University of New Mexico
UNM Digital Repository

Electrical and Computer Engineering ETDs | Engineering ETDs

Spring 4-15-2020

Advancing Elastic Solid Dynamics in Computer Graphics

Ran Luo
University of New Mexico - Main Campus

Follow this and additional works at: https://digitalrepository.unm.edu/ece_etds

Part of the Electrical and Computer Engineering Commons

Recommended Citation

Luo, Ran. "Advancing Elastic Solid Dynamics in Computer Graphics." (2020).
https://digitalrepository.unm.edu/ece_etds/522

This Dissertation is brought to you for free and open access by the Engineering ETDs at UNM Digital Repository. It has been accepted for inclusion in Electrical and Computer Engineering ETDs by an authorized administrator of UNM Digital Repository. For more information, please contact disc@unm.edu.

Ran Luo

Candidate

Electrical and Computer Engineering

Department

This dissertation is approved, and it is acceptable in quality and form for publication:

Approved by the Dissertation Committee:

Yin Yang , Chairperson

Marios Pattichis

Rafael Fierro

Meeko Oishi

Shuang Luan

Advancing Elastic Solid Dynamics in Computer Graphics

by

Ran Luo

DISSERTATION

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Doctor of Philosophy
Engineering

The University of New Mexico

Albuquerque, New Mexico

May, 2020

DEDICATION

To my family, for their support they're giving me for graduation.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Yin Yang, for his patient and advice during my Ph.D. research and study at the University of New Mexico. Without his continues selfless help, this dissertation may not exist.

I would also like to thank all of my committee members, Dr. Marios Pattichis, Dr. Rafael Fierro, Dr.Shuang Luan, and Dr.Meeko Oishi, for their kindly and expert suggestions.

Special thanks to my coworkers, Garcia Steven, Yuming Zhang, and Lan Lei, for their helping and every deadline we working together.

Finally, I would like to thank my family, Junhao Chen, Evan Luo, Max Luo, and my parents, for all their support and understanding, to allow me to focus on research and study.

Advancing Elastic Solid Dynamics in Computer Graphics

by

Ran Luo

Ph.D., Engineering, University of New Mexico, 2020

ABSTRACT

This dissertation proposes novel algorithms and applications and provides a real-time and easy-to-use simulator for realistic animation of the 3D solid model. The Finite Element Method (FEM) is a popular tool in the community because of its accurate result, however, the FEM is computationally expensive to handle a large number of DOFs. We present novel techniques to combine linear and nonlinear elasticity with model reduction to provide fast and realistic animation. On the other hand, one of the most important computation tasks of solid simulation is to evaluate the gradient vector and Hessian matrix of elastic energy function. We present a numerical routine to simplify the implementation of solid simulation with the complex-step finite difference (CSFD) that avoids subtractive cancellation. The complexity of nonlinearity is also an obstacle, and we provide a framework called NNWarp to combine the linear elasticity and neural network-based warping method to avoid expensive nonlinear optimization. We also propose an acoustic-VR system as the application. The system can convert acoustic signals of human language to realistic 3D tongue animation in real-time. The Deep Neural Networks (DNN) helps to convert the input speaking voice to positions of pre-defined EMA sensors. Then, a novel reduced physics-based solid simulator, introduced in previous, is used to synthesis the tongue animation.

