}^{\text{int}}(\mathbf{U}\mathbf{q}) \\ &= \sum_i \alpha^i q_i + \sum_{j\geq i} \beta^{ij} q_i q_j + \sum_{k\geq j\geq i} \gamma^{ijk} q_i q_j q_k\end{aligned} \tag{2.8}$$

Eq. 2.8 shows the expression of the reduced internal force $\tilde{\mathbf{f}}^{\text{int}}(\mathbf{q})$ as a expansion with third order polynomial in reduced deformation vector with the corresponding constant polynomial coefficients as $\alpha^i, \beta^{ij}, \gamma^{ijk} \in \mathbb{R}^r$. Our goal is to evaluate these coefficients and then at runtime simply compute the internal forces using this formulation. Since the terms in these coefficients are all of dimensionality r, the aforementioned dependence on the number of unreduced degrees of freedom r is eliminated.

Assuming that the coefficients for the evaluation of unreduced StVk forces are known and available, it is trivial to evaluate the coefficients for a given basis vector $\mathbf{U}$. The details of evaluation are available in [BJ05].

A similar argument follows for the evaluation of reduced tangent stiffness matrix. It is to be noted that since the tangent stiffness matrix is the derivative of the reduced internal forces, it is a second order multi-variate polynomial in the elements of the deformation vector. Each column l of $\widetilde{\mathbf{K}}(\mathbf{q})$ is given as

$$\widetilde{\mathbf{K}}(\mathbf{q})_l = \alpha^l + (\beta^{li} + \beta^{il})q_i + (\gamma^{lij} + \gamma^{ilj} + \gamma^{ijl})q_i q_j$$

where $\alpha^i, \beta^{ij}, \gamma^{ijk} \in \mathbb{R}^r$ are the aforementioned cubic polynomial coefficients.

Time and space complexity : There is one cubic polynomial per reduced force dimension (which makes a total of r cubic polynomials). There is one quadratic polynomial per entry of the reduced tangent stiffness matrix (exploiting symmetry there are $\frac{r(r+1)}{2}$ distinct entries in the tangent stiffness matrix). Precomputation of the polynomial coefficients takes $O(n.r^4)$ time. However this is a one-time cost. At run time, we only need to access each polynomial

coefficients exactly once hence the total computation time is $O(r^4)$. Also we need to store all these distinct entries which brings the storage requirement to $O(r^4)$ also.

Cubatures Computing the coefficients of the reduced polynomials as mentioned in the previous section only works for a certain class of materials (geometrically non-linear but materially linear such as Linear Elasticity and StVK). In order to circumvent this limitation, An et al. [AKJ08] proposed the method of *cubatures* that works for a general class of material. The basic idea for cubatures is noticing the fact that the elastic strain energy $E(\mathbf{q})$ and the reduced internal forces are basically integrations of the energy density $\widetilde{\Psi}(\mathbf{X},\mathbf{q})$ (Energy density is defined as the strain energy per unit undeformed volume on an infinitesimal domain around a given material point $\mathbf{X}$) and its gradient over the entire domain :

$$\widetilde{E}(\mathbf{q}) = \int_{\Omega} \widetilde{\Psi}(\mathbf{X},\mathbf{q}) dV \tag{2.9}$$

$$\widetilde{\mathbf{f}}^{\text{int}}(\mathbf{q}) = \int_{\Omega} \frac{\partial \widetilde{\Psi}(\mathbf{X},\mathbf{q})}{\partial \mathbf{q}} dV \tag{2.10}$$

Just like the method of gaussian quadrature used for evaluating approximate numerical solutions for definite integrals, cubatures also approximately evaluate the solutions of the integrals in Eq. 2.9, 2.10.

$$\widetilde{\mathbf{f}}^{\text{int}}(\mathbf{q}) = \int_{\Omega} \frac{\partial \widetilde{\Psi}(\mathbf{X},\mathbf{q})}{\partial \mathbf{q}} dV \approx \sum_{i=1}^{|T|} w_i \frac{\partial \Psi(\mathbf{X_i},\mathbf{q})}{\partial \mathbf{q}} \tag{2.11}$$

$$\widetilde{\mathbf{K}}(\mathbf{q}) \approx \sum_{i=1}^{|T|} w_i \frac{\partial^2 \Psi(\mathbf{X_i},\mathbf{q})}{\partial \mathbf{q}^2} \tag{2.12}$$

In Eq. 2.11, 2.12, the set T is basically the set of cubature points which are basically an ordered pair $(\mathbf{X_i}, w_i), \mathbf{X_i} \in \Omega, w_i \in \mathbb{R}$. The cardinality of the set T is a user adjustable

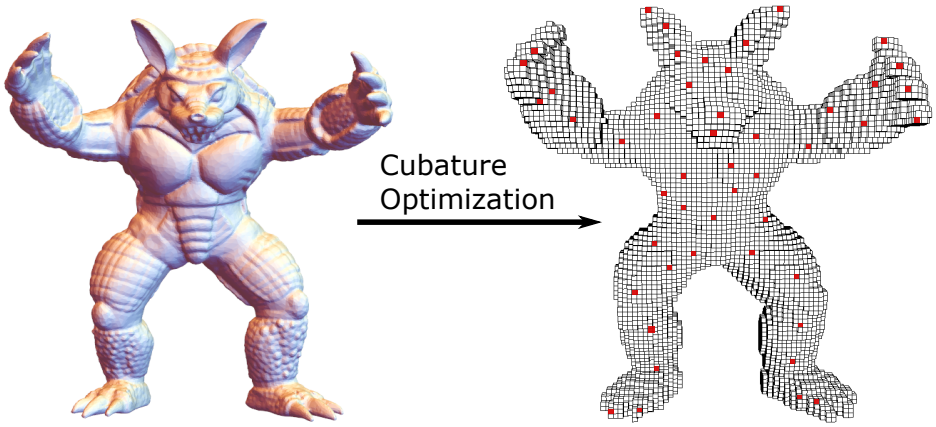


Figure 2.7: The model on the left has been voxelized. The cubic elements highlighted in red are the cubature points. This is a representative image only. The voxelization has been performed in cubes to show the cubature points more easily.

parameter and it influences how well the cubature approximates the actual values of the reduced force and stiffness matrix. (More the number of cubature points, more accurate is the evaluation however more is the computation time). A good value of $|T|$ is r (the number of basis). Fig. 2.7 shows the various cubature points for a sample geometry.

Next we come to the important question of how to evaluate these cubature point set T. The cubature points $\mathbf{X}_i$ and their associated weights are precomputed at the start of the simulation using a training process. The input to this algorithm is a set of k sample reduced coordinates $\{\mathbf{q}^{(1)}, \mathbf{q}^{(2)}, ..., \mathbf{q}^{(k)}\}$. These can be generated randomly or from actual sample deformations from unreduced coordinates. Using these sample reduced coordinates, the method computes positions and weights that best approximate the reduced force $\widetilde{\mathbf{f}}^{\text{int}}(\mathbf{q})$ for these training data points. The weights are always chosen to be non-negative using the method of Non-negative Least Squares(NNLS). The position vectors are essentially elements from the domain Ω chosen using a greedy algorithm that minimizes the NNLS

residual. The training process can be time consuming for very large models and hence [AKJ08] suggested that the algorithm be applied on a subset of the actual samples and then occasionally perform NNLS on the full training sample.

Time and space complexity : For a n_c-point cubature (i.e $|T| = n_c$) using r reduced basis dimension , the training process with k training samples has a time complexity of $O(rkn_c{}^2)$. However this is a one-time cost. At runtime, the cost of evaluation of the reduced internal force and stiffness matrix is $O(rn_c)$ and $O(r^2n_c)$ respectively. The only information that is needed to be stored is the cubature points and their weights and so the space complexity is only $O(n_c)$. Should we not need to explicitly form the stiffness matrix and only require the stiffness matrix-vector product (as is done in many implicit integrators), we can compute the product in $O(r^2)$ (assuming $n_c = r$).

2.4 One Stitch Case

Our research is focused on the reconstruction of the subspace basis, after simple editing operations. To begin with, let us consider stitching two co-located vertices together by a zero-length spring, as Figure 2.8 shows. In Subsection 2.4.1, we will present our incremental approach for computing the linear deformation modes of the stitched body. In Subsection 2.4.2, we will show how to recompute modal derivatives for nonlinear deformation. Finally in Subsection 2.4.3, we will discuss the use of the new basis in simulation.

2.4.1 Incremental Linear Modal Analysis

Let $\mathbf{K} = \partial \mathbf{f}^{\text{int}} / \partial \mathbf{u}$ be the $3N \times 3N$ stiffness matrix of the deformable body evaluated at $\mathbf{u} = \mathbf{0}$, where $\mathbf{f}^{\text{int}}$ is the internal force and $\mathbf{u}$ is the vertex displacement vector. Linear modal

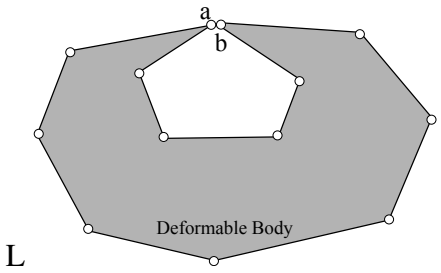


Figure 2.8: One stitch case. After a zero-length spring connects two co-located vertices a and b, our goal is to quickly calculate the subspace of the stitched body from the old one.

analysis solves a sparse, generalized eigenvalue decomposition problem:

$$\mathbf{K}\phi_i = \omega_i^2 \mathbf{M}\phi_i, \tag{2.13}$$

where $\mathbf{M}$ is the invertible mass matrix and ϕ_i is the i-th vibration mode with frequency ω_i. Since high-frequency deformations are hardly noticeable in simulation, we ignore the modes with high frequencies and the six rigid modes with zero frequencies. The remaining modes can then be organized into a $3N \times r$ linear modal matrix for subspace simulation: $\mathbf{U} = \{\phi_1, \phi_2, ..., \phi_r\}$.

Now let us consider stitching two co-located vertices a and b together, as Figure 2.8 shows. We model this stitch as a zero-length spring with stiffness k, which applies a spring force on vertex a as: $\mathbf{f}_{ab} = k(\mathbf{u}_a - \mathbf{u}_b)$. The stiffness matrix resulted from this spring is a sparse block matrix:

$$\mathbf{K}_{ab} = \begin{bmatrix} \frac{\partial \mathbf{f}_{ab}}{\partial \mathbf{u}_a} & \frac{\partial \mathbf{f}_{ab}}{\partial \mathbf{u}_b} \\ \frac{\partial \mathbf{f}_{ba}}{\partial \mathbf{u}_a} & \frac{\partial \mathbf{f}_{ba}}{\partial \mathbf{u}_b} \end{bmatrix} = \begin{bmatrix} k\mathbf{I} & -k\mathbf{I} \\ -k\mathbf{I} & k\mathbf{I} \end{bmatrix}, \tag{2.14}$$

where the blocks are corresponding to vertex a and b respectively. It is straightforward to see that $\mathbf{K}_{ab}$ is a rank-3 matrix and it can be decomposed into: $\mathbf{K}_{ab} = \mathbf{A}\mathbf{A}^\mathsf{T}$, in which $\mathbf{A}$ is a

$3N \times 3$ matrix:

$$\mathbf{A} = \begin{bmatrix} & \sqrt{k}\mathbf{I} & & -\sqrt{k}\mathbf{I} & \end{bmatrix}^{\mathsf{T}}. \tag{2.15}$$

Here the two blocks are the a-th and the b-th blocks.

Performing linear modal analysis on the newly stitched body is equivalent to solving a new generalized eigenvalue problem: $\hat{\mathbf{K}}\hat{\Phi}_i = \hat{\omega}_i^2 \mathbf{M}\hat{\Phi}_i$, in which $\hat{\mathbf{K}} = \mathbf{K} + \mathbf{K}_{ab}$ is the stiffness matrix of the new body. This generalized eigenvalue problem is computationally expensive to solve. Fortunately, we can approximate $\mathbf{K}$ by its low-frequency component and solve a low-rank eigenvalue problem afterwards. Let $\mathbf{U}$ and $\mathbf{U}_{\mathrm{H}}$ be low-frequency and high-frequency modes of $\mathbf{K}$, and $\mathbf{\Lambda}$ and $\mathbf{\Lambda}_{\mathrm{H}}$ be their diagonal eigenvalue matrices. By definition, the joint modal matrix must be mass-diagonal:

$$\begin{bmatrix} \mathbf{U} & \mathbf{U}_{\mathrm{H}} \end{bmatrix}^{\mathsf{T}} \mathbf{M} \begin{bmatrix} \mathbf{U} & \mathbf{U}_{\mathrm{H}} \end{bmatrix} = \mathbf{I}. \tag{2.16}$$

So we can split $\mathbf{K}$ into two parts:

$$\begin{aligned} \mathbf{K} &= \mathbf{M} \begin{bmatrix} \mathbf{U} & \mathbf{U}_{\mathrm{H}} \end{bmatrix} \begin{bmatrix} \mathbf{\Lambda} & \\ & \mathbf{\Lambda}_{\mathrm{H}} \end{bmatrix} \begin{bmatrix} \mathbf{U} & \mathbf{U}_{\mathrm{H}} \end{bmatrix}^{-1} \\ &= \mathbf{M} \begin{bmatrix} \mathbf{U} & \mathbf{U}_{\mathrm{H}} \end{bmatrix} \begin{bmatrix} \mathbf{\Lambda} & \\ & \mathbf{\Lambda}_{\mathrm{H}} \end{bmatrix} \begin{bmatrix} \mathbf{U} & \mathbf{U}_{\mathrm{H}} \end{bmatrix}^{\mathsf{T}} \mathbf{M} \\ &= \mathbf{M}\mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\mathsf{T}}\mathbf{M} + \mathbf{M}\mathbf{U}_{\mathrm{H}}\mathbf{\Lambda}_{\mathrm{H}}\mathbf{U}_{\mathrm{H}}^{\mathsf{T}}\mathbf{M}, \end{aligned} \tag{2.17}$$

in which $\mathbf{M}\mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\mathsf{T}}\mathbf{M}$ contains the low-frequency component and $\mathbf{M}\mathbf{U}_{\mathrm{H}}\mathbf{\Lambda}_{\mathrm{H}}\mathbf{U}_{\mathrm{H}}^{\mathsf{T}}\mathbf{M}$ contains the high-frequency component. We then solve the generalized eigenvalue problem on $\mathbf{M}\mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\mathsf{T}}\mathbf{M} + \mathbf{K}_{ab}$.

Our idea is based on the assumption that the newly added spring is stiff and it cannot cause high-frequency deformations to become low-frequency. Therefore, the low-frequency subspace of the exact matrix can be well covered by the low-frequency subspace of the approximation matrix. To evaluate the plausibility of this idea, we apply generalized eigenvalue decomposition on $\mathbf{K} + \mathbf{K}_{ab}$, and then measure the distance from each exact

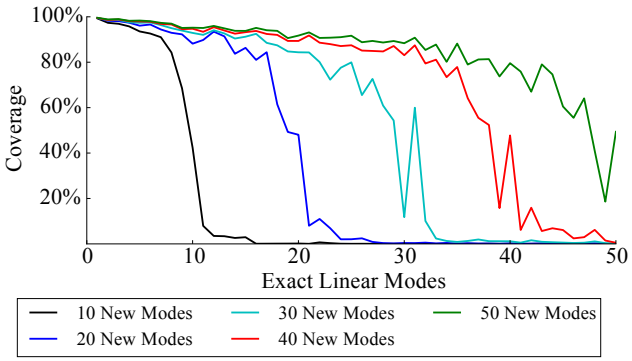


Figure 2.9: Subspace coverage. This plot illustrates how well each exact linear mode can be represented by the recalculated linear basis shown as each curve. The recalculated basis can represent the exact modes more closely, if it contains more modes.

low-frequency mode to its projection in the subspace spanned by our recalculated modes. Using this distance, we can calculate the percentage of the exact mode being covered by the recalculated basis. Figure 2.9 shows the first few exact modes can be well represented by the first few recalculated modes. After that, more exact modes can be represented by more recalculated modes, as expected.

2.4.1.1 Numerical Implementation

Since $\mathbf{MU\Lambda U}^{\mathsf{T}}\mathbf{M}$ is a rank-r matrix and $\mathbf{K}_{ab}$ is a rank-3 matrix, the rank of their sum $\bar{\mathbf{K}}$ cannot be higher than $r+3$. Inspired by the fast low-rank modification framework [Bra06], we propose to formulate the approximation of $\bar{\mathbf{K}}$ as:

$$\mathbf{L}\begin{bmatrix}\mathbf{L}^{\mathsf{T}}\mathbf{U} & \mathbf{L}^{-1}\mathbf{A}\end{bmatrix}\begin{bmatrix}\mathbf{\Lambda} & \\ & \mathbf{I}\end{bmatrix}\begin{bmatrix}\mathbf{L}^{\mathsf{T}}\mathbf{U} & \mathbf{L}^{-1}\mathbf{A}\end{bmatrix}^{\mathsf{T}}\mathbf{L}^{\mathsf{T}}, \tag{2.18}$$

where $\mathbf{M}=\mathbf{LL}^{\mathsf{T}}$ is the Cholesky decomposition of $\mathbf{M}$. Here $\mathbf{L}^{\mathsf{T}}\mathbf{U}$ is an orthogonal basis, since $\mathbf{U}$ is mass-diagonal and $\mathbf{U}^{\mathsf{T}}\mathbf{LL}^{\mathsf{T}}\mathbf{U}=\mathbf{I}$. The component of $\mathbf{L}^{-1}\mathbf{A}$ that is orthogonal to $\mathbf{L}^{\mathsf{T}}\mathbf{U}$ can be defined as: $(\mathbf{I}-\mathbf{L}^{\mathsf{T}}\mathbf{UU}^{\mathsf{T}}\mathbf{L})\mathbf{L}^{-1}\mathbf{A}$. Let $\mathbf{P}$ be its orthogonal basis calculated

using the Gram-Schmidt algorithm. We must have $\mathbf{L}^\mathsf{T}\mathbf{U}$ and $\mathbf{P}$ orthogonal to each other and they span the space of $\mathbf{L}^{-1}\mathbf{A}$. Therefore, we have:

$$\begin{bmatrix} \mathbf{L}^\mathsf{T}\mathbf{U} & \mathbf{L}^{-1}\mathbf{A} \end{bmatrix} = \begin{bmatrix} \mathbf{L}^\mathsf{T}\mathbf{U} & \mathbf{P} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{U}^\mathsf{T}\mathbf{A} \\ \mathbf{0} & \mathbf{R} \end{bmatrix}, \tag{2.19}$$

where $\mathbf{R} = \mathbf{P}^\mathsf{T}(\mathbf{I} - \mathbf{L}^\mathsf{T}\mathbf{U}\mathbf{U}^\mathsf{T}\mathbf{L})\mathbf{L}^{-1}\mathbf{A}$. We can then reformulate the approximation into:

$$\bar{\mathbf{K}} = \mathbf{L} \begin{bmatrix} \mathbf{L}^\mathsf{T}\mathbf{U} & \mathbf{P} \end{bmatrix} \mathbf{C} \begin{bmatrix} \mathbf{L}^\mathsf{T}\mathbf{U} & \mathbf{P} \end{bmatrix}^\mathsf{T} \mathbf{L}^\mathsf{T}, \tag{2.20}$$

in which $\mathbf{C}$ is a $(r+3)\times(r+3)$ matrix:

$$\mathbf{C} = \begin{bmatrix} \mathbf{\Lambda} & \\ & \end{bmatrix} + \begin{bmatrix} \mathbf{U}^\mathsf{T}\mathbf{A} \\ \mathbf{R} \end{bmatrix} \begin{bmatrix} \mathbf{U}^\mathsf{T}\mathbf{A} \\ \mathbf{R} \end{bmatrix}^\mathsf{T}. \tag{2.21}$$

Let $\mathbf{C} = \mathbf{u}\bar{\mathbf{\Lambda}}\mathbf{u}^\mathsf{T}$ be the eigenvalue decomposition of $\mathbf{C}$, we have:

$$\bar{\mathbf{K}} = \mathbf{L} \begin{bmatrix} \mathbf{L}^\mathsf{T}\mathbf{U} & \mathbf{P} \end{bmatrix} \mathbf{u}\bar{\mathbf{\Lambda}}\mathbf{u}^\mathsf{T} \begin{bmatrix} \mathbf{L}^\mathsf{T}\mathbf{U} & \mathbf{P} \end{bmatrix}^\mathsf{T} \mathbf{L}^\mathsf{T}. \tag{2.22}$$

The novel linear modal matrix can then be calculated as: $\bar{\mathbf{U}} = \mathbf{L}^{-\mathsf{T}} \begin{bmatrix} \mathbf{L}^\mathsf{T}\mathbf{U} & \mathbf{P} \end{bmatrix} \mathbf{u}$ and $\bar{\mathbf{\Lambda}}$ contains its generalized eigenvalues. This is because $(\mathbf{L}^{-1}\bar{\mathbf{K}}\mathbf{L}^{-\mathsf{T}})(\mathbf{L}^\mathsf{T}\bar{\mathbf{U}}) = (\mathbf{L}^\mathsf{T}\bar{\mathbf{U}})\bar{\mathbf{\Lambda}}$.

If $\mathbf{M}$ is a diagonal matrix, both $\mathbf{L}$ and $\mathbf{L}^{-1}$ are easy to calculate. However, if $\mathbf{M}$ is not a diagonal matrix, $\mathbf{L}^{-1}$ can become dense and we cannot calculate it directly. Fortunately, since $\mathbf{L}$ is a lower triangular matrix and $\mathbf{A}$ has three columns only, we can solve $\mathbf{L}^{-1}\mathbf{A}$ by three forward substitutions. Similarly, since the rank of $(\mathbf{I} - \mathbf{L}^\mathsf{T}\mathbf{U}\mathbf{U}^\mathsf{T}\mathbf{L})\mathbf{L}^{-1}\mathbf{A}$ cannot be greater than 3, its orthogonal basis $\mathbf{P}$ cannot have more than three columns. We can solve $\mathbf{L}^{-\mathsf{T}}\mathbf{P}$ by at most three backward substitutions. To reduce the computational cost, we precompute $\mathbf{L}$ as a sparse matrix, and we maintain both $\mathbf{U}$ and $\mathbf{L}^\mathsf{T}\mathbf{U}$ during runtime.

If the body is unconstrained, we found that it is not a good practice to ignore the six rigid modes, even though their generalized eigenvalues are zeros and they are not useful in subspace simulation. Instead, we keep them through the whole incremental linear modal

analysis process, and remove them only when we use the basis for simulation or additional processes.

2.4.2 Basis Extension for Nonlinear Deformation

Large and nonlinear deformation cannot be handled by linear vibration modes. One solution to this problem is modal warping [CK05, HTZ+11], which tries to remove the artifacts caused by linear modes directly. Unfortunately, it cannot correctly handle nonlinear material behaviors and it can cause drifting artifacts in free falling cases. Alternatively, von Tycowicz and colleagues [vTSSH13] proposed to expand the basis, by decoupling each linear mode into nine new modes. Although their method can more accurately represent nonlinear deformation, it significantly increases the basis size to $9r+12$, which cannot be easily reduced for more efficient subspace simulation.