Contents

List of Figures **x**

List of Tables **xxii**

1 Introduction **1**

2 Interactive Design and Simulation of Tubular Structure **6**

2.1 Introduction 6

2.2 Related Work 9

2.3 System Overview 11

2.4 Geometric Design of Free-form Tubes 12

2.5 Formulation of General Shell Element 15

2.6 Model Reduction of Tubular structure 20

2.6.1 Constraint Subspace - the Primary Subspace 22

2.6.2 Multiplier-free Component Coupling 23

2.6.3 Residual Subspace - the Secondary Subspace 25

2.7 Experimental Results 29

2.7.1 Hardware and Software Platform 29

2.7.2 Four-node Element vs. nine-node element 29

2.7.3 User interface and Implementation Details 30

2.7.4 Time Performance 32

3 Nonlinear Elasticity with Overlapped Domain Decomposition 36

3.1 Introduction 36

3.2 Related Work 39

3.3 Quadratic DOFs 42

3.4 Deformable Quadratic Model 43

3.5 Physics-based Elastic Weighting 46

3.6 Adaptability and Extensibility 56

3.7 Experimental Results 60

4 Complex Step Finite Difference for Solid Dynamics 69

4.1 Introduction 69

4.2 Related Work 72

4.3 Background 76

4.4 Complex-Step Finite Difference 78

4.5 CSFD Acceleration 81

4.5.1 Accelerate CSFD of a Single Elementary Function 82

4.5.2 Accelerate CSFD of Composite Binary Operators 84

4.5.3 Accelerate CSFD of Composite Unary Operators 87

4.6 Multicomplex-Step Perturbation 89

4.7 Tensor Function 94

4.8 Experimental Results 96

4.8.1 Application I: Accurate Nonlinear Optimization 100

4.8.2 Application II: Intuitive Hyperelastic Simulation 102

4.8.3 Application III: Expressive Model Reduction 106

4.8.4 Application IV: Convenient Inverse Design 107

5 Neural Network-based Nonlinear Deformation 111

5.1 Introduction 111

5.2 Related Work 114

5.3 Contextual Feature Vector 118

5.3.1 Deformable model: a quick review 118

5.3.2 Linear-nonlinear correspondence 121

5.3.3 Discriminative feature 123

5.4 Incorporate Complex Shapes 131

5.5 Network Structure and Training 133

5.6 Other Experimental Results 138

6 Application: Real-time Speech-driven Tongue System 144

6.1 Introduction 144

6.2 Related Work 147

6.3 System Overview 150

6.4 Data Acquisition 151

6.5 DNN-based Speech Inversion 154

6.6 Real-time Deformable Simulation of Tongue 157

6.7 Domains' Weight Coefficients 161

6.8 Preserving Volume within the Subspace 164

6.9 Experimental Result 168

7 Conclusion and Limitation 173

Appendices 177

A Proving Eq. 2.28 178

B Elementary Complex Promotion 180

References 182

List of Figures

2.1 Two furniture designs using supporting tubes. 7

2.2 An overview of the proposed design-simulation system. 11

2.3 The boundary conditions and sweep trajectory of a free-form tube. The designed free-form tube is on the left and its shape space of cross sections is shown on the right. 12

2.4 Boundary constrained swept surfaces using ODE-based techniques without (a) and with (b) sweep trajectory control. 15

2.5 The boundary conditions and sweep trajectory of a free-form tube. The designed free-form tube is on the left and its shape space of cross sections is shown on the right. 15

2.6 Extracting orthogonal basis vectors $\bar{\mathbf{r}}$, $\bar{\mathbf{s}}$ and $\bar{\mathbf{t}}$. 20

2.7 The shapes of five constraint modes associated with a boundary node (highlighted as blue node). Other restrained boundary DOFs are marked as red nodes. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.) 21

2.8 An illustrative example showing the coupling of tubular components η and ζ assuming that an appropriate boundary condition has been specified at the highlighted nodes. 23

2.9 The shapes of five residual modes associated with an exciting node (green). 26

2.10 Overall computational procedures of the two-level subspace simulation method. 26

2.11 Shear locking using linear four-node element (left) which eliminates the bending deflection while quadratic nine-node element (right) does not have such artifact. 30

2.12 The user interface of the proposed system. 31

2.13 Snapshots of using the proposed design-simulation system. 33

2.14 The 3D printed lamp stand fails and the failure location matches the area where high stress distribution is observed. 34

3.1 Overlapping quadratic domains make the simulation robust even under large deformations. The one-inch-tall bunny model is forced to pass a funnel whose inner diameter is only 0.3 inch. Our method yields plausible animations (the red bunny) at an interactive rate comparable to the fullspace simulation (the blue bunny). Most existing non-overlapping multi-domain simulators (i.e. [1,2]) fail in this challenging test. Indeed, our simulator remains stable even when the funnel's diameter is reduced to 0.2 inch (Fig. 3.16). Please refer to the supplementary video and executables for more details. 37

3.2 Pros and cons of existing single- and multi-domain reduction techniques for nonlinear deformable models. 38

3.3 We apply pure bending moments along the neutral axis of the beam model. The bending quality is measured using the maximum shearing angle along the neutral axis. Our method yields a much smaller shearing locking artifact than other competitors including affine DOFs [3], and rigid DOFs [4]. The ground truth is the result using fullspace quadratic tetrahedral elements. Under such challenging bending test, even fullspace simulation using linear elements will fail. 43

3.4 Decompose a deformable body into four domains. 47

3.5 The Voronoi segmentation and corresponding domain decomposition of the bunny model. 48

3.6 (a) Anchoring boundary nodes completely results in weight damping. (b) Principal projection yields smoother weight functions. 51