2.4.2.1 Modal Derivatives

To address the nonlinear deformation issue, we propose to use the modal derivative approach proposed by Barbič and James [BJ05]. Given two new linear modes $\bar{\phi}_i$ and $\bar{\phi}_j$ in the basis matrix $\bar{\mathbf{U}}$, we calculate their derivative $\bar{\phi}_{ij}$ by solving the linear system:

$$(\mathbf{K}+\mathbf{K}_{ab})\bar{\phi}_{ij} = -(\bar{\mathbf{H}} : \bar{\phi}_i)\bar{\phi}_j, \tag{2.23}$$

in which $\bar{\mathbf{H}} = \partial\bar{\mathbf{K}}/\partial\mathbf{u}$ at $\mathbf{u}=0$, an order-3 tensor representing the first-order derivative of $\bar{\mathbf{K}}$, also known as the *Hessian stiffness tensor*. According to Equation 2.14, $\mathbf{K}_{ab}$ is constant and its Hessian stiffness tensor is trivial. So we have $\bar{\mathbf{H}} = \mathbf{H}$, which can be precomputed using the base mesh.

Let r be the total number of non-rigid linear modes. Their combinations can provide us $r(r+1)/2$ modal derivatives, each of which requires to solve the linear system in

Equation 2.23 once. Although this seems to be computationally expensive, Barbič and James [BJ05] pointed out that $\mathbf{K} + \mathbf{K}_{ab}$ stays the same for all of the systems, so it can be pre-factorized for fast direct solve. According to our experiment, the real bottleneck is the evaluation of the tensor-vector-vector product on the right-hand side of Equation 2.23.

So instead of using all of the r modes to calculate modal derivatives, we choose to use the first six non-rigid linear modes only. Since the resulting 21 modal derivatives are typically insufficient to cover enough nonlinear deformations, we reuse 40 percent of the right-hand sides evaluated from the base mesh to generate additional modal derivatives, assuming that the nonlinear relationships among the modes after stitching is similar to those before stitching. The plant example in Figure 3.1 shows that this hybrid strategy still allows the resulting basis to capture large and nonlinear deformation. Meanwhile, it reduces the computational time spent on modal derivatives from 6.82s to 0.89s, thanks to fewer Tensor-vector-vector products. Note that our strategy can also introduce some high-frequency vibrations back into the basis, which were lost due to low-rank approximation discussed in Subsection 2.4.1.

2.4.2.2 Basis reduction

So far we have collected many linear modes and their modal derivatives. For faster subspace simulation, Barbič and James [BJ05] recommended the use of principal component analysis (PCA) to reduce the subspace basis again. We adopt this idea and we use the randomized PCA method proposed by Halko and colleagues [HMT11]. Previous research showed that randomized PCA can run orders-of-magnitude faster than standard PCA, yet it has similar accuracy and robustness. Before performing randomized PCA, we scale each linear mode $\bar{\phi}_i$ by $\bar{\omega}_i/\bar{\omega}_1$ and each modal derivative $\bar{\phi}_{ij}$ by $\bar{\omega}_i\bar{\omega}_j/\bar{\omega}_1^2$. Here $\bar{\omega}_i$ is the frequency of $\bar{\phi}_i$ given by $\bar{\mathbf{\Lambda}}$ in Equation 2.22, and $\bar{\omega}_1$ is the frequency of the first mode, i.e.,

the lowest frequency. These scalings are used to prevent low-frequency modes from being dominated by high-frequency modes, as suggested in [BJ05].

2.4.3 Cubature Approximation

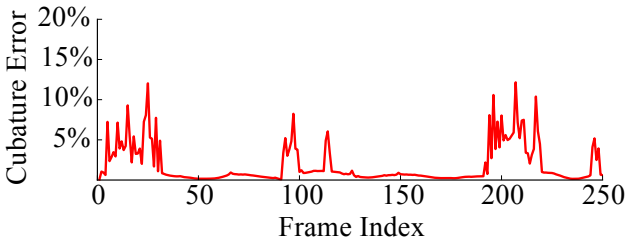
Given the new basis $\bar{\mathbf{U}}$, we are now ready to formulate the dynamical system for subspace simulation described in Equation 2.5. Suppose that the evaluation of the reduced internal force was accelerated by using cubature samples. An important question is: *how can we still quickly evaluate the reduced internal force, when the subspace basis gets changed?*

Let $\mathbf{f}^{\text{int}}$ be the internal force of the original mesh and $\mathbf{f}_{ab}$ be the new stitching force. The total reduced force is: $\bar{\mathbf{U}}^{\mathsf{T}}\mathbf{f}^{\text{int}}(\bar{\mathbf{U}}\bar{\mathbf{q}}) + \bar{\mathbf{U}}^{\mathsf{T}}\mathbf{f}_{ab}(\bar{\mathbf{U}}\bar{\mathbf{q}})$, where $\bar{\mathbf{q}}$ contains the reduced coordinates of the current shape in the new subspace. We assume that the cubature samples trained for the original mesh can still be used to evaluate the reduced internal force in the new subspace. So we have:

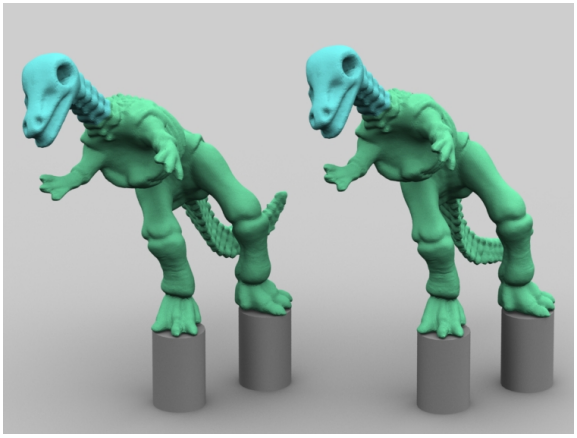
$$\bar{\mathbf{U}}^{\mathsf{T}}\mathbf{f}^{\text{int}}(\bar{\mathbf{U}}\bar{\mathbf{q}}) \approx \sum_k w_k \bar{\mathbf{U}}^{\mathsf{T}}\mathbf{f}_k^{\text{int}}(\bar{\mathbf{U}}\bar{\mathbf{q}}), \tag{2.24}$$

where k and w_k are the cubature sample and weight trained for the original mesh. To evaluate $\bar{\mathbf{U}}^{\mathsf{T}}\mathbf{f}_{ab}$, we simply calculate $\mathbf{f}_{ab}$ as a sparse vector and perform the matrix-vector product. In total, the cost of reduced force evaluation is $O(r^2)$, where r is the number of the deformation modes. We calculate the reduced stiffness matrix in the subspace using a similar approach.

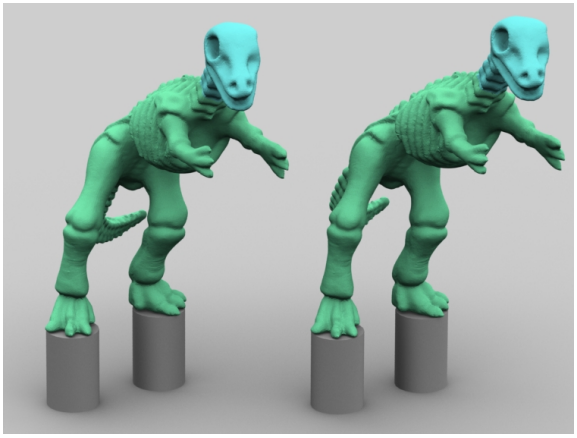
Figure 2.10a illustrates the approximation error produced by our cubature scheme in each frame, using the dinosaur example. These errors are typically small, except when user interaction happens. To further understand how the errors can be accumulated to affect the simulation, we run the same simulation twice: once without cubature approximation and once with cubature approximation. Although the two simulation results are not identical,



(a) Per-frame cubature approximation error



(b) Frame 112



(c) Frame 210

Figure 2.10: Quantitative and qualitative results without and with cubature optimization. In (b) and (c), the results without cubature approximation are shown on the left side, and the results with cubature approximation are shown on the right side.

they are visually similar as shown in Figure 2.10b and 2.10c. So we think our cubature scheme is still effective.

2.5 Complex Editing Cases

In this section, we will discuss how our techniques can be adopted to handle more complex editing operations.

2.5.1 Multiple Stitches and Domains

It is straightforward to extend the method described in Section 2.4 for multiple stitches, i.e., zero-length springs. Every spring adds three new rows and columns to the matrix $\mathbf{C}$ in Equation 2.21, so its size becomes $(r+3s)\times(r+3s)$. Here s is the number of stitches. Our method can efficiently handle a small number of stitches, as Figure 2.11 shows. When s increases, our method needs more computational cost and it becomes less attractive. In that case, we may need to recalculate the subspace basis from scratch again, using the expedited approach [YLX+15] for example.

If we consider the union of separate meshes as a single original mesh, we can use the method to reconstruct the subspace of a mesh with multiple domains as well. In that case, we can simply build the subspace of each domain separately. Let ϕ_i be a linear mode of domain k, we expand it into the linear mode of the joint original mesh as: $[\mathbf{0}^\mathsf{T},...,\mathbf{0}^\mathsf{T},\phi_i^\mathsf{T},\mathbf{0}^\mathsf{T},...,\mathbf{0}^\mathsf{T}]^\mathsf{T}$, where zeros represent no deformations in other domains. By expanding all of these linear modes, we obtain a set of linear modes for the joint original mesh. We then add stitches to connect the separate parts and apply our method to build the subspace of the newly stitched mesh as a whole.

Incorporating domain decomposition into our subspace reconstruction process has an obvious advantage: we can modify each domain separately, without affect the subspaces of other domains. For instance, we can increase or decrease the material stiffness of one domain. If the stiffness of the whole domain is scaled by a constant factor K, its linear

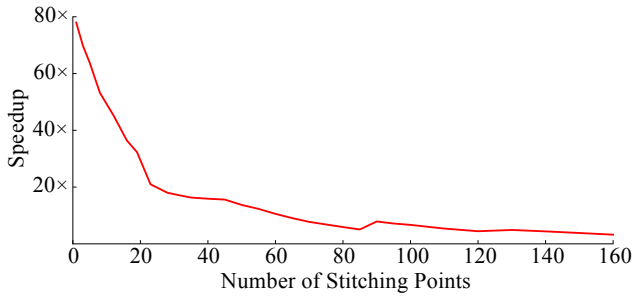


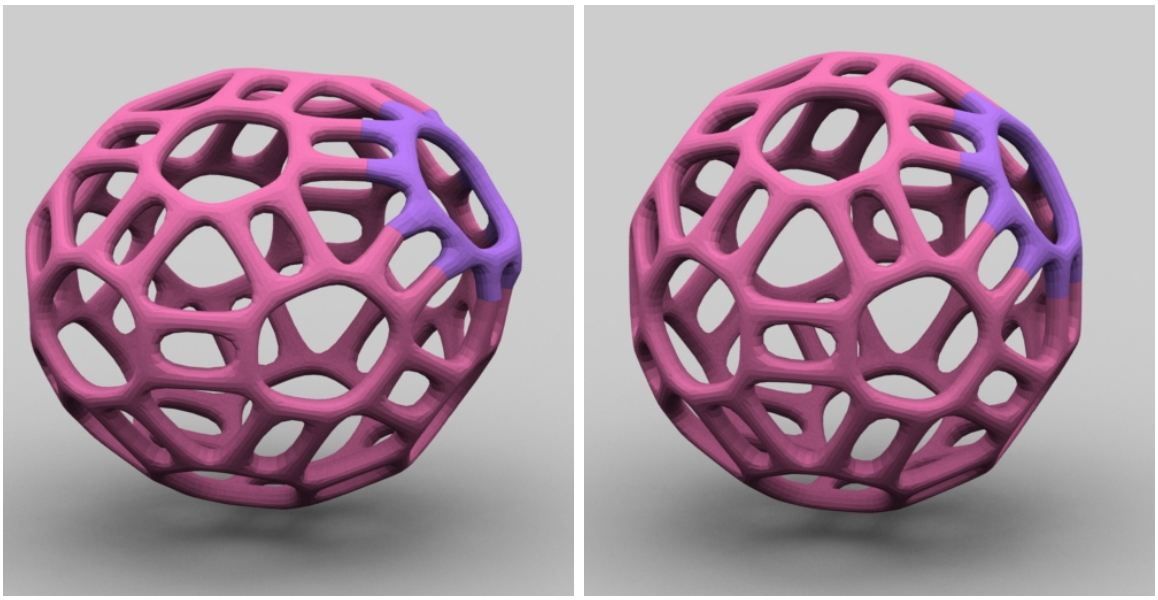
Figure 2.11: The relationship between the number of stitches and the speedup gained by using our methods. In general, using more stitches makes our methods less effective.

modes remain unchanged and the frequencies are scaled by $\sqrt{K}$, according to Equation 2.13. Figure 2.12 shows the simulation results of an Oball example. When the pink domain of the ball becomes stiffer, our method can efficiently recalculate the subspace within 4.5s for simulation afterwards.

2.5.2 Unconstrained Bodies and Domains

When a deformable body is unconstrained, it has six rigid linear vibration modes with zero frequencies. Similar to [BZ11], we do not incorporate rigid modes into subspace simulation, to prevent other modes from becoming time-dependent. Instead, we separate rigid motion from subspace simulation and animate it by rigid body dynamics. Since subspace deformation is now in a non-inertial local frame, we must add inertial forces to account for Coriolis, inertial, Euler and centrifugal effects. We use fast sandwich transform [KJ11] to calculate these forces in the subspace.

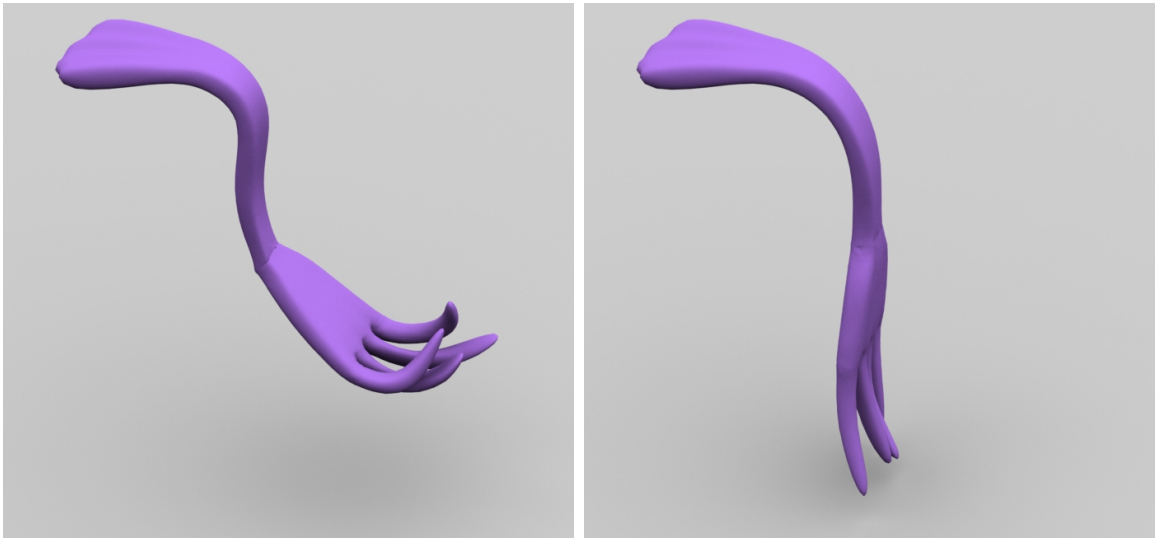
When we stitch an unconstrained domain to constrained ones, such as the fork example in Figure 2.13a, we must consider its six rigid modes to avoid rotational artifacts. Specifically, we add the rigid modes into the subspace basis and then calculate the subspace for the whole



(a) Compliant base (b) Stiff base

Figure 2.12: The Oball example. Our system allows users to modify the stiffness of different domains, with a low subspace recalculation cost. In (a), the pink domain and the purple domain share the same stiffness, while in (b), the pink domain becomes 100 times stiffer.

stitched body, as discussed previously in Subsection 2.5.1. Doing this allows the fork head to be properly rotated, as shown in Figure 2.13b. Unfortunately, there is still one remaining problem: the linear modes of the unconstrained domain, i.e. the fork head, does not get rotated to reflect the deformation of the constrained domain. One possible solution is to couple the linear modes of different domains together through interface alignment, rather than padding them with zeros. However, we may need more computational cost to reduce the increased subspace size accordingly.



(a) Without six rigid modes (b) With six rigid modes

Figure 2.13: The fork example. Without considering the six rigid modes of the fork head, the simulator cannot properly handle large rotational motions as shown in (a).

2.6 Results and Discussions

(Please watch the supplementary video for animation examples.) We implement our methods and tested our examples on an Intel Core i7-3770S 3.10GHz processor. We use the Intel MKL library and the OpenMP API for parallelization. In addition, we apply Eigen and Armadillo linear algebra libraries for linear solves. We perform generalized eigenvalue decomposition on large sparse matrices by ARPACK.

Most of our examples have a Youngs's Modulus of 5.0×10^5 and the connection spring stiffness of 1.0×10^5. The value is same for all the springs in the same example. Our experiments show that as long as the connecting spring stiffness is within a reasonable order of magnitude of the connection springs, no additional parameter tuning is needed. However if springs of extremely large magnitude (say 10^{12} or above) is used for relatively

less stiff materials, locking issues may occur. All of our examples set the time step to 0.01s. We render one frame every three time steps. We typically use 60 to 225 cubature samples for evaluating reduced forces and matrices. We cap the number of total modes in the subpace basis to 90, to ensure the simulation performance. The frame rates of our examples vary from 8 to 30 FPS, depending on the number of modes and the number of cubature samples. Table 2.1 summarizes the cost of each subspace recalculation step. It also provides the speedup compared with the standard approach that recalculates the subspace from scratch. Note that the speedup we gain in the PCA reduction step is mostly due to the use of randomized PCA, not our new methods.

We evaluate the "goodness" of our basis as compared to a regular subspace simulation using two methods. The first one quantitatively compares how well our linear modes approximates the actual linear modes as shows in Figure 2.9 . The second one is a qualitative comparison of the simulation generated by our method and a similar mesh undergoing subspace simulation as per [BJ05]. (See Video) We note that in this example our method has a visual performance similair to original subspace at a fraction of the computation cost. We shows two rounds of incremental stitching in this example - one which partially stitches the mesh and the other which completely stitches the mesh. Even though there is some loss of the high frequency modes which result in our method not able to replicate some of the more extreme deformations, the results are visually plausible.

Comparison with [YXG$^+$13] : We also directly compared our method with the boundary-aware multi-domain subspace method proposed by Yang and colleagues. The fork example shown in Fig. 2.14 is used for the comparison. Without modal warping, boundary-aware subspace method suffers from excessive elongation artifacts as demonstrated by the fork in the extreme left. The artifact is corrected using modal warping as

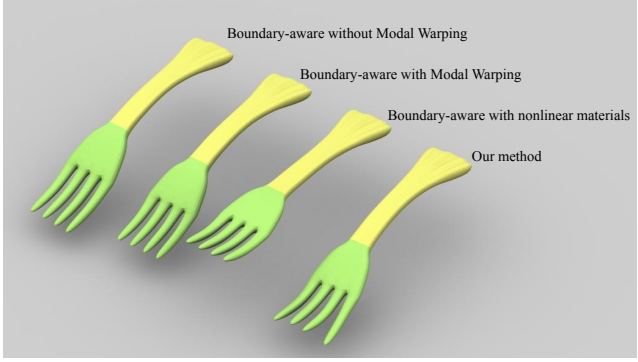
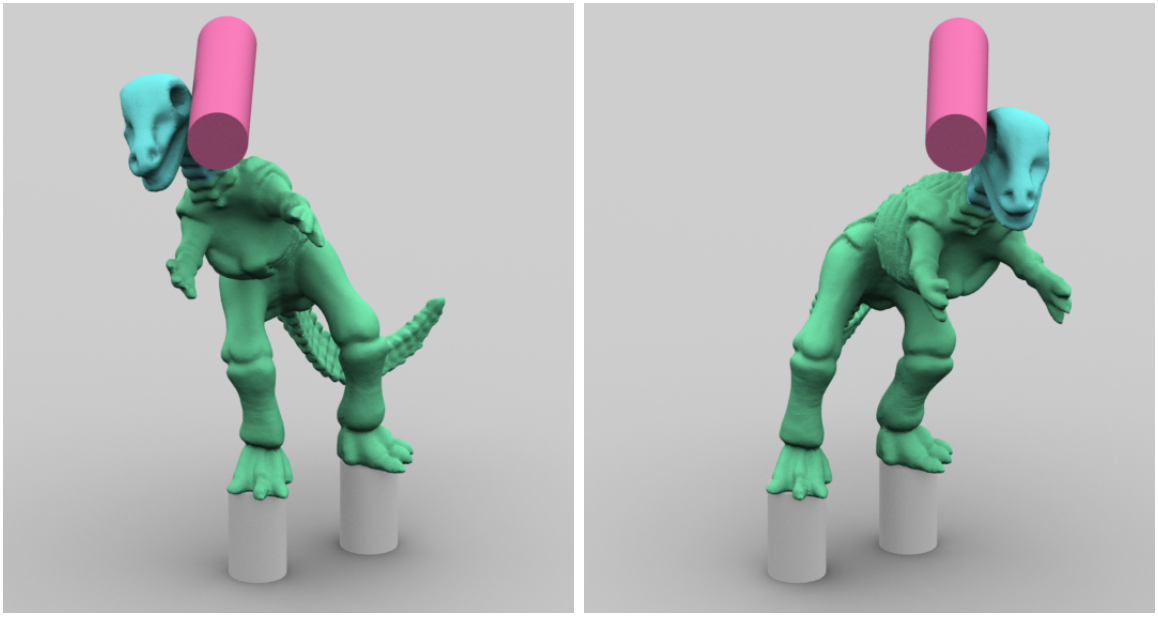


Figure 2.14: Comparison with boundary-aware subspace method.

shown in the second (from left) fork. Our method of basis generation (both linear and non-linear) is approximately twice as fast compared to that of boundary-aware subspace method, which only computes linear basis. From the figure, we can see that the fork with boundary-aware subspace can bend slightly more than ours because it is modeled with linear elasticity. However the advantage of our method is its ability to support nonlinear materials. Compared to boundary-aware multi-domain methods where non-linear material completely fail to move the fork (as shown in the third fork) , our method can gracefully handle any arbitrary materials.

Each of our examples demonstrate one possible use of our method. The ball example demonstrates how our method can be used for controlling stiffness of different materials. The stomach example shows us the potential of this method in virtual surgery applications. The plant example (both using two domains as well as three domains) shows its usefulness for domain decomposition methods. Finally the Fork and the Dinosaur example shows the application of our method in terms of real time basis recalculation speed and the ability of the method to handle complex collision scenarios.



(a) (b)

Figure 2.15: An extension of the Dinosaur example which shows how our system can handle deformations scenarios such as those involving collisions.

Incremental editing. Since our system incrementally recalculates the deformation subspace, it is naturally suitable for handling multiple stitching steps. For example, when we stitch the second branch to the plant model as shown in Figure 2.16b, we can recalculate the subspace using the new basis calculated after stitching the first branch. Similarly, when sewing the cut on a stomach model as shown in Figure 2.17, we can incrementally recalculate the subspace twice: once to form a partially stitched model and once to form a fully stitched model. Incremental subspace reconstruction can effectively reduce the computational cost, when user needs to add multiple stitches over time. That being said, recalculating the subspace too many times can cause large error accumulation. In the future, we would like to know how to quickly evaluate the subspace quality and when to completely recalculate the subspace instead.

Model	Model Statistics #Vert, #Tet, #Spr	Time Original	Time Ours	Speedup
Stomach	37K, 118K, 30	16.89s	1.10s	**15.3**×
Plant	13K, 43K, 30	4.35s	0.42s	**10.3**×
Fork	13K, 55K, 40	6.03s	0.67s	**9.0**×
Dinosaur	57K, 192K, 38	14.12s	2.65s	**5.3**×
Oball	16K, 60K, 160	7.72s	2.30s	**3.4**×

(a) The cost of recalculating linear Modes.

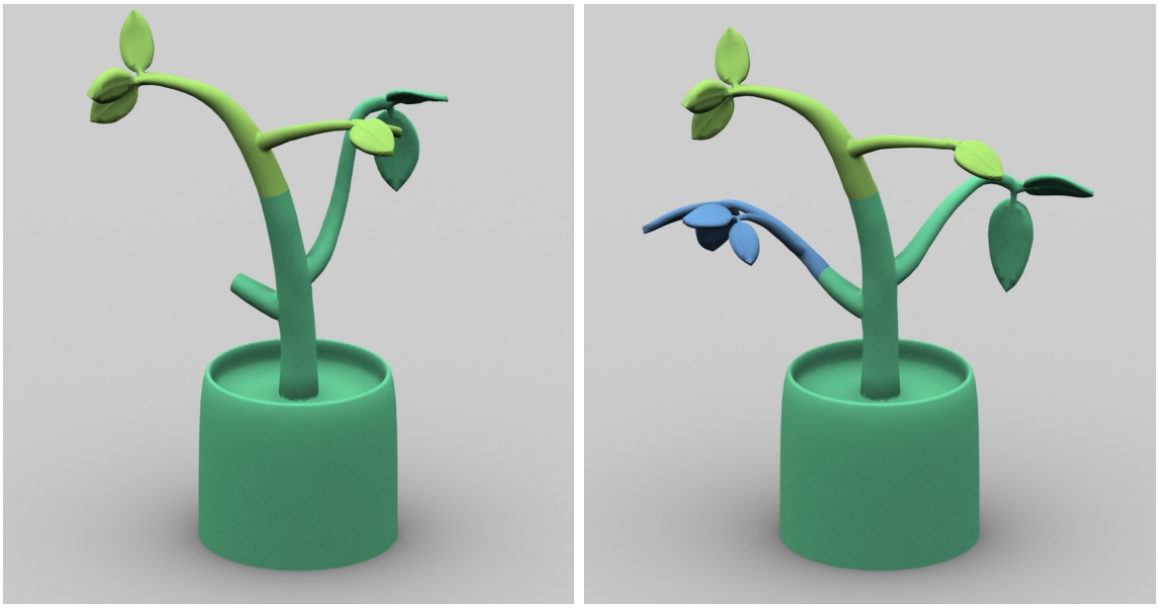
Model	Original #Deri	Original Time	Ours #Deri	Ours Time	Speedup
Stomach	201	26.08s	201	5.14s	**5.0**×
Plant	91	6.82s	91	0.89s	**7.6**×
Fork	143	6.50s	143	1.20s	**5.4**×
Dinosaur	91	34.09s	91	4.21s	**8.1**×
Oball	143	6.54s	143	1.25s	**5.2**×

(b) The cost of recalculating modal derivatives.

Model	Standard SVD #in	Standard SVD Time	Randomized SVD #in	Randomized SVD Time	Speedup
Stomach	231	15.76s	231	2.29s	**6.9**×
Plant	121	3.00s	121	0.80s	**3.8**×
Fork	173	3.75s	173	0.91s	**4.1**×
Dinosaur	121	13.74s	121	3.04s	**4.5**×
Oball	173	4.22s	173	0.98s	**4.3**×

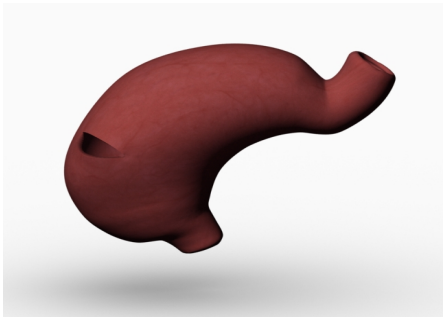
(c) The cost of final PCA reduction.

Table 2.1: Model and timing statistics.

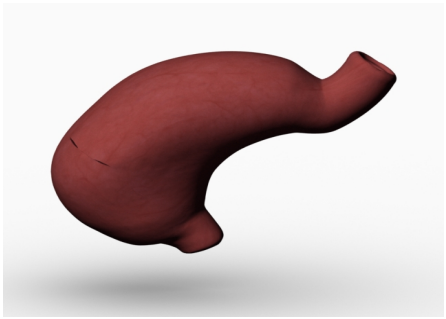


(a) Two Domains (b) Three domains

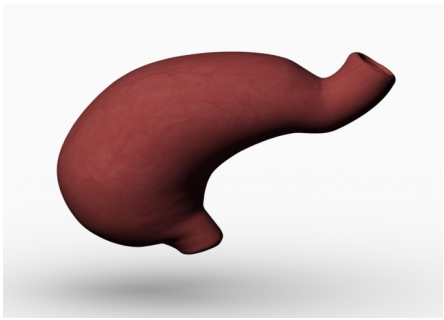
Figure 2.16: Multiple branches. Our system incrementally recalculates the deformation subspace of the whole plant model, when the branches are stitched to the model in multiple steps.



(a) Before stitching



(b) Partially stitched



(c) Fully stitched

Figure 2.17: The stomach example. A cut on the original mesh can be partially or fully stitched in simulation, depending on the constraints.

2.7 Limitations

Our method provides a fast way to construct nonlinear deformation modes for subspace simulation. Although the method is based on [BJ05], the result is not meant to be the same as the modes constructed directly from the new mesh using their technique. If the deformation modes are incrementally updated when the mesh changes constantly, the error can be accumulated and the result can become unpredictable. The performance of our method relies on the assumption that the mesh is slightly changed each time. If the mesh is too significantly changed, our method needs more computational time (see Figure 2.11 for the relationship between the number of stitches and the corresponding speedup) and it may be even slower than just calculating the subspace from scratch. When adding new stitches, we assume that vertices are co-located and the springs have zero lengths. In other words, we cannot stitch two arbitrary vertices without aligning them ahead of time. Finally, when stitching unconstrained domains to constrained ones, we cannot allow constrained domains to have large deformations, or we cannot deform unconstrained domains correctly.

Our subspace simulation reuses the cubature samples calculated for the original subspace. If the mesh is changed significantly, cubature approximation will be poor and we must recalculate the samples as well, which is computationally expensive. Another point to note is that the success of the cubature reuse is greatly dependent on the accuracy of the original cubature training. In general we should ensure that the cubature error during training is as low as possible. This can be ensured by adjusting the parameters during cubature optimization. While such a setting may increase the overall training time, we must note that it is a one-time cost and such a well-trained cubature (with low error) will allow our algorithm to reuse the cubatures effectively even after many small incremental changes.

Chapter 3: Interactive Two-Way Shape Design of Elastic Bodies

3.1 Introduction

An elastic body deforms under external loads. This property makes elastic shape design uniquely challenging, due to the discrepancy between the shapes before and after applying loads. Traditionally, elastic shape design is handled in a forward fashion: the designer first models the body without considering external loads, and then runs quasistatic simulation to check how the body gets deformed under loads. The designer must repeat this process multiple times, until he/she becomes satisfied with the outcome. A forward elastic shape design system is straightforward to implement, given the fact that quasistatic simulation has been extensively studied for many years. However, the repeating process requires heavy user intervention and consumes considerable computational resources. Forward elastic shape design is also counter-intuitive, since the designer cannot directly interact the actual shape under the influence of external loads.

A more natural and intuitive strategy is to allow the designer to manipulate the deformed shape under influence first, and then estimate the rest shape that achieves the deformed shape later. This inverse design strategy is useful not only to designers, but also to animators and surgeons, who need to know the rest shapes of sagging objects captured from the real world, for animation production and virtual surgery. Unfortunately, inverse elastic shape

(a) The target shape (b) The rest shape (c) The model made in the target shape (d) The model made in the rest shape

Figure 3.1: Based on fast simulation and inverse simulation techniques, our system allows the designer to interactively edit and examine the quasistatic shape and the rest shape of a piranha model at the same time, as shown in (a) and (b). We show physical validation of our system by fabricating the model in both shapes using a rubber-like tango black material. Under the influence of gravity, the model fabricated in the rest shape in (b) sags to the desired quasistatic shape in (a), as predicted by our system.

design requires to solve the inverse problem of quasistatic simulation, which turns out to be significantly more difficult than quasistatic simulation itself. Previous research on this topic was largely focused on lower dimensional cases [Had06, DJBDDT13], specific elastic material models [CZXZ14, WWY$^+$15], or small deformation [SPV$^+$16]. Inverse quasistatic simulation of generic elastic models remains as a challenging problem, as far as we know.

In this chapter, we present a new elastic shape design system that utilizes the power of both the CPU and the GPU. The key features of our system are:

- **Two-way design.** The system enables elastic shape design in both forward and inverse ways. In other words, the designer can modify either the rest shape or the quasistatic shape, and the system generates the other shape on the fly.

- **Flexibility.** Our system flexibly handles a wide range of nonlinear elastic models, including the neo-Hookean model, the Mooney-Rivlin model, and the spline-based model using principal stretches [XSZB15].

- **Efficiency.** After user editing, the system typically finishes the modeling process of a mesh with 73K tetrahedra in 0.5 seconds, as Figure 3.1 shows. This is substantially faster than any existing system, especially for inverse shape design.

The technical contributions involved in the development of our system can be summarized as follows.

- **Inverse quasistatic simulation.** We present the Jacobian matrix evaluation scheme for inverse quasistatic simulation of generic elastic materials. Based on this scheme, we discovered that, inverse simulation, as a special shape optimization problem, can be directly solved by Newton's method.

- **Initialization by projective dynamics.** Although our simulators are efficient, they are still unable to follow fast shape changes made by the user. To address this problem, we propose a real-time shape initialization approach based on the recent projective dynamics technique.

- **A heterogeneous system.** To achieve interactive system performance, we develop a heterogenous structure that distributes the computational workload to both the CPU and the GPU. The use of this structure offers another level of parallelization among processors.

Our experiment demonstrates that the whole system is fast, robust, and convenient for the designer to use in both forward and inverse shape design cases. It flexibly handles a variety

of nonlinear elastic material models, and its performance can be potentially improved by other numerical techniques in the future, such as multi-grid.

3.2 Related Work

Elastic body simulation. Elastic body simulation has been an important graphics research topic, since the pioneer work by Terzopoulos and colleagues [TPBF87]. Today's physics-based simulators often use implicit time integration, since explicit time integration can cause the numerical instability issue [OBH02, TBHF03]. To use implicit time integration, a simulator needs the Jacobian matrix of the force with respect to the vertex position. Teran and colleagues [TSIF05] proposed a Jacobian matrix evaluation scheme for hyperelastic materials and applied it in quasistatic simulation. Another Jacobian matrix scheme based on principal stretches was proposed by Xu and collaborators [XSZB15].

Physics-based models are flexible and close to real-world materials, but their computational costs are not so affordable by real-time applications. Because of that, researchers are getting more interested in fast constraint-based simulation techniques, such as strain limiting [Pro96, TPS09, WOR10], shape matching [MHTG05, RJ07], and position-based dynamics [MÖ8, KCMF12, MCKM14]. Liu and collaborators [LBOK13] found that the potential energy of a spring can be interpreted as a compliant edge constraint. This observation guided them to the development of an iterative local/global mass-spring simulator, whose result converges to the solution of implicit time integration. Bouaziz and colleagues [BML$^+$14] generalized this idea into projective dynamics, by formulating the elastic energy of an element as a geometric constraint. For highly stiff problems, Tournier and colleagues [TNGF15] combined forces and constraints into a better conditioned linear system with a larger problem size. To implement projective dynamics on the GPU, Wang [Wan15] explored the

Jacobi-preconditioned gradient descent method, and Fratarcangeli and colleagues [FTP16] advocated the use of multi-color Gauss Seidel. Recently, Wang and Ying [WY16] generalized Jacobi-preconditioned gradient descent to accurately and efficiently simulate generic nonlinear elastic materials, thanks to a series of divergence avoidance techniques.

Elastic shape design. While forward elastic shape design can be solved by quasistatic simulation as many researchers explored, inverse elastic shape design, fundamentally as inverse quasistatic simulation, is a much more complex problem. If the elastic shape is made of strands and curves, this problem can be well solved under reduced dynamics models [Had06, DJBDDT13]. When the goal does not require the rest shape result to be strictly consistent with the deformed shape input, the problem can be solved by augmented Lagrangian methods as Skouras and collaborators [STBG12] suggested. For inverse quasistatic simulation of neo-Hookean volumetric models, Chen and colleagues [CZXZ14] developed a Jacobian matrix evaluation formula and suggested to use an asymptotic numerical method. Wang and collaborators [WWY$^+$15] found that the inverse problem can be solved as a backward simulation process for co-rotational linear materials. Shin and colleagues [SPV$^+$16] studied inverse quasistatic simulation of generic elastic materials, when the deformation is small. Recently, Bartle and collaborators [BSK$^+$16] designed a fixed point method to solve the inverse problem for cloth patches. Their method converges fast mostly in the first few iterations.

Graphics researchers have also studied inverse elastic shape design, when the shape is parameterized by edge lengths [TKA11], control parameters [BKS$^+$12, CMT$^+$12], or joints [STC$^+$13, PTC$^+$15]. Their techniques typically involve the use of the Gauss-Newton method, by formulating shape optimization as nonlinear least square problems. For inverse quasistatic simulation, Chen and colleagues [CZXZ14] pointed out that the Gauss-Newton

method is not a suitable option, since it is often too expensive to construct its matrix, especially when the mesh is in high resolution. Outside of the computer graphics community, researchers investigated the inverse elastic problem mostly for identifying material parameter distributions or buried objects [BC05]. The developed techniques are either limited to linear elasticity, or too computationally expensive for interactive applications.

3.3 Background

Let $\mathbf{X} \in \mathbb{R}^{3N}$ and $\mathbf{x} \in \mathbb{R}^{3N}$ be rest and deformed vertex position vectors of an elastic body with N vertices. The quasistatic equilibrium state can be described using a nonlinear system:

$$\mathbf{f}(\mathbf{x}, \mathbf{X}) = \mathbf{0}, \tag{3.1}$$

in which the total force $\mathbf{f}$ is a nonlinear function of $\mathbf{x}$ and $\mathbf{X}$. Given $\mathbf{X}$, the goal of quasistatic simulation is to find $\mathbf{x}$ that satisfies Equation 3.1.

Numerical solutions. A typical solution to a nonlinear system is Newton's method, which successively refines the result $\mathbf{x}^{(k+1)}$ in the k+1-th iteration as:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \left[\mathbf{J}_{\mathbf{x}}^{(k)}\right]^{-1} \mathbf{f}(\mathbf{x}^{(k)}, \mathbf{X}), \tag{3.2}$$

where $\mathbf{J}_{\mathbf{x}} = \partial \mathbf{f} / \partial \mathbf{x}$ is the Jacobian matrix of $\mathbf{f}$ with respect to $\mathbf{x}$. Since we can define $\mathbf{f}$ as the negative gradient of the total potential energy, we can treat $\mathbf{J}_{\mathbf{x}}$ as the negative Hessian matrix of the potential energy. Therefore, $\mathbf{J}_{\mathbf{x}}$ must be symmetric.

The issue with Newton's method is that it is too computationally expensive to solve $\mathbf{J}_{\mathbf{x}}^{-1}\mathbf{f}$ in every iteration. A natural idea explored by quasi-Newton methods and conjugate gradient methods is to use an approximation of $\mathbf{J}_{\mathbf{x}}$, whose inverse can be easily calculated. These methods often require extensive use of dot product operations, making them unfriendly

with parallelization. In a special case, if we replace $\mathbf{J_x}$ by the identity matrix, we reduce Newton's method to the gradient descent method, whose convergence rate is known to be low. Alternatively, we can replace $\mathbf{J_x}$ by a carefully tuned constant matrix, which can be pre-factored as a fast runtime solver. This approach has demonstrated its effectiveness on projective dynamics [BML+14], but not so much on generic hyperelastic models [LBK16]. Recently, Wang and Yang [WY16] proposed to replace $\mathbf{J_x}$ by its diagonal. Their method can robustly and efficiently handle generic hyperelastic models, after using a dynamically adjusted step length.

Elastic force and matrix evaluation. A crucial step involved in many aforementioned methods is the evaluation of the internal elastic force and its Jacobian matrix. According to continuum mechanics, the elastic forces $\{\mathbf{g}_0, \mathbf{g}_1, \mathbf{g}_2, \mathbf{g}_3\}$ applied at the vertices of a tetrahedron is:

$$\mathbf{G} = [\mathbf{g}_1 \;\; \mathbf{g}_2 \;\; \mathbf{g}_3] = \mathbf{P}\mathbf{B}_m = \mathbf{P}\,[\mathbf{b}_1 \;\; \mathbf{b}_2 \;\; \mathbf{b}_3], \qquad \mathbf{g}_0 = -\sum_{i=1}^{3} \mathbf{g}_i, \tag{3.3}$$

in which $\mathbf{P}$ is the first Piola-Kirchhoff stress and $\mathbf{b}_i$ stands for the average area-weighted normal of the triangles adjacent to vertex i. Let $\mathbf{F} = \mathbf{D}_s\mathbf{D}_m^{-1}$ be the deformation gradient, where $\mathbf{D}_s$ and $\mathbf{D}_m$ are the edge vector matrix: $\mathbf{D}_s = [\mathbf{x}_1 - \mathbf{x}_0 \;\; \mathbf{x}_2 - \mathbf{x}_0 \;\; \mathbf{x}_3 - \mathbf{x}_0]$ and $\mathbf{D}_m = [\mathbf{X}_1 - \mathbf{X}_0 \;\; \mathbf{X}_2 - \mathbf{X}_0 \;\; \mathbf{X}_3 - \mathbf{X}_0]$. After singular value decomposition $\mathbf{F} = \mathbf{U}\mathbf{D}\mathbf{V}^\mathsf{T}$, we can compute the stress tensor $\mathbf{P}$ as:

$$\mathbf{P} = \mathbf{U}\mathbf{P}(\mathbf{D})\mathbf{V}^\mathsf{T}, \tag{3.4}$$

where the function $\mathbf{P}(\mathbf{D})$ represents the constitutive relationship between the principal stretches in $\mathbf{D}$ and the stress tensor. Given Equation 3.3 and 3.4, Xu and colleagues [XSZB15] proposed to evaluate the elastic Jacobian matrix from:

$$\frac{\partial \mathbf{G}}{\partial \mathbf{x}} = \frac{\partial \mathbf{G}}{\partial \mathbf{F}}\frac{\partial \mathbf{F}}{\partial \mathbf{x}} = \left(\frac{\partial \mathbf{P}}{\partial \mathbf{F}}\mathbf{B}_m\right)\frac{\partial \mathbf{F}}{\partial \mathbf{x}}. \tag{3.5}$$

Since both $\mathbf{B}_m$ and $\partial\mathbf{F}/\partial\mathbf{x}$ are invariant to $\mathbf{x}$, our focus is on $\partial\mathbf{P}/\partial\mathbf{F}$. By definition, we can formulate the derivative of $\mathbf{P}$ with respect to one variable of $\mathbf{F}$ as:

$$\frac{\partial\mathbf{P}}{\partial F_{ij}} = \mathbf{U}\left\{\mathbf{\Omega}_{\mathbf{U}}^{ij}\mathbf{P}(\mathbf{D}) + \frac{\partial\mathbf{P}(\mathbf{D})}{\partial\mathbf{D}}\frac{\partial\mathbf{D}}{\partial F_{ij}} + \mathbf{P}(\mathbf{D})\mathbf{\Omega}_{\mathbf{V}}^{ij}\right\}\mathbf{V}^{\mathsf{T}}, \tag{3.6}$$

in which,

$$\mathbf{\Omega}_{\mathbf{U}}^{ij} = \mathbf{U}^{\mathsf{T}}\frac{\partial\mathbf{U}}{\partial F_{ij}}, \quad \mathbf{\Omega}_{\mathbf{V}}^{ij} = \frac{\partial\mathbf{V}^{\mathsf{T}}}{\partial F_{ij}}\mathbf{V}. \tag{3.7}$$

To evaluate Equation 3.6, we calculate $\mathbf{U}$, $\mathbf{V}$, $\mathbf{P}(\mathbf{D})$, and $\partial\mathbf{P}(\mathbf{D})/\partial\mathbf{D}$ by their definitions. The additional terms needed by Equation 3.6 are $\mathbf{\Omega}_{\mathbf{U}}^{ij}$, $\mathbf{\Omega}_{\mathbf{V}}^{ij}$, and $\partial\mathbf{D}/\partial F_{ij}$. According to [PL00], we have:

$$\mathbf{U}^{\mathsf{T}}\frac{\partial\mathbf{F}}{\partial F_{ij}}\mathbf{V} = \mathbf{\Omega}_{\mathbf{U}}^{ij}\mathbf{D} + \frac{\partial\mathbf{D}}{\partial F_{ij}} + \mathbf{D}\mathbf{\Omega}_{\mathbf{V}}^{ij}. \tag{3.8}$$

Since $\mathbf{D}$ is a diagonal matrix, $\partial\mathbf{D}/\partial F_{ij}$ is diagonal as well. Meanwhile, $\mathbf{\Omega}_{\mathbf{U}}^{ij}$ and $\mathbf{\Omega}_{\mathbf{V}}^{ij}$ must be skew-symmetric, because $\mathbf{\Omega}_{\mathbf{U}}^{ij} + (\mathbf{\Omega}_{\mathbf{U}}^{ij})^{\mathsf{T}} = \partial(\mathbf{U}^{\mathsf{T}}\mathbf{U})/\partial F_{ij} = \mathbf{0}$. Based on these observations, we obtain $\partial\mathbf{D}/\partial F_{ij}$ from the diagonal elements of $\mathbf{U}^{\mathsf{T}}(\partial\mathbf{F}/\partial F_{ij})\mathbf{V}$. The remaining off-diagonal elements form a linear system, whose solution gives the six unique elements of $\mathbf{\Omega}_{\mathbf{U}}^{ij}$ and $\mathbf{\Omega}_{\mathbf{V}}^{ij}$. To avoid the system from being singular when two or more principal stretches are close, we simply place a limit on the determinant of the system matrix.

3.4 Inverse Quasistatic Simulation

In this section, we will study inverse quasistatic simulation, the key component in inverse elastic shape design. Specifically, given $\mathbf{x}$, we would like to find $\mathbf{X}$ that satisfies $\mathbf{f}(\mathbf{x},\mathbf{X}) = \mathbf{0}$. Before we discuss the applicability of the numerical methods, we will formulate the Jacobian matrix of the force with respect to the rest shape: $\mathbf{J}_{\mathbf{X}} = \partial\mathbf{f}/\partial\mathbf{X}$, based on our prior knowledge in Section 3.3.

3.4.1 Jacobian Matrix Evaluation

We consider two Jacobian matrices for the inverse problem: the Jacobian of the elastic force under a generic elastic material model and the Jacobian of the gravity force.