3.7 Computing weight function along different directions other than the principal direction leads to locking artifacts. (a) shows the rest shape of the beam, whose principal direction is along the y axis. We show its equilibrium shapes under the gravity using weight functions calculated with directions further and further away from the principal direction. The resulting shapes are aggregated for the comparison. The ground truth is also given in (c). In (d) we plot the relative shape difference for different directions. 52

3.8 Comparative simulation of a winding snake. 55

3.9 Left: the initial weight distribution w of an existing domain seeded at S. Mid: the weighting function of a newly inserted domain w_a. Right: the updated weighting function w' can be fast obtained as the linear combination of w and w_a. 57

3.10 Adding new domain produces natural denting effects on the brick. . 58

3.11 Use reduced IFE simulation to imitate the inflation of a hot-air balloon. In this example, we simply use Cubature points as restorative elements. Three domains are defined. 59

3.12 Compare different weighting algorithms with a cylinder beam of heterogenous materials: the Young's modulus of the beam varies along its neutral axis. Two quadratic domains are set. The weight distributions from different algorithms are also plotted. 59

3.13 Shape functions of a simple 1D element. 63

3.14 BBW may induce locking artifacts. 63

3.15 Simulate a swinging cactus using deformation substructuring [1], unified domain decomposition [2], our method, and the fullspace solver. 64

3.16 Drag the bunny through a thin funnel (available as the supplementary executable too). 66

3.17 Our solver remains stable under severe geometry constraints. 67

3.18 Adding new domains according to the contacts from a moving spike board greatly enriches detailed local deformation. 67

3.19 Runtime domain adaption does not only yield better denting effects on the bunny, but also enriches local deformations at the lollipop. . 68

4.1 We present an accelerated complex-step finite difference algorithm, which efficiently computes highly accurate numerical derivative. This method can be coupled with the Cauchy-Riemann formula to allow us to fully exploit existing (real-valued) linear algebra libraries to evaluate derivatives of tensor-valued functions. This figure reports an example of designing vibration frequency of a bridge model (21,414 elements) by changing its geometry. The target frequency is visualized on a rectangular beam. Given an external force field, the bridge oscillates under the same frequency as the beam model does. 70

4.2 We use FD, CD, and CSFD to calculate the first-order derivative of $f(x) = e^x/(x^4 + x^2 + 1)$ at $x = 4$. The resulting numerical derivative is compared with the analytic derivative, and the relative error is plotted against the size of the perturbation, ranging from 2^{-2} to 2^{-127}. 79

4.3 Our fast CSFD implementation has good numerical stability and accuracy. The relative error converges as quickly as the regular CSFD and remains at the order of machine epsilon after h is sufficiently small. 83

4.4 The procedure of evaluating a chain of multiplications (and divisions) can be visualized with a binary tree. The leaf nodes can be concisely encoded by a binary number. As a result, we can discard higher-order infinitesimals with two or more 1s (e.g. 011) at the bottom level. . . 86

4.5 The performance of MCSFD approximation. We use the same test function of $f(x) = e^x/(x^4 + x^2 + 1)$ as in Fig. 4.2 and calculate its second-order derivative at $x = 4$. The relative error w.r.t to value of analytic derivative is plotted against the size of the perturbation, ranging from 2^{-2} to 2^{-63}. 93

4.6 Cauchy-Riemann formula allows us to use existing linear algebra libraries to compute high-order numerical derivative without referring to an explicit complex promotion. In this example, we compute the first- and second-order derivative of 3×3 matrix inverse. CR-form based CSFD and MCSFD still have excellent accuracy compared with regular finite difference. 96

4.7 The Armadillo model falls quickly and hits a glassy rod. Due to the sharp collision, gradient descent method [5] yields artifact because the residual is not sufficiently reduced. Regular finite difference method crashes instantly. Newton's method with MCSFD-based Hessian yields the same result as using the analytic Newton. Newton-PCG with CSFD-based directional derivative also has the same result. . . 97

4.8 We simulate a Neo-Hookean Armadillo model using Newton's method. The Armadillo has $69,074$ elements. The gradient and Hessian of the target function f (i.e. Eq. (4.35)) is approximated using numerical CSFD/MCSFD. The result is identical to the one computed using analytic gradient and Hessian. 98

4.9 CSFD/MCSFD allows the user to easily simulate all kinds of hyper-elastic materials. The figure reports the material behaviors under standard bending, compressing, stretching, and twisting tests of a box model with $14,678$ elements. From left to right, each column gives the result of Arruda–Boyce, Fung, Mooney-Rivlin, Neo-Hookean, Ogden, Polynomial, invertible StVK [6] (for improved stability), and Yeoh materials. 102