Elastic matrix evaluation. To evaluate the elastic Jacobian matrix for the inverse problem, our idea is to use the inverse of the deformation gradient:

$$\hat{\mathbf{F}} = \mathbf{F}^{-1} = \mathbf{D}_m \mathbf{D}_s^{-1} = \mathbf{V}\mathbf{D}^{-1}\mathbf{U}^{\mathsf{T}}, \tag{3.9}$$

which is a linear function of $\mathbf{X}$. We can obtain $\mathbf{J}_{\mathbf{X}}$ from:

$$\frac{\partial \mathbf{G}}{\partial \mathbf{X}} = \left(\frac{\partial \mathbf{P}}{\partial \hat{\mathbf{F}}}\mathbf{B}_m\right)\frac{\partial \hat{\mathbf{F}}}{\partial \mathbf{X}} + \mathbf{P}\frac{\partial \mathbf{B}_m}{\partial \mathbf{X}}, \tag{3.10}$$

where only $\partial\hat{\mathbf{F}}/\partial\mathbf{X}$ is invariant to $\mathbf{X}$. To evaluate the first component in Equation 3.10, we compute the derivative of $\mathbf{P}$ with respect to one variable of $\hat{\mathbf{F}}$:

$$\frac{\partial \mathbf{P}}{\partial \hat{F}_{ij}} = \mathbf{U}\left\{\hat{\mathbf{\Omega}}_{\mathbf{U}}^{ij}\mathbf{P}(\mathbf{D}) + \frac{\partial \mathbf{P}(\mathbf{D})}{\partial(\mathbf{D}^{-1})}\frac{\partial(\mathbf{D}^{-1})}{\partial \hat{F}_{ij}} + \mathbf{P}(\mathbf{D})\hat{\mathbf{\Omega}}_{\mathbf{V}}^{ij}\right\}\mathbf{V}^{\mathsf{T}}, \tag{3.11}$$

in which,

$$\hat{\mathbf{\Omega}}_{\mathbf{U}}^{ij} = \mathbf{U}^{\mathsf{T}}\frac{\partial \mathbf{U}}{\partial \hat{F}_{ij}}, \quad \hat{\mathbf{\Omega}}_{\mathbf{V}}^{ij} = \frac{\partial \mathbf{V}^{\mathsf{T}}}{\partial \hat{F}_{ij}}\mathbf{V}. \tag{3.12}$$

By taking the derivative of the singular value decomposition of $\hat{\mathbf{F}}$ and multiplying the result with $\mathbf{V}^{\mathsf{T}}$ and $\mathbf{U}$, we get:

$$\left(\mathbf{V}^{\mathsf{T}}\frac{\partial \hat{\mathbf{F}}}{\partial \hat{F}_{ij}}\mathbf{U}\right)^{\mathsf{T}} = \hat{\mathbf{\Omega}}_{\mathbf{U}}^{ij}\mathbf{D}^{-1} + \frac{\partial \mathbf{D}^{-1}}{\partial \hat{F}_{ij}} + \mathbf{D}^{-1}\hat{\mathbf{\Omega}}_{\mathbf{V}}^{ij}. \tag{3.13}$$

As before, $\partial\mathbf{D}^{-1}/\partial\hat{F}_{ij}$ is diagonal, and both $\hat{\mathbf{\Omega}}_{\mathbf{U}}^{ij}$ and $\hat{\mathbf{\Omega}}_{\mathbf{V}}^{ij}$ are skew-symmetric. Therefore, we calculate $\partial\mathbf{D}^{-1}/\partial\hat{F}_{ij}$ from the diagonal elements of $(\mathbf{V}^{\mathsf{T}}\partial\hat{\mathbf{F}}/\partial\hat{F}_{ij}\mathbf{U})^{\mathsf{T}}$, and $\hat{\mathbf{\Omega}}_{\mathbf{U}}^{ij}$ and $\hat{\mathbf{\Omega}}_{\mathbf{V}}^{ij}$ from the remaining off-diagonal elements after solving a linear system.

The evaluation of the second component in Equation 3.10 needs $\partial \mathbf{B}_m / \partial \mathbf{X}$. Since $\mathbf{B}_m$ is made of area-weighted triangle normals and each area-weighted normal is equal to the cross product of two edge vectors, we formulate the derivative of one normal first:

$$\frac{\partial (\mathbf{X}_3 - \mathbf{X}_2) \times (\mathbf{X}_1 - \mathbf{X}_2)}{\partial \mathbf{X}_1} = \mathbf{R}^*_{32}, \tag{3.14}$$

in which $\mathbf{R}^*_{32}$ is the skew-symmetric cross product matrix of edge $\mathbf{X}_3\mathbf{X}_2$. Once we obtain all of these cross product matrices, we assemble them into $\partial \mathbf{B}_m / \partial \mathbf{X}$ and obtain the second component.

Gravitational matrix evaluation. The gravity force, as a function of the rest volume, also depends on $\mathbf{X}$. Therefore, we should consider its contribution to the Jacobian matrix of the total force as well. For simplicity, we use a lumped mass model, which equally distributes the mass of a tetrahedron to its four vertices. Based on this model, we calculate the Jacobian matrix of the gravity force contributed by tetrahedron t to every vertex i as:

$$\mathbf{J}^{t,i}_{\mathbf{X}} = \frac{\rho_t}{24} \mathbf{g} \frac{\partial \mathtt{det}(\mathbf{D}_m)}{\partial \mathbf{X}}, \tag{3.15}$$

in which ρ_t is the mass density, $\mathbf{g}$ is the gravity acceleration, and $\frac{1}{6}\mathtt{det}(\mathbf{D}_m)$ gives the tetrahedron volume. By definition, we can calculate the derivative of the determinant from its 2×2 sub-matrices. Once we calculate each $\mathbf{J}^{t,i}_{\mathbf{X}}$, we sum them up to form the whole gravitational Jacobian matrix.

The importance of matrix components. Our research reveals that the first component of the elastic matrix dominates the total Jacobian matrix. In average, it contributes more than 99.7 percent of the total Jacobian matrix, while the other two components contribute less than 0.3 percent. This phenomenon makes us wonder whether we can ignore the other components in inverse quasistatic simulation. Figure 3.2 demonstrates that the answer is no.

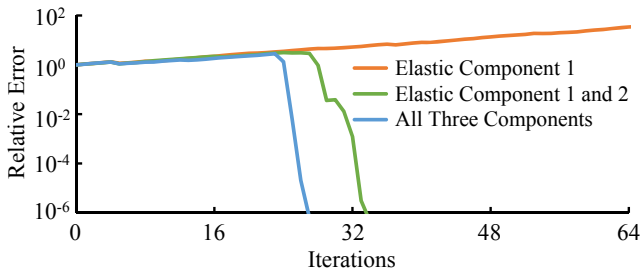


Figure 3.2: The effects of Jacobian matrix components on the convergence of Newton's method. This plot shows that it is necessary to evaluate all of the three Jacobian matrix components, even though the contributions of the second elastic component and the gravitational component are small. By default, we choose the armadillo example for evaluation purposes.

When we ignore the second elastic component and the gravitational component, Newton's method fails to converge. When we ignore the gravitational component only, Newton's method converges within 34 iterations. Finally, when we use all of three components, Newton's method converges within 27 iterations. Since the computation overhead of the two additional components is small, we choose to evaluate the whole Jacobian matrix in practice.

3.4.2 Numerical Solutions

An interesting question is whether the numerical methods used for quasistatic simulation are still applicable to the inverse problem. Here we must keep in mind that the Jacobian matrix of the inverse problem cannot be interpreted as the Hessian matrix of a potential energy. Therefore, it may not be symmetric.

Our first experiment is to test nonlinear descent methods on the GPU, including nonlinear conjugate gradient and preconditioned gradient descent, both of which have demonstrated their robustness and performance in quasistatic simulation [WY16]. Figure 3.3 shows that gradient descent is effective in the first few iterations. But once the error becomes small,

it diverges if we do not significantly reduce the step length. The performance of nonlinear conjugate gradient is even worse, probably because conjugate gradient is not suitable for asymmetric matrices. We note that here the cost of a gradient descent iteration is higher than that cost in [WY16], since gradient descent must evaluate the Jacobian matrix in every iteration to avoid more severe divergence issues.

Since some nonlinear descent methods can be considered as applying one step of an iterative solver in every iteration of Newton's method, our next experiment is to choose Newton's method and try different iterative solvers and different numbers of iterative steps. Figure 3.3 shows that if we use 128 steps of a Jacobi solver in each Newton iteration, the error curve behaves just like that of preconditioned gradient descent: the error decreases only in the first few iterations. The reason it runs faster is because it does not need frequent force or matrix evaluation. The asymmetric nature of the system matrix inspires us to test the biconjugate gradient stabilized method next. Figure 3.3 shows that this method still cannot reach a small error, even if we use 1,024 iterative steps per Newton iteration. It also verifies the high computational cost of the biconjugate gradient stabilized method on the GPU, due to extensive use of inner and matrix-vector products.

In the end, we believe that Newton's method with a direct solver provides the best way to solve inverse quasistatic simulation. Our simulator chooses to evaluate forces and matrices on the GPU, and solve each linear system by a LU solver on the CPU. In most cases, the simulator can reduce the error to a low level within one second, as Figure 3.3 shows. To further improve its robustness and efficiency, we will investigate error metrics, step lengths, and matrix evaluations in the rest of this subsection.

Errors and step lengths. When the initialization is away from the solution, Newton's method needs a sufficiently small step length to protect it from the divergence issue. One

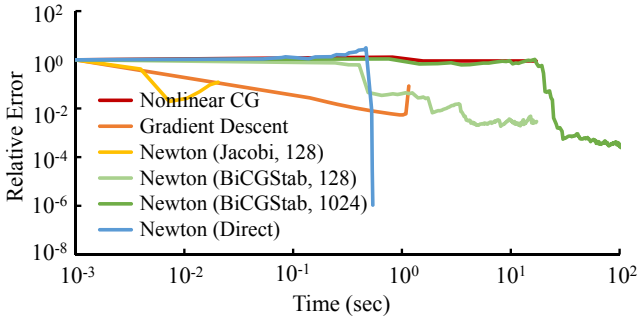


Figure 3.3: The performances of different numerical methods solving the same inverse simulation problem. Due to the complex nature of the problem, many methods fail to converge. In contrast, Newton's method with a direct solver converges within one second.

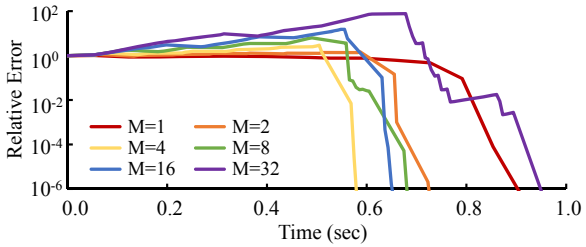
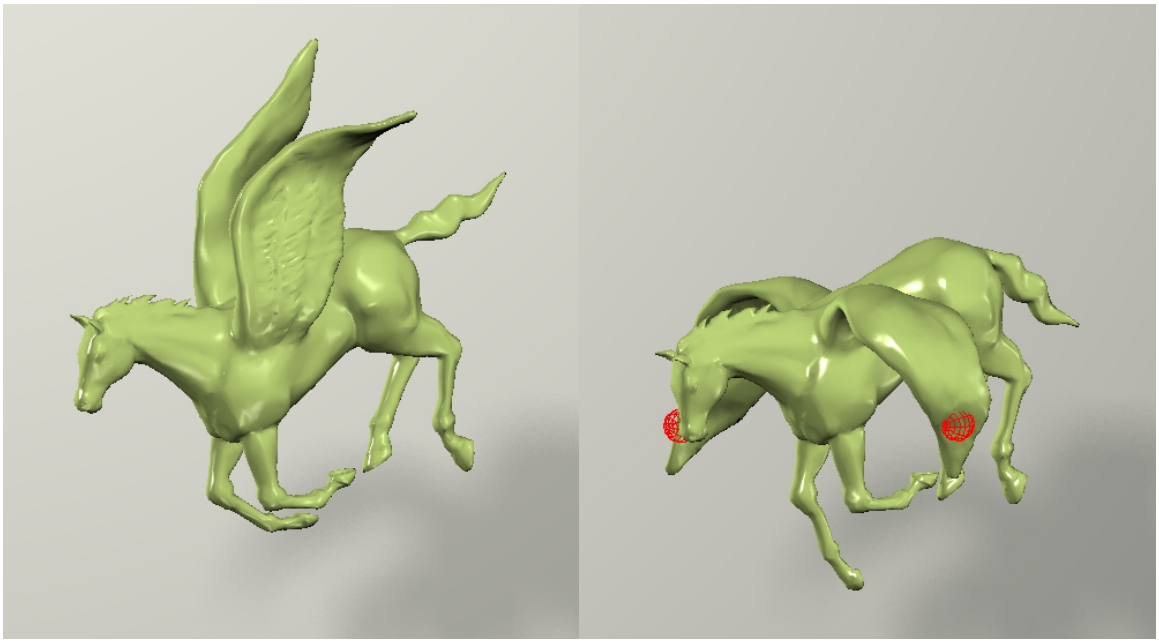


Figure 3.4: The performances of Newton's method when it chooses different M values. Newton's method converges slowly when M is too small or too large. It converges the fastest when $M = 4$.

way to find the step length is to perform backtracking line search based on the Armijo rule, which requires the system energy to decrease monotonically. Since we cannot define inverse quasistatic simulation as an energy minimization problem, we replace the energy in the Armijo rule by the residual error, i.e., the magnitude of the residual force in Equation 3.1. We also change the rule to accept a step length, as long as it does not introduce too much error increase. The reason we do not strictly enforce an error decrease condition is because the error behaves differently from the energy and a strict condition can cause an unnecessarily small step length, as we experienced in our experiment. Figure 3.3 demonstrates that

although the error reported from Newton's method increases in the first few iterations, it quickly drops afterwards.

Skipping matrix evaluations. Newton's method spends a large portion of its computational time on Jacobian matrix evaluation and factorization. Therefore, a natural idea of improving its performance is to skip matrix evaluations and reuse factored matrices from previous iterations. Researchers have explored this idea as high-order Newton's methods [CHMT10] before. In this work, we simply perform matrix evaluation and factorization every M iterations, and use the same factored matrix to construct the linear systems in the M iterations. Figure 3.4 shows that Newton's method converges the fastest when $M = 4$. If M is too large, this approach will be not efficient either, since the error cannot decrease sufficiently in many iterations.



(a) The rest shape (b) The quasistatic shape

Figure 3.5: The pegasus example. Our system allows the user to interactively modify the quasistatic shape by new fix constraints (in red) and examine the rest shape at the same time.

3.5 An Interactive System

The goal of our system is to achieve interactive elastic shape design in two ways. In the forward way, the user interacts with the rest shape and the system updates the quasistatic shape in real time. In the inverse way, the user modifies the quasistatic shape and the system generates the rest shape accordingly.

Given the techniques described in Section 3.3 and 3.4, it is straightforward to implement basic functionalities of our system. Specifically, we use the preconditioned gradient descent method [WY16] to handle quasistatic simulation in forward elastic shape design. For inverse quasistatic simulation, we use Newton's method solved by a direct solver, as mentioned in Subsection 3.4.2. Our system can adopt any tool for interactive shape editing. In practice,

we simply use another quasistatic simulator, which allows the user to modify the shapes by adding new fix constraints (in red), as shown in Figure 3.5.

A basic way of implementing our system is to run interactive shape editing and forward/inverse quasistatic simulation at the same time. However, such a system is highly susceptible to the divergence issue, since the simulation result in the current iteration can become a bad initialization in the next iteration, when the designer makes dramatic shape changes. Even though this issue can be addressed by using small step lengths, it reduces the system performance and makes divergence avoidance sophisticated. Our solution is a projective dynamics initialization method for estimating shape deformation in both forward and inverse directions. This method not only allows the designer to preview the outcome in real time, but also speeds up the convergence of simulation and inverse simulation after each user modification.

3.5.1 Shape Initialization by Projective Dynamics

Without loss of generality, here we study the initialization during inverse elastic shape design. It is straightforward to adjust the proposed method for forward elastic shape design, by simply swapping the quasistatic shape with the rest shape.

Let $\mathbf{x}_0$ and $\mathbf{X}_0$ be a pair of the quasistatic shape and the rest shape obtained from the last inverse simulation process. Let $\mathbf{x}$ be the newly modified quasistatic shape and $\mathbf{X}$ be the corresponding rest shape to be found. We argue that the transformation from $\mathbf{x}$ to $\mathbf{X}$ should be similar to the transformation from $\mathbf{x}_0$ to $\mathbf{X}_0$. For every tetrahedron, this similarity can be measured by the difference between two deformation gradients:

$$E_t^{\mathsf{sim}} = \left\| \mathbf{D}_m \mathbf{D}_s^{-1} - \mathbf{D}_{m0} \mathbf{D}_{s0}^{-1} \right\|^2 = \left\| \mathbf{A}_t \mathbf{X} - \mathbf{F}_{t0}^{-1} \right\|^2, \tag{3.16}$$

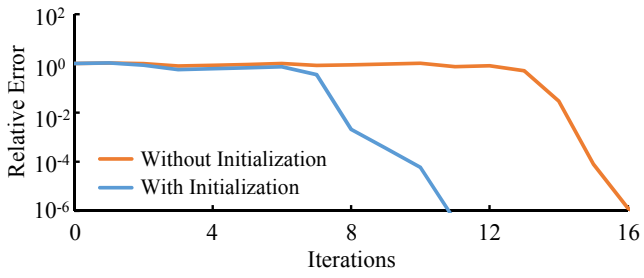


Figure 3.6: The convergence of Newton's method with and without using shape initialization. Based on projective dynamics, our shape initialization technique speeds up inverse quasistatic simulation by approximately 30 percent, as shown in this plot.

where $\mathbf{A}_t$ is the matrix that converts $\mathbf{X}$ into the deformation gradient and $\mathbf{F}_{t0}^{-1}$ is the deformation gradient from $\mathbf{x}_0$ to $\mathbf{X}_0$. We also argue that the tetrahedron should not be too stretched or compressed from its input shape. This strain limiting condition can be quantified using another energy metric:

$$E_t^{\mathsf{limit}} = \|\mathbf{B}_t\mathbf{X} - \mathbf{p}_t(\mathbf{B}_t\mathbf{X})\|^2, \tag{3.17}$$

in which $\mathbf{B}_t\mathbf{X}$ is the deformation gradient of the tetrahedron from the input shape to $\mathbf{X}$, and $\mathbf{p}_t(\mathbf{B}_t\mathbf{X})$ is its projection after isotropic strain limiting [WOR10]. Finally, we use another energy to keep the vertices fixed in simulation remain fixed in initialization:

$$E^{\mathsf{fix}} = \left(\mathbf{X} - \mathbf{X}^{(0)}\right)^{\mathsf{T}}\mathbf{S}\left(\mathbf{X} - \mathbf{X}^{(0)}\right), \tag{3.18}$$

in which $\mathbf{X}^{(0)}$ is the starting $\mathbf{X}$, and $\mathbf{S}$ is a diagonal matrix containing nonnegative fixing weights. By putting the three energies together, we form an energy minimization problem that finds an optimal $\mathbf{X}$ as: $\mathbf{X} = \arg\min\left\{\frac{k_0}{2}\sum_t E_t^{\mathsf{sim}} + \frac{k_1}{2}\sum_t E_t^{\mathsf{limit}} + \frac{1}{2}E^{\mathsf{fix}}\right\}$, which leads to a nonlinear system:

$$\begin{aligned} \mathbf{M}\mathbf{X} &= \mathbf{S}\mathbf{X}^{(0)} + \sum_t \left(k_0\mathbf{A}_t^{\mathsf{T}}\mathbf{F}_{t0}^{-1} + k_1\mathbf{B}_t^{\mathsf{T}}\mathbf{p}_t\right), \\ \mathbf{M} &= \mathbf{S} + \sum_t \left(k_0\mathbf{A}_t^{\mathsf{T}}\mathbf{A}_t + k_1\mathbf{B}_t^{\mathsf{T}}\mathbf{B}_t\right). \end{aligned} \tag{3.19}$$

Here the nonlinearity comes from the dependence of $\mathbf{p}_t$ on $\mathbf{X}$. The projective dynamics technique [BML+14] proposes to solve Equation 3.19 as a linear system multiple times, by treating $\mathbf{p}_t$ as constant each time. Projective dynamics is fast not only because $\mathbf{M}$ is independent of $\mathbf{X}$ and it can be pre-factored by Cholesky decomposition for less runtime cost, but also because the linear system can be separated into three for the three coordinates of $\mathbf{X}$. Since the initialization is called repetitively during user interaction and strain limit violation is uncommon, we found it is sufficient to solve Equation 3.19 just once per frame on the CPU. Note that $\mathbf{M}$ is symmetric positive definite, so the linear system must have a solution. Figure 3.6 reveals the effect of shape initialization on the convergence of Newton's method handling the inverse simulation process that follows.

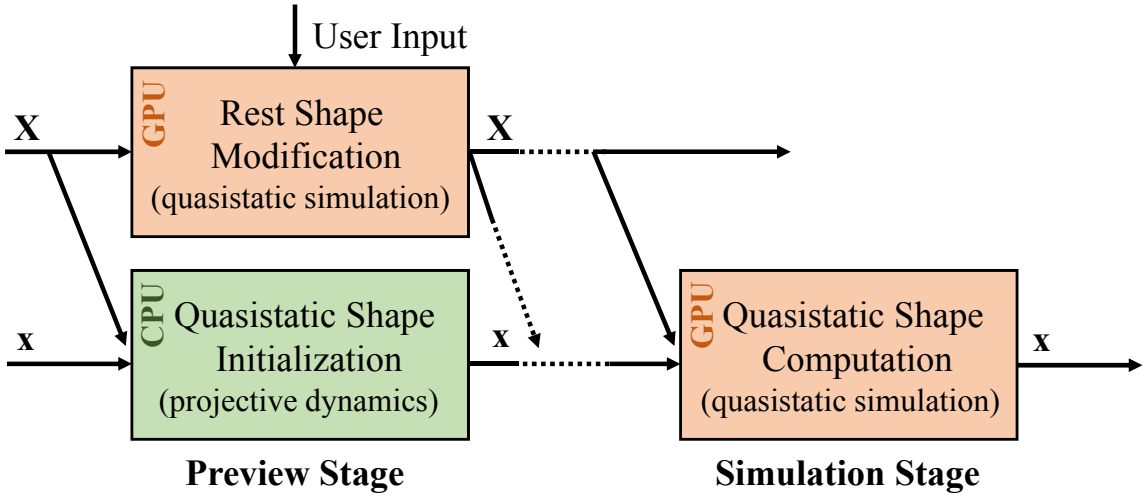
3.5.2 A Heterogeneous Structure

Based on the initialization method proposed in Subsection 3.5.1, we present our elastic shape design system as Figure 3.7 shows. This system divides the shape design process into two stages. In the preview stage, the system uses the initialization technique to estimate the other shape from the modified shape, for real-time preview purposes. Once the user stops editing and the modified shape converges, the system switches to the simulation stage and runs quasistatic or inverse quasistatic simulation to obtain the final optimized result.

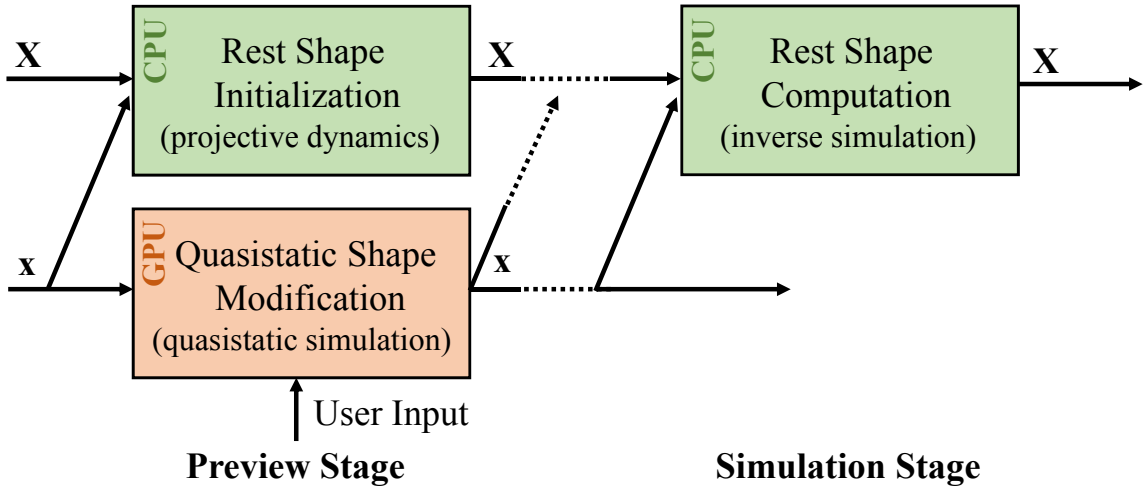
An interesting feature of our system is that shape modification is performed on the GPU, while shape initialization is performed on the CPU[1]. This provides another level of parallelization between the processors in the preview stage, so they do not need to wait each other. For the armadillo example, shape modification takes 4.1ms per frame and shape initialization takes 12.6ms per frame in average. The theoretical upper bound on the speedup

[1]Like inverse simulation, shape initialization runs the linear solver on the CPU. It still performs force and matrix evaluation on the GPU.

caused by CPU-GPU parallelization is 24.5 percent, while the actual speedup we observed from the experiment is approximately 18.0 percent, due to the overhead of multi-threading. We expect this speedup to be greater in the future, once the performance of a GPU receives more improvement than that of a CPU.