4.10 Timing information of CSFD/MCSFD derivative in simulating various hyperelastic materials. **Opt.** is the optimized CSFD/MCSFD computation time. **Img. only** is the time without computing the real part of the promoted energy functions. **B.F.** is the computation time using a brute-force CSFD/MCSFD implementation. 103

4.11 Our new material (Eq. (4.38)) with a more aggressive volume penalty term is able to better preserve the volume of this jelly box during the compression than the stable Neo-Hookean material [7]. 104

4.12 We design a new volume penalty term of $\log^2(1-4(J-1)^2)$, which yields much bigger internal forces when $J=|\mathbf{F}|$ deviates from 1 than the regular Neo-Hookean volume penalty of $(J-1)^2$ does. 104

4.13 Example-based hyperelastic energy can also be easily handled with CSFD/MCSFD. We make the energy a the function of bending angle so that a smiling face appears when the box bends to left and a sad face appears when the box bends to right. This box model has $14,678$ elements. 105

4.14 Complicated energy formulation as Eq. (4.39) could hide singularities that are unfriendly for AD. CSFD/MCSFD can tackle this issue robustly. 106

4.15 Real-time simulation of six falling dinosaur models using modal derivative (30 modes for each dinosaur). The first-order derivative modes are computed using CSFD, and we use Fung, Mooney-Rivlin, Ogden, Yeoh, Arruda–Boyce and Polynomial materials for each dinosaur. . . 108

4.16 MCSFD allows us to compute higher-order modal derivatives that capture extreme bending effects of the dinosaur model (using 30 second-order derivative modes). Interestingly, the hyperelastic energy of Eq. (4.38), because of its strong resistance to volume change, cannot be bent as hard as other materials under the same external force. . . 109

4.17 We develop a system with an intuitive interface for the linear frequency design (left). The error reduces quickly along Newton iterations, with the Hessian accurately computed from MCSFD. (right) 109

5.1 NNWarp is a data-driven neural network based nonlinear deformable simulator. By learning from full FEM simulation poses, it yields more accurate results than existing warping methods. We design three compact contextual features making the network training highly reusable. In this example, the maple bonsai model consists of $255,552$ elements, and is decomposed into $1,771$ domains. A single net trained using a regular beam handles local dynamics for all the domains. High-quality animations with well-preserved local high-frequency deformations are produced at a near-interactive rate (5 FPS) without using model reduction. 112

5.2 Different motion trajectories lead to different equilibrium shapes even under the same external force. 121

5.3 Only using kinematic feature as the input of the network yields noticeable jittery artifacts. A node, because of its kinematic feature is not discriminative, could be influenced by many irrelevant instances in the training date. Large discrepancies among these mismatched nodes induce high-frequency variations of the NNWarp. Incorporating geodesic feature effectively eliminates this artifact. 123

5.4 Volume expansion artifact remains even with the geodesic feature added. This is because the nodes with similar geodesic value may have different internal tractions. We use the potential feature to sort the training data to avoid this issue. 125

5.5 We test the generality of the designed features on a variety of shapes. The training data are generated using the standard rectangular beam (highlighted by a red box). The resulting DNN successfully handles many beam-like models but with distinctively different shapes. The distributions of three features are also plotted. 126

5.6 In order to make NNWarp re-usable for various deformable bodies, we use the digression as our third discriminative feature. With this feature included, the neural network is able to handle an irregular beam based on the training set generated using a standard rectangular beam model. 127

5.7 NNWarp works well under circular force field too. In this case, the digression feature is set as -1 for all the nodes on the mesh. 129

5.8 When the shape becomes more concave, the network trained using a rectangular beam produces artifacts. 130

5.9 Building domain graph for the domain decomposed model is an easy and effective way to identify shapes with similar concavity. 130

5.10 While a simple rectangular beam is not able to handle highly concave shapes, by referring to the domain graph we can train a network using a T-shape beam and the resulting network can be used to warp a wide range of concave beams whose domain graphs are isomorphic to the T-shape beam. 131

5.11 Rotation invariance allows us to further compress the input kinematic feature. 134

5.12 The distribution of three kinematic features of $1,000$ entries after the alignment. $\angle\widetilde{\mathbf{u}}\mathbf{w}$ denotes the angle between aligned vectors $\widetilde{\mathbf{u}}$ and $\mathbf{w}$. 135