(a) Forward elastic shape design



(b) Inverse elastic shape design

Figure 3.7: The system pipeline. When the user performs modification on one shape during the preview stage, the system utilizes both the CPU and the GPU to initialize the other shape in real time. After that, it runs quasistatic or inverse quasistatic simulation to reach the final result of the other shape.

3.6 Results and Discussions

(Please watch the supplemental video for animation examples.) The implementation of our system uses the Intel MKL PARDISO library and the CUDA library. Our experiment runs on an Intel Core i7-5930K 3.5GHz processor and an NVIDIA GeForce GTX TITAN X Graphics Card. Table 3.1 summarized the statistics and the timings of our examples. In each frame, our quasistatic simulator runs 96 gradient descent iterations for shape modification, or 128 gradient descent iterations for forward simulation. The other two simulators, i.e., the projective dynamics simulator and the inverse simulator, run only one Newton iteration in each frame. By default, we choose the neo-Hookean model for our examples.

Comparison to asymptotic numerical methods (ANM). To compare the performance of our inverse simulator with that of an asymptotic numerical method [CZXZ14], we test both methods using the same example as shown in Figure 3.8. Our experiments shows that our simulator can generate its result in 0.6 seconds, while ANM needs 3.5 seconds. Both methods use the Intel MKL PARDISO library on the CPU. We note that ANM is currently designed for the neo-Hookean model only.

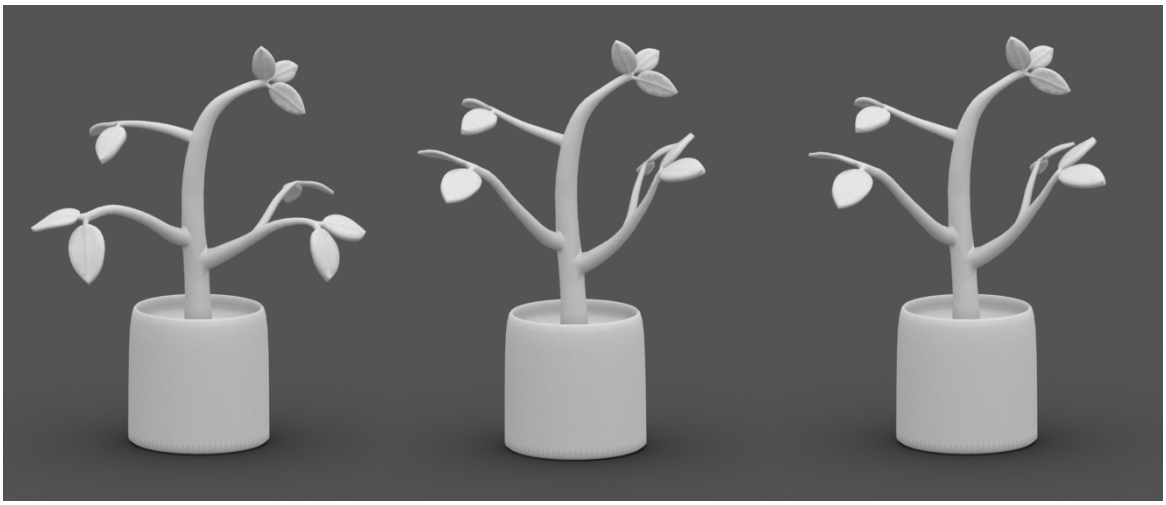
Validation by 3D printing. We validate the correctness of our simulation results by 3D printing, as shown in Figure 3.1 and 3.9. In both examples, we use a Stratasys Objet 30 Prime Printer with a rubber-like tango black material. Since we do not exactly know the physical properties of *tango black*, we print the model in the target shape (in Figure 3.1c) and estimate material parameters manually from forward simulation. Given these parameters, we run inverse simulation on the target shape (in Figure 3.1a) to get the rest shape (in Figure 3.1b). The printed model in the rest shape sags to the desired target shape, as shown in Figure 3.1d. The statue example demonstrates that our inverse simulator can handle

Name (#vert, #ele)	Preview Costs		Simulation Costs	
	Before	After	Forward	Inverse
Pegasus (12K, 49K)	14.9ms	12.1ms	12.0ms	15.8ms
Plant (14K, 47K)	14.8ms	12.1ms	11.8ms	15.5ms
Armadillo (14K, 55K)	16.7ms	13.7ms	12.6ms	20.4ms
Statue (16K, 60K)	19.0ms	14.7ms	14.1ms	23.0ms
Piranha (20K, 73K)	23.3ms	17.1ms	16.9ms	29.0ms

Table 3.1: Statistics and timings of our examples. This table summarizes the computational cost per frame in each stage. Here the two preview costs are the costs before and after implementing our CPU-GPU parallelization concept.

additional loads, which are invariant to the rest shape. In this example, we treat the input shape as the rest shape in Figure 3.9a, and run forward simulation to get the deformed shape in quasistatic equilibrium as Figure 3.9b shows. By running inverse simulation on the deformed shape, our system successfully recovers the rest shape.

Inverted elements. Since many elastic material models do not consider inverted element cases, researchers developed a variety of techniques to help quasistatic simulation handle inverted elements in the past. An interesting question is whether these techniques can help inverse quasistatic simulation handle inverted elements as well. Figure 3.10 shows the results of a box simulated by our inverse simulator. When the initialization is completely mirrored, the simulator fails to recover the mesh shown in Figure 3.10c, as expected. Unfortunately, techniques that work in quasistatic simulation, such as clamping the energy or the principal stretches [TSIF05, WY16], do not work in inverse quasistatic simulation as far as we have experienced. While this issue can be easily addressed by placing a strain limit on the initial mesh, we would like to know whether a better solution exists in the future.



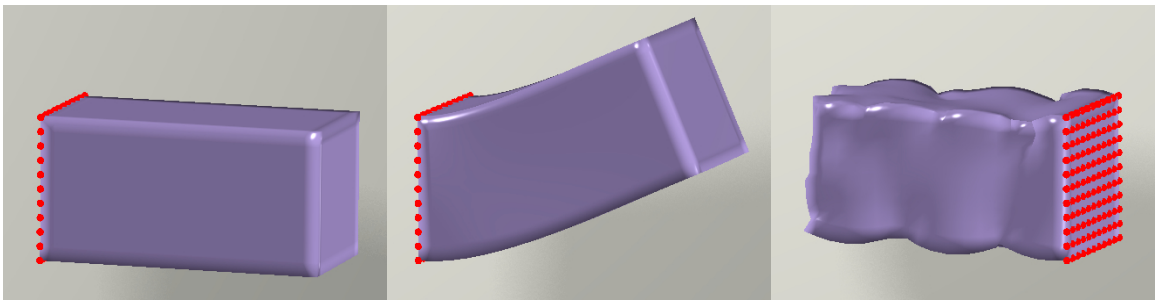
(a) The target shape (b) Our result (c) The result of ANM

Figure 3.8: The plant example. The result of our inverse simulator in (b) is identical to that of the asymptotic numerical method in (c).

Nonlinear elastic models. Figure 3.11 demonstrates the capability of our system in handling the armadillo example under different nonlinear elastic models. In this experiment, we treat the input shape as the quasistatic shape as shown in Figure 3.11a. Since our spline-based stVK model [XSZB15] is the most compliant one, its rest shape differs the most from the quasistatic shape as Figure 3.11b shows. We note that the original StVK model has little resistance to compression, so we incorporate an additional volumetric strain energy as proposed in [KTY09]. Figure 3.11c and 3.11d show that the results under the neo-Hookean model and the Mooney-Rivlin model are similar, because the system uses the same elastic modulus values. The Mooney-Rivlin result looks slightly closer to the quasistatic shape, due to an additional energy term.

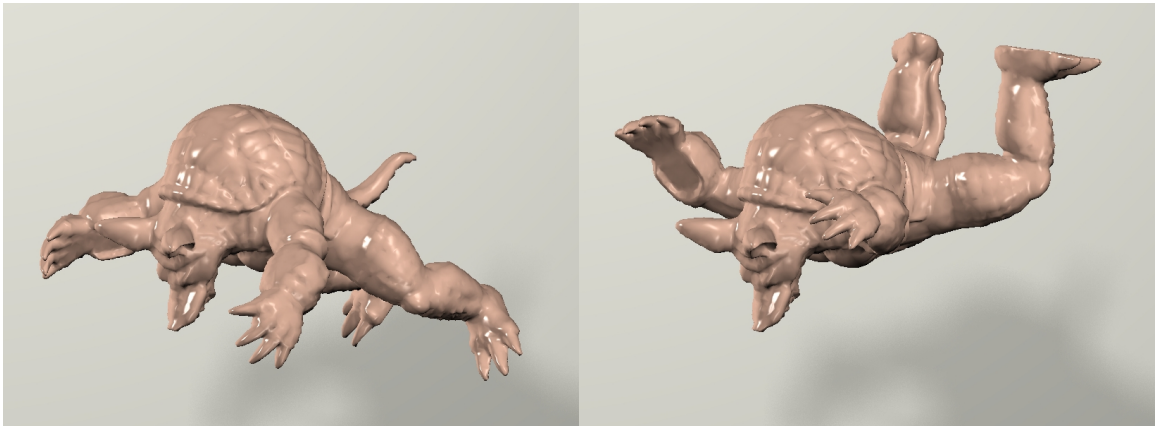
(a) The rest shape (b) The model made in the rest shape

Figure 3.9: The statue example. Our inverse quasistatic simulator can successfully recover the rest shape of this model in (a) from the deformed shape under an additional 500g load shown in (b).



(a) The target shape (b) The rest shape (c) The inverted shape

Figure 3.10: The box example. This example reveals that our inverse quasistatic simulation method cannot easily handle inverted element cases, as shown in (c).



(a) The quasistatic shape

(b) The result of spline-based StVK

(c) The result of neo-Hookean

(d) The result of Mooney-Rivlin

Figure 3.11: The armadillo example. This example compares the rest shape results of our inverse quasistatic simulator under different nonlinear elastic models.

3.7 Limitations

Perhaps the biggest limitation of our system is its requirement on the mesh quality. When the quality of the rest mesh is lower, forward elastic shape design needs smaller step lengths and more computational time to avoid the divergence issue. Interestingly, the mesh quality also affects inverse elastic shape design, if the modified mesh leads to a bad rest mesh. For example, when the user lifts the beam in the quasistatic shape as shown in Figure 3.12, the top white vertices will become more squeezed in the rest shape to count against gravity. This will cause oscillation or even divergence during the transition from the preview stage to the simulation stage, unless we eliminate the preview stage and always run Newton's method for inverse simulation. Our solution to this problem is to relax the fixing weights of the fixed vertices according to their distances to the free vertices, so that the free vertices do not get squeezed too much. Our experiment indicates that this solution effectively avoids oscillation or divergence issues.

Other limitations of the system include interruptive transition from the preview stage to the simulation stage, the need of a balanced workload between the CPU and the GPU, and lower convergence rates when it deals with stiffer or more nonlinear materials.

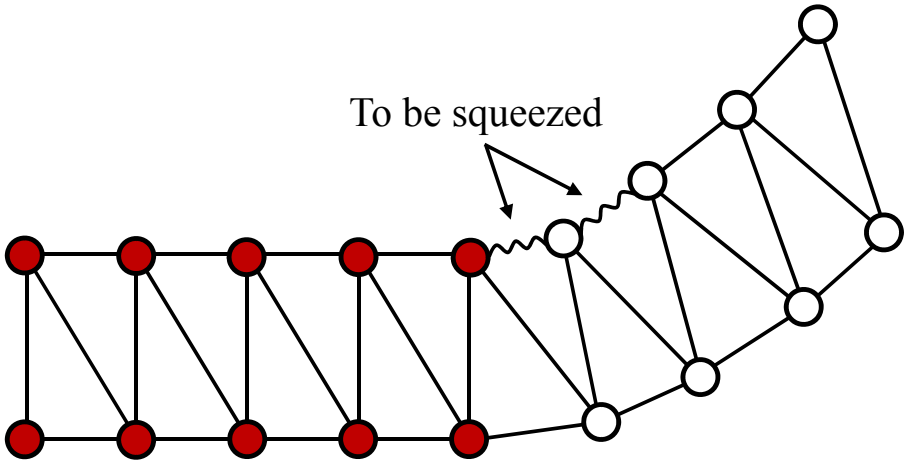
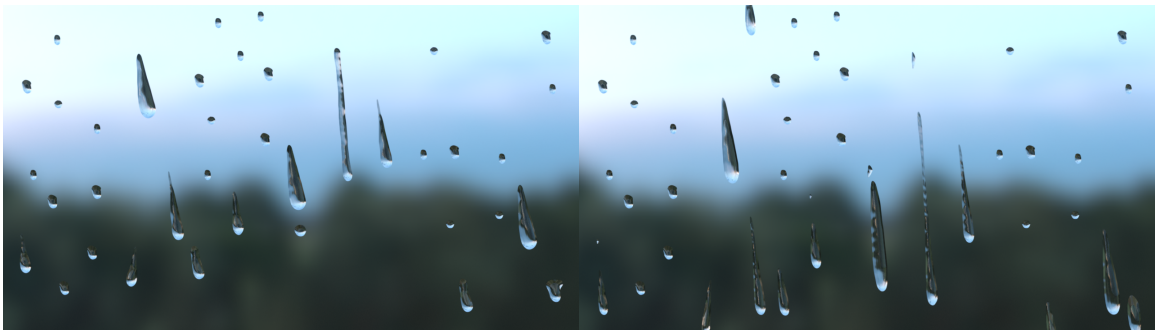


Figure 3.12: A challenging example. When the red vertices are fixed and the white vertices get lifted in the quasistatic shape, the white vertices must be lifted even more in the rest shape to fight against gravity. This causes the top white vertices to be squeezed and deteriorates the rest mesh quality.

Chapter 4: Deep Neural Network based Simulation of Small-Scale Liquid Flows on Solids

4.1 Introduction

Realistic simulation of small-scale liquid flows on solids, i.e., liquid drops and streamlets, is an important feature in virtual surgery, forensics (for simulating crime scenes), virtual painting, and entertainment applications. Unfortunately, small-scale liquid flows on solids are notoriously difficult to simulate, because of the surface tension. At a small scale, liquids, especially water, exhibit strong surface tension effect that places a stringent CFL condition on physics-based liquid simulation and demands a large computational cost to handle. While this issue can be remedied by ignoring interior flows and animating drop motions by the exterior surfaces only [ZWW$^+$12, DHB$^+$16], a much more critical issue exists: how to model the surface tension among liquid, air, and solid. At the contact front where the three phases meet, the surface tension effect is highly complicated [Fow64, Mit13]. For example, it is known that the surface tension acts differently when the contact front is advancing vs. receding, known as the *hysteresis* effect. The surface tension also relies on the material properties of the solid, which are not conveniently measurable due to varying factors, including moisture, roughness, and dirt. In computer graphics, researchers [WMT05, WMT07] tried to simulate these liquid-solid effects by a varying contact angle scheme.



(a) Before simulation (b) After simulation

Figure 4.1: A window example. The scene in this example contains 50 liquid drops simulated by our novel learning-based liquid simulator, specifically designed for small-scale flows on solid surfaces. The key component of this simulator is LSTM-based recurrent neural networks, trained by real-world flow data. Using the neural networks, our simulator efficiently predicts the motion of every liquid drop at the next time step in 0.1s. Integrated with geometric operations, the simulator realistically simulate complex liquid flow effects under surface tension, as shown in (b).

While their result looks plausible, it is still far from realistic and they fail to display the same level of complexity demonstrated by real-world drops.

An interesting idea of simulating complex natural phenomena is to use data acquired from the real world. Real-world data are free of artifacts caused by incompetent physical models and they are able to capture complex dynamics well. However, data-driven liquid simulation is not easy, due to the volatile nature of real-world liquids. The recent advance in the machine learning technology inspires graphics researchers to revisit this idea from a learning-based perspective. While the limited research in learning-based simulation is largely focused on generic simulation of large-scale liquid bodies, small-scale liquid flows and their interactions with solid surfaces remain unexplored. Large-scale liquids are volatile and their flows are turbulent. This causes many previous learning-based simulation techniques to be developed for predicting fluid behaviors in a local neighborhood, such as

using convolutional neural networks (CNN) [GLI16, YYX16, CT17]. In contrast, the motion of a small-scale liquid body, i.e., a liquid drop, is more predictable and often free of heavy topological events. Hence we prefer to simulate small-scale liquids in a unique way, rather than in a unified way with large-scale liquids.

In this chapter, we advocate the use of deep neural networks to predict the motion of a small-scale liquid body on surfaces as a whole. We believe this is reasonable practice, because: 1) small-scale liquid flows are difficult to simulate in a physics-based way; 2) small-scale liquid flows are free of turbulence and they are more predictable. We also would like to promote the idea of using a Lagrangian form to represent small-scale liquids. Compared with large-scale liquid bodies, small-scale liquids do not experience a significant number of merging or splitting events, therefore, we can afford handling them by simple operations. The outcome of this research is a DNN-based simulator for animating small-scale liquids on solid surfaces. To this end, we have made the following technical contributions.

- *Data preparation.* We present a convenient way to acquire raw liquid flow data from the real world. We then develop practical techniques to convert raw data into training data under a novel Lagrangian representation, which models an individual liquid drop solely by its contact front.

- *Neural networks.* Given the training data, we study the development and the training of three neural networks. These networks are responsible for predicting the contour of a liquid drop's contact front, the color gradient at the contact front, and finally whether a contact front breaks at the next time step.

- *DNN-based simulation.* Our simulator combines neural networks with simple geometric/topological operations to achieve realistic simulation of water drops. The

implementation of this simulator involves a series of research on shape reconstruction, initialization, topological events, and flows on curved surfaces.

Our experiment shows that the simulator is efficient, robust, parallelizable, convenient to use and its cost is linearly scalable to the number of liquid drops, as expected. The animation results generated by our simulator are realistic and contain many complex details that cannot be easily handled by physics-based methods.

4.2 Related Work

Physics-based small-scale liquid simulation. Physics-based liquid simulation has been extensively studied by graphics researchers in last two decades. Here we narrow our discussions down to physics-based liquid simulation at a small scale. Similar to general liquid simulation, the simulation of small-scale liquids can be developed in three ways: volumetric simulation [EFFM02, WMT05, WMT07], particle-based simulation [AAT13, HWZ+14, CKIW15, JS17], and mesh-based simulation [TWGT10, CWSO13, ZWW+12, ZLQF15, DHB+16]. While researchers have expressed their strong interests in free liquid surface flows in these works, they paid much less attention to liquid flows on surfaces. One of the reasons is because the surface tension at the contact front where liquid, air, and solid meet can be highly complicated. In materials science and computational physics, this surface tension can be quantified by the contact angle between the liquid-air surface and the solid surface. Wang and colleagues [WMT05] developed a virtual surface method to control the contact angles along the contact front in volumetric simulation. This enabled them to model the hydrophobicity of a solid and produce contact angle hysteresis effects in animation. Later they [WMT07] extended this idea to the simulation of water drops on unstructured surface meshes. Recently, Zhang and collaborators [ZWW+12] studied the

virtual surface method in surface-only liquid simulation. In general, small-scale liquid flows on solids are still difficult to be simulated realistically and efficiently, not only because of the computational cost, but also because of the complex physical phenomena that cannot be precisely predicted by mathematical models.

Learning-based fluid simulation. The use of machine learning in fluid simulation starts to emerge recently, but it is still sparse in the computer animation community. Ladicky and colleagues [LJS+15] used random regression forests to correct particle positions in incompressible SPH simulation. Guo and collaborators [GLI16] trained a convolutional neural network (CNN) to efficiently approximate steady flow. Yang and colleagues [YYX16] accelerated a pressure projection solver in Eulerian fluid simulation by CNN. Tompson and collaborators [TSSP16] incorporated multi-frame information into a similar network structure for long-term accuracy improvement. Bonev and colleagues [BPT17] proposed to predict surface deformation in water simulation by neural networks and soon Um and collaborators [UHT17] used it for modeling water splashes. To synthesize smoke animation, Chu and colleagues [CT17] trained feature descriptors with CNN for fast example database query. As mentioned before, the difference between large-scale liquids and small-scale liquids restricts the applicability of these techniques in the simulation of small-scale liquid flows.

4.3 Background

4.3.1 Fluid Simulations

Fluid simulation is one of the most studied fields both in the areas of Computational Fluid Dynamics (CFD) as well as from the computer graphics perspective. There exists a

Table 4.1: Notations associated with the Navier Stokes Equation

Notation	Definition
$\vec{u}$	Velocity
$\vec{g}$	Acceleration due to Gravity
p	Pressure
ρ	Density
ν	Kinematic Viscosity

myriad number of approaches that can simulate the motion of fluids just like that of solid mechanics. We will briefly review one such approach.

Navier Stokes Equation : The incompresseible Navier-Strokes equations are a set of partial differential equations that always hold true throughout the fluids. It is actually composed of two equations. We will briefly review the notations associated with fluid simulation in Table 4.1 and then consider the actual equations below. Note that this equation holds true for each particle in the fluid being simulated. The primary component of any fluid particle is its associated pressure (a scalar term)$p(\vec{x},t)$ and its velocity (a vector term) $\vec{u}(\vec{x},t)$both of which varies in space and time.

$$\frac{\partial \vec{u}}{\partial t} = -\vec{u}\cdot\nabla\vec{u} - \frac{1}{\rho}\nabla p + \nu\nabla\cdot\nabla\vec{u} + \vec{g} \tag{4.1}$$

$$\nabla\cdot\vec{u} = 0 \tag{4.2}$$

- *Momentum Equation* : Equation 4.1 is commonly referred to as the Momentum Conservation Equation which is a restatement of Newton's second law of motion. The left hand side of the equation is just the acceleration of the fluid particle ($\frac{\partial\vec{u}}{\partial t}$).

The Navier Stokes equation says that this acceleration is caused by four components which are enlisted on the right hand side. We quickly itemize each component of the right hand side of the equation.

  - Self-advection term($\vec{u} \cdot \nabla \vec{u}$) - The velocity of the fluid particle $\vec{u}$ causes the transport of objects along its flow. This same velocity is what carries the fluid itself and it is known as the self-advection term.
  - Pressure term ($\frac{1}{\rho} \nabla p$) - If you apply pressure to a fluid, it is transmitted gradually throughout the mass of the fluid. An application of localized pressure causes a change in the velocity (hence acceleration) of the fluid. The pressure term accounts for this.
  - Viscosity term ($\nu \nabla \cdot \nabla \vec{u}$) - "Thicker" fluids like maple syrup are more resistant to flow than "thinner" fluids like water. The quantification of this resistance to flow is a property of the fluid known as Kinematic Viscosity. This resistance results in an acceleration in fluid particles known as Viscosity or diffusion.
  - External Forces ($\vec{g}$) - Liquids are acted upon by their own weights hence force of gravity also causes acceleration. This accounts for the fourth term in the equation.

- *Incompressibility Equation* : Equation 4.2 describes what is known as a divergence-free vector field or an incompressible flow. In a nutshell, incompressible fluids means that their volume doesn't change during the liquid flow. More specifically, for an infinitesimally small region of the fluid, the total volume will always remain constant. One of the consequences of the incompressibility assumption is that density of the fluid is always constant in time and space.

Smoothed Particle Hydrodynamics (SPH) - SPH is one of the most common methods of simulating fluid flow using particles. In SPH, we treat the entire volume of fluid as a combination of discrete particles [MCG03]. Of course to model the effects of pressure and viscosity, it is important to also model forces that occur because of interaction among neighboring particles. In order to keep the computation tractable, we limit this interaction to some finite radius (r). The amount of influence a particle has on its neighbor is modeled by a smoothing kernel which is a function dependent on the radius. The choice of Kernel for this is known as the `poly6` kernel defined in Equation 4.3

$$W^{ij}_{poly6}(r) = \frac{315}{64\pi h^9}(h^2 - r^2)^3 \tag{4.3}$$

In the above equation, h is a constant (known as the *smoothing radius*). As in the Navier Stokes equation, every particle of the fluid has an associated position $\vec{x_i}$ and an associated velocity $\vec{u_i}$. The goal of the SPH algorithm is to compute the new values of this position and velocity vectors with each timestep. This is done as follows.

1. Compute the set of neighbors of the particle i (call this set as $\mathcal{N}_i$) which are in the sphere of influence of the particle. These are the particles which exert interactive forces and hence are necessary for the computation of pressure and viscous forces. To accelerate this computation, we can use an efficient space hashing grid that needs to be dynamically recomputed at every step [OD08].

2. Compute density of the particle by taking into account its interaction with the neighboring particles. The density of the particle ρ_i is calculated as:

$$\rho_i = \sum_{j \in \mathcal{N}_i} m_j W^{ij}_{poly6}(||\vec{r_{ij}}||) \tag{4.4}$$

where $\vec{r_{ij}}$ is the distance vector between particle i and j. Clearly $\vec{r_{ij}} = \vec{x_i} - \vec{x_j}$

3. Compute the pressure exerted by the particle using the ideal gas law.

$$p_i = K(\frac{\rho_i}{\rho_0} - 1)^\gamma \tag{4.5}$$

where K is a pressure constant and ρ_0 is the reference density of the fluid. Care should be taken to avoid negative pressures by clipping values if needed.

4. Compute the acceleration of all the particles involved in the SPH grid as follows.

$$\vec{a}_i = -\sum_{j \in \mathcal{N}_i} (\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2}) \hat{r_{ij}} \nabla W^{ij}_{poly6}(||\vec{r_{ij}}||) \tag{4.6}$$

Interestingly note that we haven't still taken viscosity into account in this computation. Only mass, density and pressure of the fluid are the components of this acceleration.

5. Compute the average velocity of the particle due to artificial viscosity as follows.

$$\bar{u_i} = \sum_{j \in \mathcal{N}_i} \vec{u}_j \frac{m_j}{\rho_j} W^{ij}_{poly6}(||\vec{r_{ij}}||) \tag{4.7}$$

6. Dampen the velocity to account for the viscous forces as follows.

$$\vec{u}_i \leftarrow (1 - \nu)\vec{u}_i + \nu \bar{u}_i \tag{4.8}$$

7. Compute the new velocity of the particle as follows.

$$\vec{u}_i \leftarrow \vec{u}_i + \Delta t \vec{a}_i \tag{4.9}$$

8. Compute the new position of the particle as follows.

$$\vec{x}_i \leftarrow \vec{x}_i + \Delta t \vec{u}_i \tag{4.10}$$

9. Enforce any constraints if needed as per the problem.

One of the most important things to note about this process is that this can be parallelized very easily. Hence it is quite suitable for implementation on the GPU. A result of using this algorithm is shown in Figure 4.2

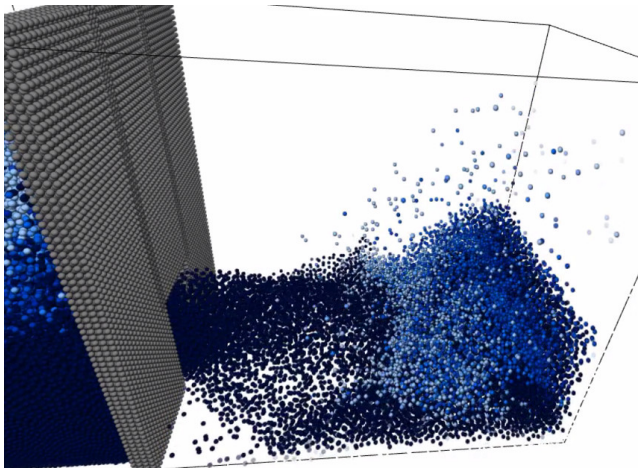


Figure 4.2: SPH Example (Image courtesy of [Guy15])

4.3.2 Neural Networks

In its most broad sense, Neural Networks are universal function approximators. Given a set of input data $x_1, x_2, ...x_n$ and a set of expected output data $y_1, y_2, ...y_n$, a Neural Network tries to learn a (non)-linear function approximation $\hat{f}$ such that

$$\hat{f}(x_1) = y_1, \hat{f}(x_2) = y_2, ..., \hat{f}(x_n) = y_n \tag{4.11}$$

Each of these x_i (and in turn y_i) could be a single-integer, a vector, a matrix or a tensor. Neural Networks, more specifically Multi-Layered Perceptron(MLP) have been around in the Machine Learning community as early as 1980s [RHW88]. However, most of the machine learning community focused on statistical techniques and it was not until the work of [HOT06, KSH12] in late 2000's that Neural Network returned to research forefront. Convolutional Neural Networks(CNN) are extremely popular in the domain of image recognition and segmentation for their ability to model spatial relationships in images, [CMS12, SZ14]. However, in this dissertation we focus on Recurrent Neural Networks

(RNN) which is used specifically for their ability to model temporal data. RNNs are mostly used in the domain of speech [MGM15] and language modeling [SNS15].

RNN : In order to motivate the discussion on Recurrent Neural Networks, we must first understand why they are needed in time sequence based task. Recall from Eq. 4.11 about Neural Networks being universal function approximators. In a sequence data however, both the input data x_i and output data y_i are now time-sequences. That is $x_i = \{x_i^1, x_i^2, ...x_i^t\}$ where the number in the super-script denotes the data-point in the sequence. While it may seem straightforward to extend previous perceptron work to such recurrent connections, the main problem in training such a network by the method of gradient descent is that of Exploding and Vanishing Gradient. This problem is discussed in details in [PMB13]. While Exploding and Vanishing gradient issues can occur with any deep neural network, they are particularly persistant in recurrent networks where the loss (or explosion) of derivatives during backpropagation can result in poor long term dependence learning. While Exploding gradient problem can be fixed by gradient clipping, the vanishing gradient problem is more difficult to solve. The following section discusses a special type of RNN cell known as LSTM which avoids the problem of vanishing gradient by careful derivative propagation.

LSTM(Long Short Term Memory) RNN : In this portion of the thesis, we will focus on a special type of RNN called Long Short Term Memory. An LSTM RNN is basically a recurrent neural network which alleviates the problem of Vanishing Gradients. Conceptually, it is easier to think of LSTMs in terms of memory. That is, at every state in the sequence, the Neural Network has a chance to "forget" information from previous states or retain it. This is demonstrated in the overall cell state diagram Figure. 4.3 where the purple circle

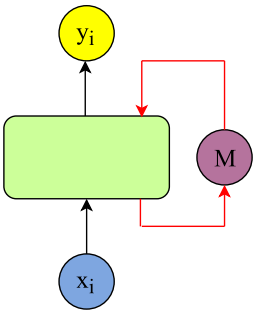


Figure 4.3: Overview of an LSTM Cell

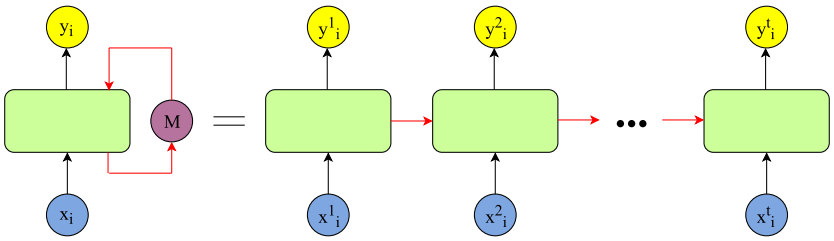


Figure 4.4: LSTM Cell Unrolled to show the sequence states

denotes this symbolic "memory" that allows the LSTM to remember or forget information between states.

Now in order to understand this in more depth, we can "unroll" this overall LSTM cell into its component sequences as shown in Figure 4.4. Notice that the superscripts denote the sequence state numbers. The red line denoting the memory state now runs through all the unrolled cells and in each cell it has to make a decision (based on the current input) to modify the prior information it retained. This red line is known as cell state and it is crucial to simulating the memory like features in an LSTM. We look at the interior of one unrolled LSTM cell to understand how it can modify the cell state as shown in Figure. 4.5.

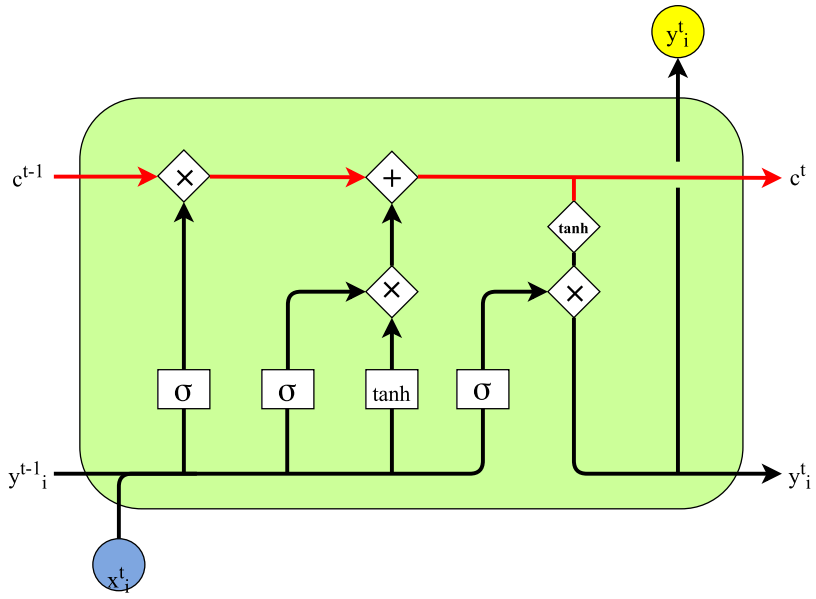


Figure 4.5: The interior of an LSTM Cell (Image inspired from [Ola15]). The diamonds denote pointwise vector operations while the rectangles denote neural network operations.

Emulating "Memory" : In this section, we discuss briefly how an LSTM cell simulates memory. As mentioned before, this is achieved using the cell state tensor which allows the LSTM cell to add or remove previous information. To decide what data from the previous cell state tensor c^{t-1} to keep, the LSTM uses it along with the present input x_i^t in the so called "forget gate". Next it has to decide what new data to add to the cell state. This is done using a combination of the current input x_i^t, the previous output y_i^{t-1} and the cell state data after processing through the forget gate. This is done in the second sigmoid layer and the third tanh layer. Finally, the LSTM also needs to output the value at the current state y_i^t. This is decided by the current input x_i^t, the previous output y_i^{t-1} and the final newly modified cell state c^t. In this way, each cell of an LSTM simulates a memory that allows persistence and long term dependency modeling which otherwise wouldn't be possible because of the vanishing gradient issues.

4.4 Data Preparation

Before we discuss the architecture of our deep neural networks in Section 4.5, we would like to present the acquisition and the representation of our training data from real-world experiments. The choices we make here are important to the success of the whole simulator and we will discuss the rationales behind them.

4.4.1 Data Acquisition

As mentioned in Section 4.1, we choose to use real-world experiments, rather than physics-based simulation, to generate the training data, since they are more accessible and more accurate in describing liquid flow behaviors. The setup of our data capture system is shown in Fig. 4.6. It consists of a 8" $\times$ 12" test area made of a non-adsorbent matte plastic sheet, a dripping device that drops dyed water onto the top of the test area, and a commodity camera that records the video of liquid flows at 720p and 240 FPS. We set the test area on an inclined ramp with a fixed angle of 30°, so that liquid drops can flow at a suitable speed. The surrounding environment uses indirect natural light to minimize reflected highlight spots on liquid drops. We calibrate the camera and perform basic image rectification to remove camera distortion.

The length of each recorded video clip varies from 150 to 1200 frames. In total, we capture 70 video clips, each of them contains one to three drops. We typically do not include videos that contain too fast drops, since they can interfere with the overall network training. Nearly one third of the clips contain splitting or merging events. From the video clips, we obtain a data set that contains 80K training sets. In addition to the training clips, we capture 10 more video clips for testing and evaluation purposes.

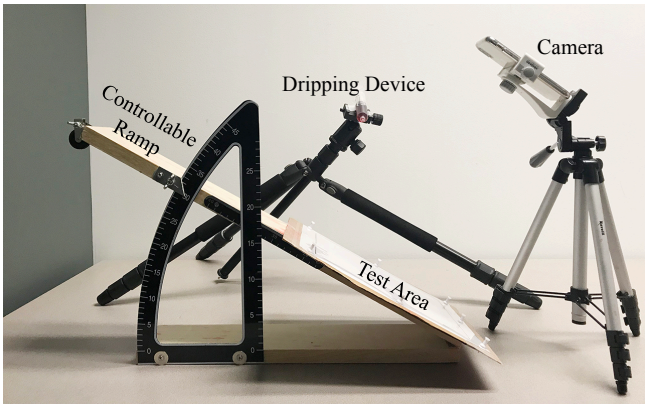


Figure 4.6: Our data capture system. We build a simple system to capture liquid flow data from the real world. This system is easy to build and convenient to use.

4.4.2 Liquid Drop Representation

Given the captured raw video data, we need to convert them into a compact representation for training purposes later in Section 4.5. Perhaps the most straightforward way is to cover the area of a liquid drop by a local window and use the interior color field as its representation. However, such an Eulerian representation suffers from two drawbacks. First, as the size of the liquid drop grows or shrinks, so does the size of the local window, which cannot be directly used for training neural networks. Possible solutions include normalizing the window size, or using a fixed window size. But they can cause other issues, such as wasted window space or confusing networks by liquid drops in different sizes. The second drawback is the dimensions of this representation. For a small liquid drop contained in a low-resolution window with 32×32 cells only, the overall dimension of the Eulerian representation vector would be 1024. This is too big for neural networks to be trained with efficiency, especially if the training set is large.

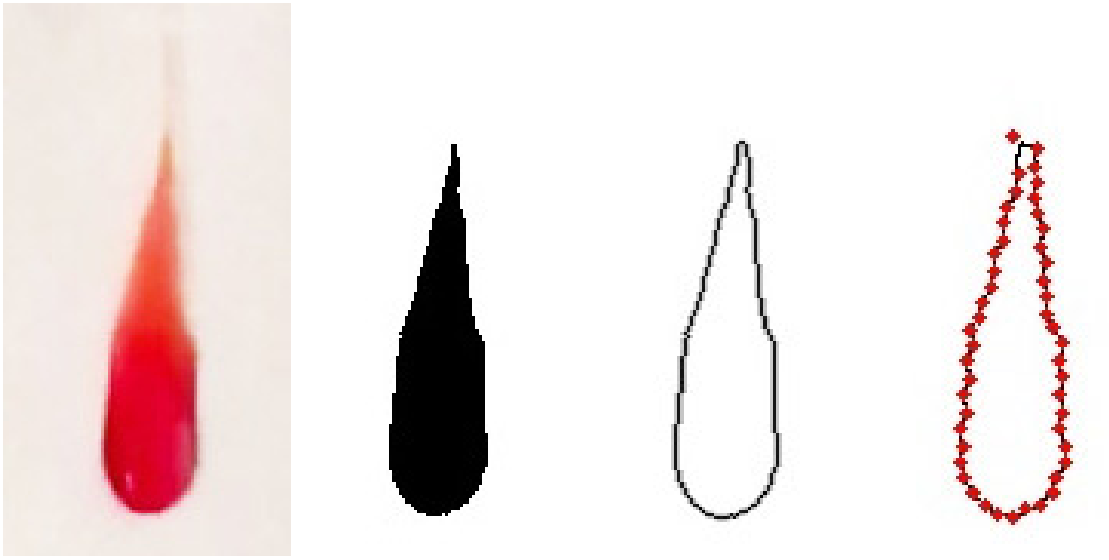
Instead, we present a Lagrangian liquid drop representation, solely based on the drop's contact front as shown in Fig. 4.7. This representation uses a fixed number of control points at the contact front, each of which stores its location and its color gradient in 2D. The point locations outline the contour of the liquid drop, and the sampled gradients determine the interior shape of the drop. Our representation is not only compact, but also effective in separating the 2D contour shape from the 3D interior shape. This allows the contour and the interior, i.e., the locations and the gradients, to be trained and predicted by two separate networks, as shown in Section 4.5.

4.4.2.1 Contour extraction

Given a video sequence obtained by our data capture device, our first job is to extract the contours of the contact fronts for our learning system.

First, we apply morphology operations to smooth out any irregularity or noise in each frame, and run Otsu's thresholding algorithm to convert the frame into a binary image, as shown in Fig. 4.7b. We then use an active contour method on the binary image to obtain a contour, represented as a densely sampled piecewise linear curve shown in Fig. 4.7c. To make those contours useful in our training system, we need both their shapes and their temporal dependencies. While a contour in one frame remains as a contour in the next frame most of the time, merging and splitting events do occur occasionally as well. Our solution is an automatic contour tracking algorithm that uses contour overlaps to determines whether a contour in one frame becomes:

- another contour in the next frame,
- or part of another contour in the next frame, due to merging,
- or two contours in the next frame, due to splitting.



(a) Image (b) Binary image (c) Dense samples (d) Control points

Figure 4.7: The contour of a liquid drop in a Lagrangian representation. After we densely sample the contour of a liquid drop in a binary image (b), we approximate it by a B-spline curve with a fixed number of control points in (d). These control points form a vector representing the liquid drop contour, used by our training system.

We do not consider the splitting of a contour into three or more contours, nor the merging of three or more contours, since they are rare given the frame rate of our video camera. The outcome of our algorithm is a set of N contour sequences, each of which is tracked over a different number of frames in our videos. The contour sequences starts when the video starts, or when it gets newly generated by merging or splitting event. The contour sequence ends when the video ends, when it leaves the field view, or when it gets terminated by merging or splitting events. Those contour sequences that end with splitting are specifically labeled for breakage prediction later in Subsection 4.5.3.

The dense samples can represent the contour well, but they are not suitable for training neural networks, since the number of samples per contour is determined by the contour

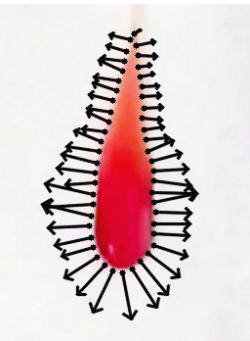


Figure 4.8: The gradients at the control points of a contact front.

size, which can vary from contour to contour, and from time to time. To solve this issue, we use a cubic B-spline curve with exactly 52 control points to approximate the sampled piecewise linear contour. The control points are ordered in a clock-wise fashion from the top. In this way, we represent each contour by a control point vector in the $\mathbb{R}^{104}$ space. Fig. 4.7e shows that the reconstruction from the control points is a good approximation to the original contour.

4.4.2.2 Gradient extraction

The contour representation given in Subsection 4.4.2.1 does not provide any information inside of the contour, i.e. the color intensity of every liquid pixel. Based on the assumption that the surface tension is sufficiently large to smooth out the mean curvature of the liquid surface, we assume that the color field is smooth as well and it can be estimated from its gradient at the contact front as discussed later in Section 4.6.1.

To predict such gradient information by our neural networks, we extract the gradient of the color field at each control point by the Sobel operator in the image space. We stack the gradient vectors together to form another large vector for training gradient prediction

networks. Fig. 4.8 shows the gradients extracted at the contact front of a liquid drop. Compared with the contour, the gradients are more sensitive to image noises and imperfections, such as highlight spots. While these issues can be addressed by applying more image editing operations, we find it is unnecessary as we expect the neural networks to learn the noises and remove their influence on predictions automatically.

4.5 Neural Networks

Given the extracted contour and gradient at the contact front of a liquid drop, our neural networks are responsible for predicting the contour and the gradient at the next time step. It also needs to predict whether the contact front breaks or not, for simulating liquid drop splitting events in Subsection 4.6.3.

Our networks are a combination of fully connected classification networks and recurrent regression networks. Our deep learning architecture is a chimeric network consisting of three individual sub-networks operating at each individual time step. To reduce the overfitting issue, we adopt the dropout approach in all of the layers except for the output layer. What it basically does is to randomly select a network node and exclude it during the training stage. In this way, the networks can be more generalized without developing co-dependencies among its neurons. See [SHK+14] for more details.

4.5.1 Contour Prediction Network

Given the contours of a contact front at the previous K time steps, we would like to predict the contour at the next time step by a subset, rather than physics models. The training process of such a neural network involves heavy temporal data dependencies, which are prone to problems of exploding and vanishing gradients. To avoid these problems, we choose

X Coordinates	Y Coordinates	Center
Input–52	Input–52	Input–2
LSTM(Lin)–260	LSTM(Lin)–260	LSTM(Lin)–260
Merge		LSTM(Lin)–260
LSTM(Lin)–260		LSTM(Lin)–260
LSTM(Lin)–260		
LSTM(Lin)–260		
LSTM(Lin)–260		
Merge		
LSTM(Lin)–260		
LSTM(Lin)–260		
Dense(Lin)–106		

Table 4.2: The structure of the shape prediction network

Gradient Magnitude
Input–104
LSTM(Lin)–250
LSTM(Lin)–250
LSTM(Lin)–250
LSTM(Lin)–250
LSTM(Lin)–250
LSTM(Lin)–250
Dense(Lin)–50

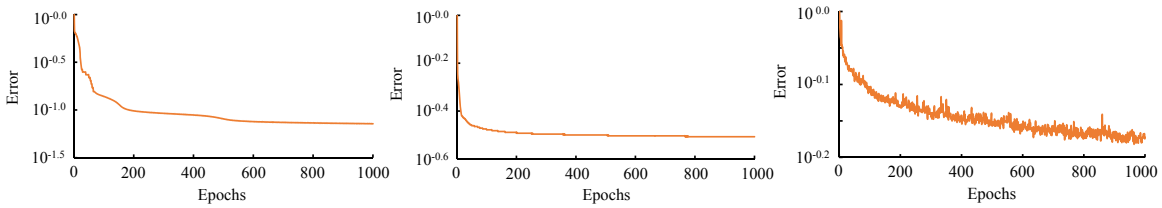
Table 4.3: The structure of the gradient prediction network

X and Y Coordinates
Input–50
Dense(ReLu)–150
Dense(ReLu)–150
Dense(ReLu)–150
Dense(ReLu)–150
Dense(ReLu)–150
Dense(ReLu)–150
Dense(Sig)–1

Table 4.4: The structure of the breakage prediction network

to use a recurrent neural network (RNN) architecture. Based on the compact representation provided in Subsection 4.4.2.1, we can then consider the sub-network training process as predicting the temporal trajectory of contour control points.

Our deep learning model uses long short term memory (LSTM) units [HS97], based on the recurrent neural network (RNN) architecture. LSTM is popularly used in natural language processing community to model complex language model dependencies. As is the common practice for most deep learning applications, we normalize (with respect to the overall dimensions of the 2D domain on which the B-spline curve is drawn) and centralize the control points. As an additional input, we track and predict the center using the previous contours. Considering the complexity and the variability of our network, its structure is shown in Table 4.2. We note that each input has its own separate processing as well as combined processing in the neural network. The intuition for this is to allow the network to organically learn motion interdependence between the two co-ordinate axes as well as cases where the major direction of the flow is just one single direction. This allows all of the attributes of this nonlinear input function to be learned both on its own and with mutual dependencies. To optimize network parameters, we use stochastic gradient descent optimization, with Nesterov acceleration [Nes83] to speed up the convergence rate. We have also tested the adaptive moment estimation (Adam) technique [KB14], but its performance is generally lower than stochastic gradient descent with Nesterov acceleration. The convergence of the training process for this contour prediction network is illustrated in Fig. 4.9a.



(a) Contour prediction network (b) Gradient prediction network (c) Breakage prediction network

Figure 4.9: Relative training error losses during the training of the three prediction networks. For contour and gradient predictions, the error is defined as the L2-norm between the ground truth and the prediction outcome. For breakage predictions, the error is defined as the accuracy of the prediction outcome.

4.5.2 Gradient Prediction Network

To fill the interior of the contact front, we need a network to predict the gradient of the color field at the contact front next. Similar to the contour prediction network in Subsection 4.5.1, this gradient prediction network uses the gradients at the last K time steps and it is formulated under a LSTM-based RNN framework. To simplify the training process, we predict the gradient magnitude at each control point only and we assume that the gradient is always perpendicular to the B-spline curve. The structure of the gradient prediction network is shown in Table 4.3. The training procedure is similar to that of the shape prediction network. The convergence behavior of the training process for the contour prediction network is illustrated in Fig. 4.9b.

It can be observed that both the gradient and the contour prediction network are both deep regression type LSTM networks differing in their inputs and outputs while other parameters such as depth, cell-type, loss functions are the same. Hence an obvious question is if they can be combined into one single network. From a fluid simulation perspective this makes sense because the shape and the volume gradient in a fluid flow functions are

essentially dependent on the same set of parameters, namely, velocity and pressure. However, practically we observe that merging the two networks can have serious repercussions on their learning capability since in that case, we will have to sample the contact front much more densely hence increasing their overall input and output dimensions. To see the rationale for this, observe that gradient prediction network needs to sample the gradient at the contact front uniformly while the B-Spline control points at the contour prediction network mostly samples densely at areas of high curvature. To avoid this discrepancy, we will need to sample the contact front much more densely hence increasing the overall dimensions of the network and in term its training and prediction time.

4.5.3 Breakage Prediction Network

Finally, we use a breakage prediction network to determine whether the contact front of a liquid drop breaks at the next time step or not. If it does break, it will cause the liquid drop to split into two new drops. While this phenomenon has a substantially complex explanation in the real world, we formulate it in our simulator as a binary classification problem based on the contact front information.

The input to this breakage prediction network is the mean-centered and normalized control points and its decision is binary: whether the contact front breaks or not. For this classification problem, we use a fully connected deep learning classifier network. The structure of the breakage prediction network is shown in Table 4.4.

An important aspect to note here is that breakage is a relatively uncommon event and the resulting class classification problem is highly unbalanced. Therefore, accuracy is not a good measure of the overall classification success. Instead, we train the network by a balanced

mixture of positives (those with breakages) and negatives (those without breakages), which are selected using an under-sampling method based on the near-miss algorithm [ZM03].

Unlike the regression problems discussed in Subsection 4.5.1 and 4.5.2, this classification problem can use either adaptive moment estimation (Adam) [KB14] or stochastic gradient descent with Nesterov acceleration, both of which have plausible convergence rates as shown in our experiment. In our simulator, we choose Adam and the convergence of the training process is shown in Fig. 4.9c.

4.6 DNN-based Simulation

Given the neural networks trained for predicting the contour, the gradient, and the breakage of a contact front, we now would like to develop a simulator that animates the time evolution of every liquid drop. To begin with, we will discuss the 3D shape reconstruction of a liquid drop from its contact front in Subsection 4.6.1. We will then study how to initialize the simulation of a liquid drop newly added to the scene in Subsection 4.6.2. In Subsection 4.6.3, we investigate the modeling and simulation of topological events, i.e., splitting and merging of liquid drops. Finally, we will present an ad hoc method for simulating drops on curved surfaces in Subsection 4.6.4.

4.6.1 Shape Reconstruction

Without external force, a liquid drop in its equilibrium state has a uniform mean curvature over its surface. With external force, a moving liquid drop no longer has a uniform mean curvature surface, but the surface should still be sufficiently smooth. Similar to [WMT07], we describe such a smooth liquid surface by a biharmonic equation of the color field, subject

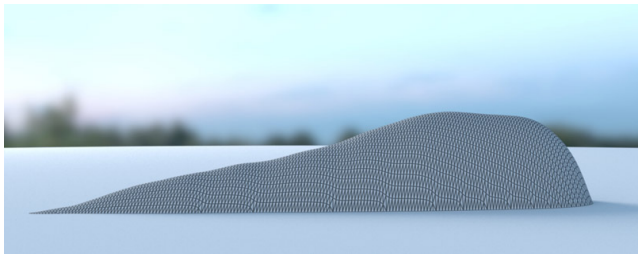


Figure 4.10: The side view of a liquid drop. Using the gradient information at the contact front, we solve a biharmonic equation to reconstruct the overall shape of this liquid drop.

to boundary conditions at the contact front:

$$\begin{cases} \nabla^4 c(\mathbf{x}) = 0, & \text{for } \mathbf{x} \in \Omega, \\ c(\mathbf{x}) = 0, & \text{for } \mathbf{x} = \partial\Omega, \\ \nabla c(\mathbf{x}) = \mathbf{g}(\mathbf{x}), & \text{for } \mathbf{x} \in \partial\Omega, \end{cases} \tag{4.12}$$

in which Ω is the liquid drop domain encircled by the contact front and $\mathbf{g}(\mathbf{x})$ is the predicted gradient at the contact front. To solve the PDE system in Equation 4.12, we discretize the biharmonic operator by finite differencing and creates a stencil covering the 2-ring neighborhood of each grid cell. To make this stencil well defined near the boundary, we use the two boundary conditions to generate the color values around the contact front. The boundary conditions are discretize in the first order. The resulting linear system is denser than the linear system of Laplace's equation, but still sparse enough for fast linear solvers. Therefore, we solve it efficiently on the CPU by MKL PARIDSO in real time. Once we obtain the color field, we convert it into a height field after proper scaling for maintaining an explicitly tracked volume, and generate a triangle mesh from the height field for visualization afterwards.

Fig. 4.10 shows the side view of a liquid drop generated by our method. Thanks to larger gradients near the head and smaller gradients near the tail, we are able to recover a plausible shape of this liquid drop. We note that first-order boundary conditions can cause

grid artifacts at both the contact front and on the interior surface. Instead of switching to use higher-order boundary conditions, we simply smooth the contact front and the interior surface as a post-processing.

4.6.2 Initialization

To initialize the simulation of a liquid drop, we request the contour of its contact front to be provided externally, either by animators or from a contour template database. One problem is that the contour prediction network needs the contours at the last K time steps, which do not exist at the very beginning of the simulation process. To solve this problem, we simply perform a *cold start*, by replicating the same contour shape $K-1$ times. This is a common practice in LSTM, when it is applied to language models and recommender systems.

Similar to shape prediction, gradient prediction also needs the last K color field gradients to begin with. Simply asking the animators to provide such gradient information is impractical. One solution is to initialize the whole 3D shape of a liquid drop and then calculate the contour and the gradient at the contact front from the shape. Alternatively, we provide a more convenient solution, which automatically chooses the proper value of each gradient vector from captured real-world data. Our idea is based on the assumption that liquid drops with similar contact front contours have similar contact front gradients. We search through a representative liquid drop database and find the one with the most similar contact front contour. We then assign its gradients to the newly initialized liquid drop. We note that this database is used for initialization only, so it is much more compact than the training database.

4.6.3 Splitting and Merging

In this subsection, we will discuss how to process two topological events in our simulation: splitting and merging.

4.6.3.1 Splitting

Once we are informed by the breakage prediction network about a splitting event, the first thing we must decide is: where does splitting happen? Assuming that splitting always happens at the “neck" of a liquid drop, we propose to find the splitting location by solving a discrete optimization problem:

$$\{i, j\} = \arg\min \left\|\mathbf{x}_i - \mathbf{x}_j\right\| - C_{i,j}, \quad \text{for } \mathbf{n}_i \cdot \mathbf{n}_j < \delta. \tag{4.13}$$

in which $\mathbf{x}_i$ and $\mathbf{x}_j$ are the 2D positions of two control points i and j, $C_{i,j}$ are their curvalinear distance along the contact front, and $\mathbf{n}_i$ and $\mathbf{n}_j$ are their normals. Intuitively, the first term in Equation 4.13 tries to reduce the Euclidean distance between the two points, while the second term tries to make them well separately along the contact front, and the constraint ensures that the two points are well opposite to each other. Since there are only 52 control points per contour, we can check every control point pair to find the one with the minimum cost, with some culling to avoid unnecessary computation.

Once we decide the splitting pair, we simply divide the B-spline curve into two and resample each new curve by their own control points. We then calculate the gradients at these new control points by linearly interpolating the gradients of the original control points. We note that this resampling and interpolation process happens not only at the current time step, but also at the last $K-1$ time steps. This ensures that the newly generated drops are ready for future prediction.

4.6.3.2 Merging

Compared with splitting, merging is relatively easy to handle in our simulator. At every time step, we first check whether the B-spline contours of two liquid drops start to overlap using dense samples. If they do, we remove the intersecting samples and approximate the remaining samples by a new B-spline curve. The gradients at newly generated control points are linear interpolated from the old ones, as in Subsection 4.6.3.1.

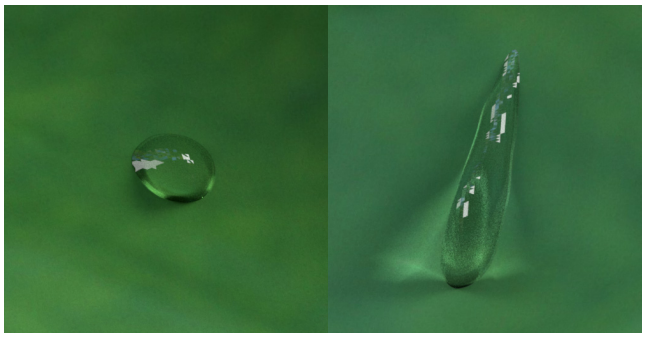
An interesting issue is how to initialize the needed data of this new drop at the previous $K-1$ time steps. It makes little sense to simply combine the information of the two merged drops, since they are irrelevant before merging happens and the prediction result would be problematic as shown in our experiment. Instead we simply treat a merged drop as a new drop and initialize it by a "cold start". A side effect of this practice is a small lag right after merging. One possible solution is to calculate the average control point displacements at the last $K-1$ time steps and apply them to the whole contour of the new drop to obtain the estimations of the last $K-1$ contours. We have not tried this idea yet.

4.6.4 Liquid Flows on Curved Surfaces

Our data capture device acquires the original data as liquid drops flowing on a planar slope with a fixed incline angle. While the resulting neural networks can effectively predict liquid behaviors in the same scenario, they become less accurate in the simulation of liquid drops on curved surfaces, or planar slopes with other incline angles.

To solve this problem, we propose to provide two adjustments to our simulator. First, we discretize the curved surface into a height field as well and place the height field of the liquid drop on it. Doing this allows the generated liquid triangle mesh to be aligned with the curved surface. Second, we scale the contact front and gradient of a liquid drop by a factor:

(a) The whole scene



(b) A static drop (c) A dynamic drop

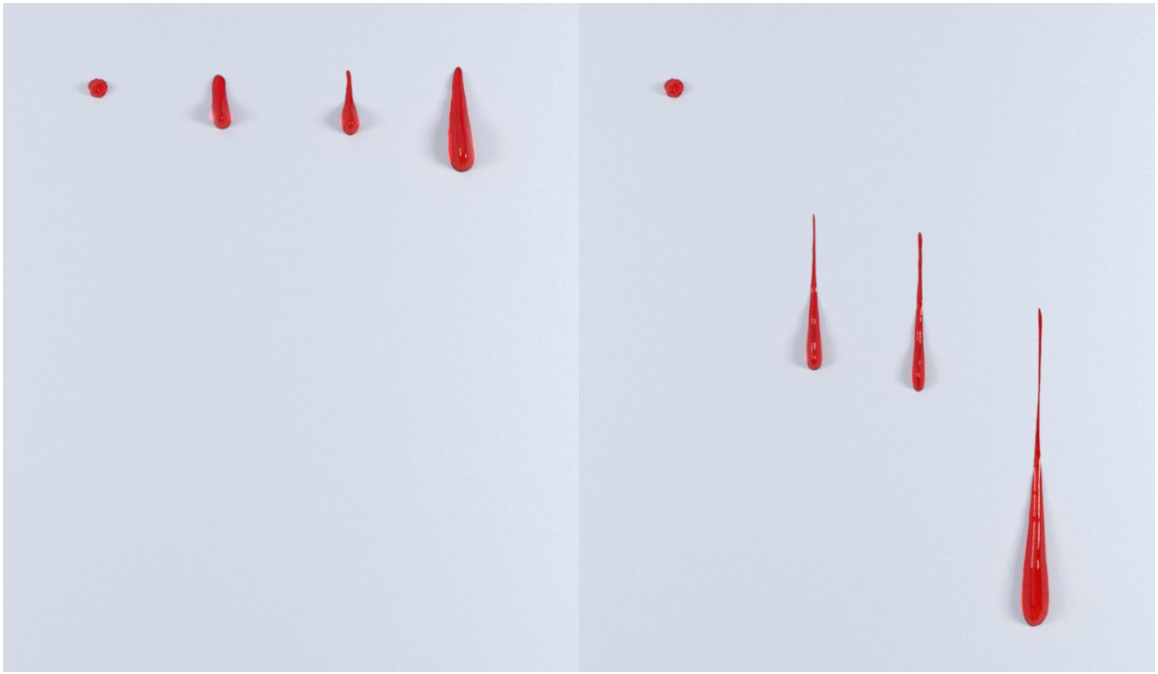
Figure 4.11: A lotus leaf example. Our simulator can simulate both static and dynamic liquid drops on a curved solid surface.

$s = (sin\theta)^{1/3}$ during prediction, in which θ is the average incline angle of the surface area contacting the liquid drop. The motive behind this practice is that larger drops move faster while smaller drops move slower, as shown in Fig. 4.12. By adjusting the size of the liquid drop during prediction, we can indirectly control its moving speed due to a varying incline angle. Fig. 4.11 shows that these techniques can be used to effectively simulate liquid flows on a curved lotus leaf.

4.7 Results

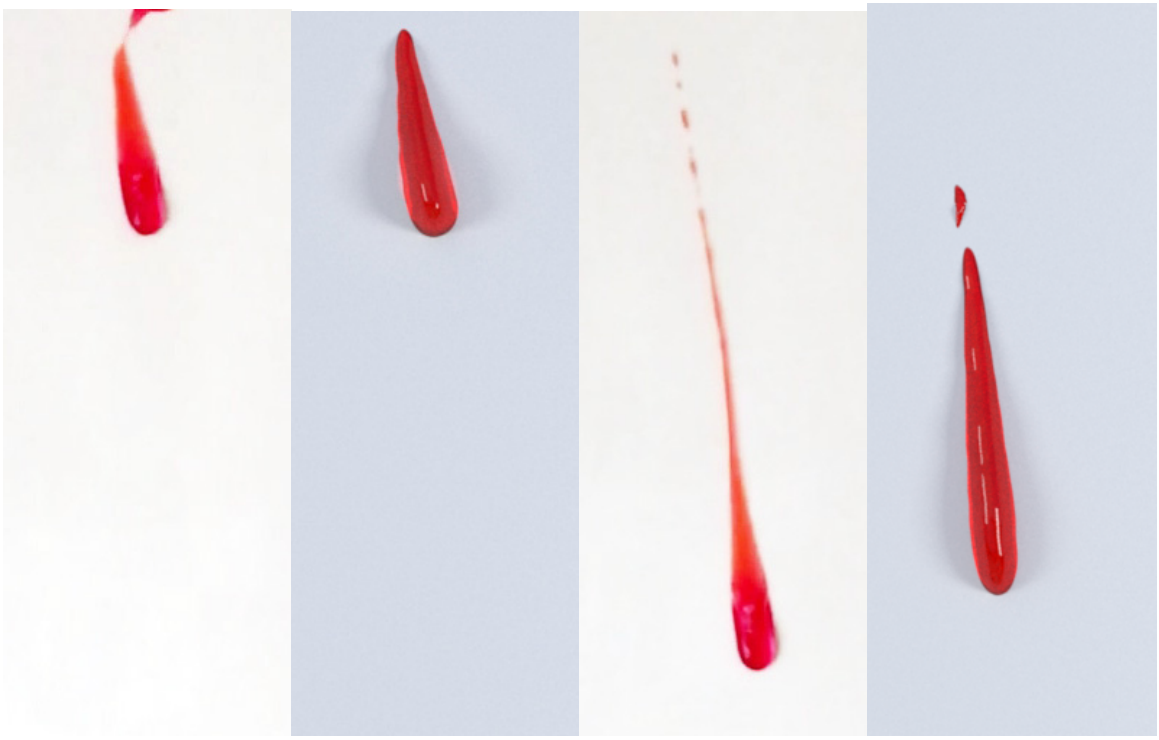
(Please watch the supplemental video for animation examples. We will release our 10GB data set including recorded videos and pre-trained models after this work is released to the public domain.) Our framework is implemented in Python 3 using Keras with the TensorFlow backend. Each network is trained for 1000 epochs with a mini-batch size of 128. The initial learning rate is 10^{-2} and learning rate decay of 10^{-6} was used. All the training procedures are done on an Intel Xeon E6-2640 Processor with a single NVIDIA P100 GPU. The evaluation is done on a desktop with Intel i7 Processor with NVIDIA 980GTX GPU. It takes approximately 0.5s per epoch for the breakage prediction networks, 204s per epoch for the shape predictor networks and 130s per epoch for the gradient predictor networks to train respectively. The total training time is approximately four days.

4.7.0.0.1 Flow comparison with the ground truth. To evaluate how accurate our simulation is, we select one liquid drop from the evaluation data set and use our simulator to generate its time evolution from the initial state, as shown in Fig. 4.13. Fig. 4.13 shows that our simulator can provide a plausible prediction for the dynamic motion of this drop. The breakage prediction subset also correctly predicts one splitting event, although the real liquid drop splits multiple times. We think there are several factors contributing to the difference



(a) Before simulation (b) After simulation

Figure 4.12: Four liquid drops in different sizes. This example demonstrates the ability of our simulator in animating flowing behaviors of liquid drops, based on their sizes.



(a) Ground truth (b) Simulated result (c) Ground truth (d) Simulated result

Figure 4.13: A comparison example with the ground truth. The ground truth here is a video clip selected from the evaluation data set, which is unused for training.

between our simulation and the ground truth video. First, we use the color image to estimate the initial drop state, which is subject to noises and errors. Second, our neural networks have their own errors. Third, the shape reconstruction process that generates the 3D shape of the liquid drop may not be accurate. Finally, each video is captured at slightly different physical conditions, which can cause the same drop to flow differently anyway.

4.7.0.0.2 Flow comparison of drops in different sizes. Fig. 4.12 compares four drops in different sizes flowing on the same solid surface. It shows that our simulator can correctly handle flowing speeds with respect to drop sizes. In particular, large drops flow

faster, while small drops remain static because of contact angle hysteresis. When two drops are in similar sizes, such as the two in the middle of Fig. 4.12a, it becomes difficult to know which one moves faster ahead of time and our simulator determines the outcome in an uncontrollable fashion.

4.8 Limitations

Due to a limited number of data categories, our neural networks are not able to predict flow behaviors for a wide range of liquid and solid material properties. Even if they do, our intrinsic data representation is unable to handle highly hydrophobic surfaces with contact angles greater than 90°. Currently, our simulator provides only an approximation to liquid flows on curved surfaces and the result would be problematic if the surface is highly uneven. The simulator has difficulty in handling liquid flows under external forces, such as wind or user interaction. The simulator becomes less accurate when it simulates long liquid streamlets and their topological events. Since the simulator is ignorant of volume preservation, it needs the volume to be explicitly tracked and maintained. Finally, our current simulator works as it is and it cannot handle liquids entering or leaving solid surfaces, such as the dripping effect.

Chapter 5: Conclusion

The primary motivation of this thesis was to emphasize the importance of real-time simulation in diverse areas of computer graphics such as designing and prototyping, haptics, fabrication and virtual surgery. In this thesis, we show how we can accelerate real-time simulations of deformable bodies and fluids emphasizing principles of systems engineering(heterogeneous CPU-GPU load balancing), Linear Algebra(Incremental Eigen Decomposition), Calculus(Newton's method) and Machine Learning(Deep Neural network). In addition to these technical insights, we also exploit problem-specific domain knowledge to further accelerate existing real-time simulation techniques. The end result is a set of interactive tools that allows artists to efficiently and elegantly model 3D-scene assets such as deformable bodies and fluids with minimum overhead, interactivity and simplified inputs. We review each contribution of the thesis individually and provide additional directions for research.

Incremental Deformation Subspace Reconstruction :In this part, we demonstrate how to perform linear modal analysis in an incremental fashion, based on the assumption that the stiffness matrix can be approximated by its low-rank low-frequency part. Compared with linear motions, nonlinear and locally rigid motions are more complex to handle. So it is difficult or computationally expensive to cover them well in the recalculated subspace.

We are actively looking for better ways to handle nonlinear and locally rigid motions by our recalculated subspace. We are also interested in studying fast ways to evaluate the quality of our subspace, so we can know when it should be replaced by the subspace calculated from scratch. How to stitch two arbitrary vertices without an expensive initial alignment step is another problem we plan to study. Finally, we would like to investigate the development of our techniques on the GPU for even faster performance.

Interactive Two-Way Shape Design of Elastic Bodies : In this part, we find that inverse quasistatic simulation can be immediately solved by Newton's method with a direct solver. Based on this observation, we develop an interactive two-way elastic shape design system, using a novel shape initialization method and a heterogeneous structure that utilizes both the CPU and the GPU.

In the future, we will explore better ways to handle inverted elements in inverse quasistatic simulation and we will improve the performance of our system using other acceleration techniques, such as multi-grid. We also would like to integrate adaptive remeshing into our system, in case the user performs significant changes to the rest shape or the quasistatic shape. Finally, we are interested in developing fast simulation systems on other platforms, such as field-programmable gate array (FPGA).

Deep Neural Network based Simulation of Small-Scale Liquid Flows on Solids :In this part, we present a novel approach to simulate small-scale liquid flows on solid surfaces by deep neural networks. Our research shows that the shape and the topological events associated with an individual liquid drop can be well predicted by trained neural networks. Combined with geometric operations, these networks can be effectively used to form a learning-based simulator for realistic animation of small-scale liquids.

Our future plan is centered around solving the limitations discussed in the relevant subsection of Chapter 4. Specifically, we plan to collect more diversified data, to see if our simulator is able to simulate flow behaviors for various liquid and solid material properties. We also plan to combine our learning-based system with existing physics-based liquid simulation techniques for simulating comprehensive liquid behaviors. Parallelization of our system for real-time liquid simulation, especially on the GPU, is another important future work we would like to investigate.

Bibliography

[AAT13] Nadir Akinci, Gizem Akinci, and Matthias Teschner. Versatile surface tension and adhesion for sph fluids. *ACM Trans. Graph. (SIGGRAPH Asia)*, 32(6):182:1–182:8, November 2013.

[AKJ08] Steven S. An, Theodore Kim, and Doug L. James. Optimizing cubature for efficient integration of subspace deformations. *ACM Trans. Graph. (SIGGRAPH Asia)*, 27(5):165:1–165:10, December 2008.

[ATW15] Ryoichi Ando, Nils Thürey, and Chris Wojtan. A dimension-reduced pressure solver for liquid simulations. *Computer Graphics Forum (Eurographics)*, 2015.

[BC05] Marc Bonnet and A. Constantinescu. Inverse problems in elasticity. *Inverse Problems*, 21(2):R1–R50, 2005.

[BJ05] Jernej Barbič and Doug L. James. Real-time subspace integration for St. Venant-Kirchhoff deformable models. *ACM Trans. Graph. (SIGGRAPH)*, 24(3):982–990, July 2005.

[BJ10] Jernej Barbič and Doug L. James. Subspace self-collision culling. *ACM Trans. Graph. (SIGGRAPH)*, 29(4):81:1–81:9, July 2010.

[BKS+12] Bernd Bickel, Peter Kaufmann, Mélina Skouras, Bernhard Thomaszewski, Derek Bradley, Thabo Beeler, Phil Jackson, Steve Marschner, Wojciech Matusik, and Markus Gross. Physical face cloning. *ACM Trans. Graph. (SIGGRAPH)*, 31(4):118:1–118:10, July 2012.

[BML+14] Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly. Projective dynamics: Fusing constraint projections for fast simulation. *ACM Trans. Graph. (SIGGRAPH)*, 33(4):154:1–154:11, July 2014.

[BNC96] Morten Bro-Nielsen and Stephane Cotin. Real-time volumetric deformable models for surgery simulation using finite elements and condensation. *Computer Graphics Forum*, 15(3):57–66, 1996.

[BPT17] Boris Bonev, Lukas Prantl, and Nils Thuerey. Pre-computed liquid spaces with generative neural networks and optical flow. *CoRR*, abs/1704.07854, 2017.

[Bra06] Matthew Brand. Fast low-rank modifications of the thin singular value decomposition. *Linear Algebra and Its Applications*, 415(1):20–30, 2006.

[BSK+16] Aric Bartle, Alla Sheffer, Vladimir G. Kim, Danny M. Kaufman, Nicholas Vining, and Floraine Berthouzoz. Physics-driven pattern adjustment for direct 3D garment editing. *ACM Trans. Graph. (SIGGRAPH)*, 35(4):50:1–50:11, July 2016.

[BW08] Javier Bonet and Richard Dixon Wood. *Nonlinear continuum mechanics for finite element analysis*. Cambridge University Press, Cambridge (UK), New York, 2008. Table des matières disponible à l'adresse suivante http://www.loc.gov/catdir/enhancements/fy0805/2007039137-t.html.

[BZ11] Jernej Barbič and Yili Zhao. Real-time large-deformation substructuring. *ACM Trans. Graph. (SIGGRAPH)*, 30(4):91:1–91:8, July 2011.

[CHMT10] Alicia Cordero, José L. Hueso, Eulalia Martínez, and Juan R. Torregrosa. New modifications of Potra-Pták's method with optimal fourth and eighth orders of convergence. *J. Comput. Appl. Math.*, 234(10):2969–2976, September 2010.

[CK05] Min Gyu Choi and Hyeong-Seok Ko. Modal warping: Real-time simulation of large rotational deformation and manipulation. *IEEE Trans. Vis. Comp. Graph.*, 11(1):91–101, January 2005.

[CKIW15] Zhili Chen, Byungmoon Kim, Daichi Ito, and Huamin Wang. Wetbrush: Gpu-based 3d painting simulation at the bristle level. *ACM Trans. Graph. (SIGGRAPH Asia)*, 34(6):200:1–200:11, October 2015.

[CMS12] D. Ciregan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3642–3649, June 2012.

[CMT+12] Stelian Coros, Sebastian Martin, Bernhard Thomaszewski, Christian Schumacher, Robert Sumner, and Markus Gross. Deformable objects alive! *ACM Trans. Graph. (SIGGRAPH)*, 31(4):69:1–69:9, July 2012.

[CT17] Mengyu Chu and Nils Thuerey. Data-driven synthesis of smoke flows with cnn-based feature descriptors. *ACM Trans. Graph.*, 36(4):69:1–69:14, July 2017.

[CWSO13] Pascal Clausen, Martin Wicke, Jonathan R. Shewchuk, and James F. O'Brien. Simulating liquids and solid-liquid interactions with lagrangian meshes. *ACM Trans. Graph.*, 32(2):17:1–17:15, April 2013.

[CZXZ14] Xiang Chen, Changxi Zheng, Weiwei Xu, and Kun Zhou. An asymptotic numerical method for inverse elastic shape design. *ACM Trans. Graph. (SIGGRAPH)*, 33(4):95:1–95:11, July 2014.

[DHB+16] Fang Da, David Hahn, Christopher Batty, Chris Wojtan, and Eitan Grinspun. Surface-only liquids. *ACM Trans. Graph. (SIGGRAPH)*, 35(4):78:1–78:12, July 2016.

[DJBDDT13] Alexandre Derouet-Jourdan, Florence Bertails-Descoubes, Gilles Daviet, and Joëlle Thollot. Inverse dynamic hair modeling with frictional contact. *ACM Trans. Graph. (SIGGRAPH Asia)*, 32(6):159:1–159:10, November 2013.

[EFFM02] Douglas Enright, Ronald Fedkiw, Joel Ferziger, and Ian Mitchell. A hybrid particle level set method for improved interface capturing. *J. Comput. Phys.*, 183(1):83–116, November 2002.

[EMF02] Douglas Enright, Stephen Marschner, and Ronald Fedkiw. Animation and rendering of complex water surfaces. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, pages 736–744, New York, NY, USA, 2002. ACM.

[Fow64] Frederick M. Fowkes, editor. *Contact Angle, Wettability, and Adhesion*, volume 43. American Chemical Society, 1964.

[FTP16] Marco Fratarcangeli, Valentina Tibaldo, and Fabio Pellacini. Vivace: a practical Gauss-Seidel method for stable soft body dynamics. *ACM Trans. Graph. (SIGGRAPH Asia)*, 35(6):214:1–214:9, November 2016.

[GKS02] Eitan Grinspun, Petr Krysl, and Peter Schröder. Charms: a simple framework for adaptive simulation. *ACM Trans. Graph. (SIGGRAPH)*, 21(3):281–290, July 2002.

[GLI16] Xiaoxiao Guo, Wei Li, and Francesco Iorio. Convolutional neural networks for steady flow approximation. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 481–490, 2016.

[Guy15] Ryan L. Guy. Smoothed particle hydrodynamics fluid simulation, 2015.

[Had06] Sunil Hadap. Oriented strands: Dynamics of stiff multi-body system. In *Proceedings of SCA*, pages 91–100, 2006.

[HLB+06] Jin Huang, Xinguo Liu, Hujun Bao, Baining Guo, and Heung-Yeung Shum. An efficient large deformation method using domain decomposition. *Comput. Graph.*, 30(6):927–935, December 2006.

[HMT11] N. Halko, P. G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Rev.*, 53(2):217–288, May 2011.

[HOT06] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554, July 2006.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[HSTP11] Klaus Hildebrandt, Christian Schulz, Christoph Von Tycowicz, and Konrad Polthier. Interactive surface modeling using modal analysis. *ACM Trans. Graph. (SIGGRAPH)*, 30(5):119:1–119:11, October 2011.

[HTC+14] Fabian Hahn, Bernhard Thomaszewski, Stelian Coros, Robert W. Sumner, Forrester Cole, Mark Meyer, Tony DeRose, and Markus Gross. Subspace clothing simulation using adaptive bases. *ACM Trans. Graph. (SIGGRAPH)*, 33(4):105:1–105:9, July 2014.

[HTZ+11] Jin Huang, Yiying Tong, Kun Zhou, Hujun Bao, and Mathieu Desbrun. Interactive shape interpolation through controllable dynamic deformation. *IEEE Trans. Vis. Comp. Graph.*, 17(7):983–992, 2011.

[HWZ+14] Xiaowei He, Huamin Wang, Fengjun Zhang, Hongan Wang, Guoping Wang, and Kun Zhou. Robust simulation of sparsely sampled thin features in sph-based free surface flows. *ACM Trans. Graph.*, 34(1):7:1–7:9, December 2014.

[HZ13] David Harmon and Denis Zorin. Subspace integration with local deformations. *ACM Trans. Graph.*, 32(4):107:1–107:10, July 2013.

[IC85] Sergio R. Idelsohn and Alberto Cardona. A reduction method for nonlinear structural dynamic analysis. *Computer Methods in Applied Mechanics and Engineering*, 49(3):253 – 279, 1985.

[ITF04] G. Irving, J. Teran, and R. Fedkiw. Invertible finite elements for robust simulation of large deformation. In *Proceedings of SCA*, pages 131–140, 2004.

[JP99] Doug L. James and Dinesh K. Pai. Artdefo: Accurate real time deformable objects. In *Proceedings of the 26th Annual Conference on Computer*

Graphics and Interactive Techniques, SIGGRAPH '99, pages 65–72, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[JS17] Richard Jones and Richard Southern. Physically-based droplet interaction. In *Proceedings of SCA*, pages 5:1–5:10, 2017.

[JSS+15] Chenfanfu Jiang, Craig Schroeder, Andrew Selle, Joseph Teran, and Alexey Stomakhin. The affine particle-in-cell method. *ACM Trans. Graph.*, 34(4):51:1–51:10, July 2015.

[KB14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[KCMF12] Tae-Yong Kim, Nuttapong Chentanez, and Matthias Müller-Fischer. Long range attachments - A method to simulate inextensible clothing in computer games. In *Proceedings of SCA*, pages 305–310, 2012.

[KD13] Theodore Kim and John Delaney. Subspace fluid re-simulation. *ACM Trans. Graph.*, 32(4):62:1–62:9, July 2013.

[KJ09] Theodore Kim and Doug L. James. Skipping steps in deformable simulation with online model reduction. *ACM Trans. Graph. (SIGGRAPH Asia)*, 28(5):123:1–123:9, December 2009.

[KJ11] Theodore Kim and Doug L. James. Physics-based character skinning using multi-domain subspace deformations. In *Proceedings of SCA*, pages 63–72, 2011.

[KLM01] P. Krysl, S. Lall, and J. E. Marsden. Dimensional model reduction in non-linear finite element dynamics of solids and structures. *International Journal for Numerical Methods in Engineering*, 51(4):479–504, 2001.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[KTY09] Ryo Kikuuwe, Hiroaki Tabuchi, and Motoji Yamamoto. An edge-based computationally efficient formulation of Saint Venant-Kirchhoff tetrahedral finite elements. *ACM Trans. Graph.*, 28(1):8:1–8:13, February 2009.

[LBK16] Tiantian Liu, Sofien Bouaziz, and Ladislav Kavan. Towards real-time simulation of hyperelastic materials. *arXiv preprint arXiv:1604.07378*, 2016.

[LBOK13] Tiantian Liu, Adam W. Bargteil, James F. O'Brien, and Ladislav Kavan. Fast simulation of mass-spring systems. *ACM Transactions on Graphics*, 32(6):209:1–7, November 2013. Proceedings of ACM SIGGRAPH Asia 2013, Hong Kong.

[LJS+15] L'ubor Ladický, SoHyeon Jeong, Barbara Solenthaler, Marc Pollefeys, and Markus Gross. Data-driven fluid simulations using regression forests. *ACM Trans. Graph. (SIGGRAPH Asia)*, 34(6):199:1–199:9, October 2015.

[Lum67] J. L. Lumley. The structure of inhomogeneous turbulent flows. In A. M. Yaglom and V. I. Tatarski, editors, *Atmospheric turbulence and radio propagation*, pages 166–178, Moscow, 1967. Nauka.

[MÖ8] Matthias Müller. Hierarchical position based dynamics. In *Proceedings of VRIPHYS*, pages 1–10, 2008.

[MCG03] Matthias Müller, David Charypar, and Markus Gross. Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '03, pages 154–159, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[MCKM14] Matthias Müller, N. Chentanez, T.Y. Kim, and M. Macklin. Strain based dynamics. In *Proceedings of SCA*, pages 21–23, 2014.

[MG04] Matthias Müller and Markus Gross. Interactive virtual materials. In *Proceedings of Graphics Interface 2004*, GI '04, pages 239–246, 2004.

[MGM15] Yajie Miao, Mohammad Gowayyed, and Florian Metze. EESEN: end-to-end speech recognition using deep RNN models and wfst-based decoding. *CoRR*, abs/1507.08240, 2015.

[MHTG05] Matthias Müller, Bruno Heidelberger, Matthias Teschner, and Markus Gross. Meshless deformations based on shape matching. *ACM Trans. Graph. (SIGGRAPH)*, 24(3):471–478, July 2005.

[Mit13] Kashmiri Lal Mittal, editor. *Advances in Contact Angle, Wettability and Adhesion, Volume 001*. Wiley Higher Education, 2013.

[Nes83] Yurii Nesterov. A method for unconstrained convex minimization problem with the rate of convergence o(1/k2). In *Soviet Mathematics Doklady*, volume 27, pages 372–376, 1983.

[OBH02] James F. O'Brien, Adam W. Bargteil, and Jessica K. Hodgins. Graphical modeling and animation of ductile fracture. *ACM Trans. Graph. (SIGGRAPH)*, 21(3):291–294, July 2002.

[OD08] Juraj Onderik and ROMAN Durikovic. Efficient neighbor search for particle-based fluids. *Journal of the Applied Mathematics, Statistics and Informatics*, 4(1), 2008.

[OH99] James F. O'Brien and Jessica K. Hodgins. Graphical modeling and animation of brittle fracture. In *Proc. of SIGGRAPH 98*, Annual Conference Series, pages 137–146, 1999.

[Ola15] Christopher Olah. Understanding lstm networks, 2015.

[PL00] Théodore Papadopoulo and Manolis I. A. Lourakis. Estimating the Jacobian of the singular value decomposition: Theory and applications. In *Proceedings of the 6th European Conference on Computer Vision-Part I*, pages 554–570, 2000.

[PMB13] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pages III–1310–III–1318. JMLR.org, 2013.

[Pro96] Xavier Provot. Deformation constraints in a mass-spring model to describe rigid cloth behavior. In *Proc. of Graphics Interface*, pages 147–154, 1996.

[PTC+15] Jesús Pérez, Bernhard Thomaszewski, Stelian Coros, Bernd Bickel, José A. Canabal, Robert Sumner, and Miguel A. Otaduy. Design and fabrication of flexible rod meshes. *ACM Trans. Graph. (SIGGRAPH)*, 34(4):138:1–138:12, July 2015.

[PW89] A. Pentland and J. Williams. Good vibrations: Modal dynamics for graphics and animation. *SIGGRAPH Comput. Graph.*, 23(3):207–214, July 1989.

[RHW88] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA, 1988.

[RJ07] Alec R. Rivers and Doug L. James. FastLSM: Fast lattice shape matching for robust real-time deformation. *ACM Trans. Graph. (SIGGRAPH)*, 26(3), July 2007.

[Sha90] Ahmed A. Shabana. *Theory of vibration. 2. , Discrete and continuous systems*. Mechanical engineering series. Springer, New York, 1990.

[She94] Jonathan R Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical report, Pittsburgh, PA, USA, 1994.

[SHK+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014.

[SLYF15] Rahul Sheth, Wenlong Lu, Yue Yu, and Ronald Fedkiw. Fully momentum-conserving reduced deformable bodies with collision, contact, articulation, and skinning. In *Proceedings of SCA*, pages 45–54, 2015.

[SNS15] M. Sundermeyer, H. Ney, and R. Schlüter. From feedforward to recurrent lstm neural networks for language modeling. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(3):517–529, March 2015.

[SPV+16] Hijung V. Shin, Christopher F. Porst, Etienne Vouga, John Ochsendorf, and Frédo Durand. Reconciling elastic and equilibrium methods for static analysis. *ACM Trans. Graph.*, 35(2):13:1–13:16, February 2016.

[Sta99] Jos Stam. Stable fluids. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '99, pages 121–128, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[STBG12] Mélina Skouras, Bernhard Thomaszewski, Bernd Bickel, and Markus Gross. Computational design of rubber balloons. *Comput. Graph. Forum*, 31(2pt4), May 2012.

[STC+13] Mélina Skouras, Bernhard Thomaszewski, Stelian Coros, Bernd Bickel, and Markus Gross. Computational design of actuated deformable characters. *ACM Trans. Graph. (SIGGRAPH)*, 32(4):82:1–82:10, July 2013.

[SYBF06] Lin Shi, Yizhou Yu, Nathan Bell, and Wei-Wen Feng. A fast multigrid algorithm for mesh deformation. In *ACM SIGGRAPH 2006 Papers*, SIGGRAPH '06, pages 1108–1117, New York, NY, USA, 2006. ACM.

[SZ14] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[TBHF03] J. Teran, S. Blemker, V. Ng Thow Hing, and R. Fedkiw. Finite volume methods for the simulation of skeletal muscle. In *Proceedings of SCA*, pages 68–74, 2003.

[Tis11] Paolo Tiso. Optimal second order reduction basis selection for nonlinear transient analysis. In *Modal Analysis Topics, Volume 3*, pages 27–39, 2011.

[TKA11] Christopher D. Twigg and Zoran Kačić-Alesić. Optimization for sag-free simulations. In *Proceedings of SCA*, pages 225–236, 2011.

[TLP06] Adrien Treuille, Andrew Lewis, and Zoran Popović. Model reduction for real-time fluids. *ACM Trans. Graph. (SIGGRAPH)*, 25(3):826–834, July 2006.

[TNGF15] Maxime Tournier, Matthieu Nesme, Benjamin Gilles, and François Faure. Stable constrained dynamics. *ACM Trans. Graph. (SIGGRAPH)*, 34(4):132:1–132:10, July 2015.

[TOK14] Yun Teng, Miguel A. Otaduy, and Theodore Kim. Simulating articulated subspace self-contact. *ACM Trans. Graph. (SIGGRAPH)*, 33(4):106:1–106:9, July 2014.

[TPBF87] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically deformable models. *SIGGRAPH Comput. Graph.*, 21(4):205–214, August 1987.

[TPS09] Bernhard Thomaszewski, Simon Pabst, and Wolfgang Strasser. Continuum-based strain limiting. *Computer Graphics Forum (Eurographics)*, 28(2):569–576, 2009.

[TSIF05] Joseph Teran, Eftychios Sifakis, Geoffrey Irving, and Ronald Fedkiw. Robust quasistatic finite elements and flesh simulation. In *Proceedings of SCA*, pages 181–190, 2005.

[TSSP16] J. Tompson, K. Schlachter, P. Sprechmann, and K. Perlin. Accelerating Eulerian Fluid Simulation With Convolutional Networks. *ArXiv e-prints*, July 2016.

[TWGT10] Nils Thürey, Chris Wojtan, Markus Gross, and Greg Turk. A multiscale approach to mesh-based surface tension flows. *ACM Trans. Graph. (SIGGRAPH)*, 29(4):48:1–48:10, July 2010.

[UHT17] Kiwon Um, Xiangyu Hu, and Nils Thuerey. Liquid splash modeling with neural networks. *CoRR*, abs/1704.04456, 2017.

[vTSSH13] Christoph von Tycowicz, Christian Schulz, Hans-Peter Seidel, and Klaus Hildebrandt. An efficient construction of reduced deformable objects. *ACM Trans. Graph. (SIGGRAPH)*, 32(6):213:1–213:10, November 2013.

[Wan15] Huamin Wang. A Chebyshev semi-iterative approach for accelerating projective and position-based dynamics. *ACM Trans. Graph. (SIGGRAPH Asia)*, 34(6):246:1–246:9, October 2015.

[WMT05] Huamin Wang, Peter J. Mucha, and Greg Turk. Water drops on surfaces. *ACM Trans. Graph. (SIGGRAPH)*, 24(3):921–929, July 2005.

[WMT07] Huamin Wang, Gavin Miller, and Greg Turk. Solving general shallow wave equations on surfaces. In *Proceedings of SCA*, pages 229–238. Eurographics Association, 2007.

[WMW15] Xiaofeng Wu, Rajaditya Mukherjee, and Huamin Wang. A unified approach for subspace simulation of deformable bodies in multiple domains. *ACM Trans. Graph. (SIGGRAPH Asia)*, 34(6):241:1–241:9, October 2015.

[WOR10] Huamin Wang, James O'Brien, and Ravi Ramamoorthi. Multi-resolution isotropic strain limiting. *ACM Trans. Graph. (SIGGRAPH Asia)*, 29(6):156:1–156:10, December 2010.

[WST09] Martin Wicke, Matt Stanton, and Adrien Treuille. Modular bases for fluid dynamics. *ACM Trans. Graph. (SIGGRAPH)*, 28(3):39:1–39:8, July 2009.

[WWY+15] Bin Wang, Longhua Wu, KangKang Yin, Uri Ascher, Libin Liu, and Hui Huang. Deformation capture and modeling of soft objects. *ACM Trans. Graph. (SIGGRAPH)*, 34(4):94:1–94:12, July 2015.

[WY16] Huamin Wang and Yin Yang. Descent methods for elastic body simulation on the GPU. *ACM Trans. Graph. (SIGGRAPH Asia)*, 35(6):212:1–212:10, November 2016.

[XLCB15] Hongyi Xu, Yijing Li, Yong Chen, and Jernej Barbivč. Interactive material design using model reduction. *ACM Trans. Graph. (SIGGRAPH)*, 34(2):18:1–18:14, March 2015.

[XSZB15] Hongyi Xu, Funshing Sin, Yufeng Zhu, and Jernej Barbič. Nonlinear material design using principal stretches. *ACM Trans. Graph.*, 34(4):75:1–75:11, July 2015.

[YLX+15] Yin Yang, Dingzeyu Li, Weiwei Xu, Yuan Tian, and Changxi Zheng. Expediting precomputation for reduced deformable simulation. *ACM Trans. Graph. (SIGGRAPH Asia)*, 34(6):243:1–243:13, October 2015.

[YXG+13] Yin Yang, Weiwei Xu, Xiaohu Guo, Kun Zhou, and Baining Guo. Boundary-aware multidomain subspace deformation. *IEEE Tran. Vis. Comp. Graph.*, 19(10):1633–1645, Oct 2013.

[YYX16] Cheng Yang, Xubo Yang, and Xiangyun Xiao. Data-driven projection method in fluid simulation. *Comput. Animat. Virtual Worlds*, 27(3-4):415–424, May 2016.

[ZB13] Yili Zhao and Jernej Barbič. Interactive authoring of simulation-ready plants. *ACM Trans. Graph.*, 32(4):84:1–84:12, July 2013.

[ZLQF15] Bo Zhu, Minjae Lee, Ed Quigley, and Ronald Fedkiw. Codimensional non-newtonian fluids. *ACM Trans. Graph. (SIGGRAPH)*, 34(4):115:1–115:9, July 2015.

[ZM03] J. Zhang and I. Mani. KNN Approach to Unbalanced Data Distributions: A Case Study Involving Information Extraction. In *Proceedings of the ICML'2003 Workshop on Learning from Imbalanced Datasets*, 2003.

[ZSTB10] Yongning Zhu, Eftychios Sifakis, Joseph Teran, and Achi Brandt. An efficient multigrid method for the simulation of high-resolution elastic solids. *ACM Trans. Graph.*, 29(2):16:1–16:18, April 2010.

[ZWW+12] Yizhong Zhang, Huamin Wang, Shuai Wang, Yiying Tong, and Kun Zhou. A deformable surface model for real-time water drop animation. *IEEE Transactions on Visualization and Computer Graphics*, 18(8):1281–1289, 2012.