5.13 A side-by-side comparison shows a clear advantage of NNWarp over the existing warping techniques. Its data-driven nature makes the result almost identical to the ground truth while the simulation is as fast as the linear elasticity. The training still uses the rectangular beam model. 136

5.14 Training error vs epoch of the Adam solver. 137

5.15 The deformable motion trajectory (at the nose tip of the wolf head) generated using NNWarp well matches the ground truth under different time step sizes. The vibration frequency resembles the ground truth as well. We use the Newmark integrator in this example. . . . 139

5.16 Substructuring allows us to re-use the training data of a regular shape to handle complex deformable bodies. The Armadillo, dinosaur and dragon models use the StVK, co-rotation and Neo-Hookean materials respectively. They use the networks trained with the rectangular beam. 141

5.17 We can simulate free-floating deformable bodies by creating an artificial boundary condition to constrain the element near the mass center. 142

6.1 A snapshot of the interface of the proposed system. 145

6.2 An overview of our acoustic-VR system. 150

6.3 Experiment setting for data acquisition. (a) the `SIEMENS` MRI system. (b) `NDI Wave` EMA system. (c) Gathering the articulary movement data using EMA. (d) Placing EMA coils on the tongue. 152

6.4 VCV syllables used for MRI-based tongue shape retrieval. 153

6.5 (a) EMA sensors' placement and (b) the aligned MRI-CBCT-EMA volume. 153

6.6 The domain partition of the input tongue mesh as well as the weight distribution for each domain. 161

6.7 **Left:** In this illustrative example, we compare a simple bending simulation of a standard load-end cantilever beam (with three domains) using (a) our method, (b) weighting computed with a completely fixed boundary condition, (c) weighting computed along the direction perpendicular to the principle direction, and (d) using harmonic coordinates. **Right:** We pick key frames (grey blocks) by checking the finite difference acceleration magnitude of each EMA sensor and compute the corresponding pressure field. Selected pressure vectors and the corresponding tongue shapes are visualized as well using the red-white color map. 165

6.8 Collision detection is limited at few selected collision points. 167

6.9 Applying the volume preserving constraint yields more natural tongue shapes, and our subspace volume preserving is able to effectively suppress volume change during tongue's deformation. The first row of snapshots is the shapes without volume preserving. The second and the third rows are the results using fullspace method and our method. It can be seen that our method is able to produce almost identical results compared to the fullspace volume correction. With our method, the volume change during the tongue simulation is always less than 2%. 169

6.10 Side-by-side comparisons between simulated tongue shapes (textured) and real-world shapes extracted from MRI-CBCT fused images (in cyan) 170

6.11 More snapshots of the tongue during speech production. The input acoustic signal waves are also provided. 170

List of Tables

2.1 Time performance of our method, full-space simulator as well as subspace simulator using Lagrange multiplier method. #Ele.: the number of elements; #Nodes: the number of free nodes; #DOFs: the number of full-space DOFs: #S(L)DOFs the size of the simulator using constraint mode and Lagrange multiplier method; SDOFs: the size of the simulator using the propose multiplier-free coupling method; FT: time used to solve the system in full-space; S(L)T: time used to solve the multiplier-based subspace system; ST: time for our method; #Com: the number of the tube components. 34

3.1 Model statistics and simulation benchmarks. **#Ele**: the number of elements on the simulation mesh; **#D**: the number of initialized quadratic domains; **#Cub**: the number of cubature elements in total; **Pre**: pre-computation time in minutes (with multi-threading enabled) and the accuracy of the Cubature training; **Sim**: average FPS for simulating the deformable bodies (collision handling not included). **FPS**: over all FPS including collision/self-collision handling and OpenGL rendering. 61

4.1 Time statistics of using the optimized CSFD (i.e. Eqs. (4.8) and (4.10)) and the naïve CSFD implementation (Eq. (B.4)) of the exponential function $f(x) = x^{1/m}$ for 100 million times. The computation time using analytic derivative is also reported for the reference. Our CSFD simplification is over 200× faster than the naïve implementation. In this example, it is even faster than using the analytical derivative. The relative error is at the order of the machine epsilon (10^{-16}). 83

4.2 Time statistics of computing first- (**1st**), second- (**2nd**), and third-order (**3rd**) derivatives for 1 million times of the function: $f(x) = e^x/(x^4 + x^2 + 1)$ using CSFD/MCSFD and some popular AD packages. 97

4.3 Computing the internal force and tangent stiffness matrix for $10k$ linear tetrahedral elements of StVK and Neo-Hookean materials. Similar to Tab. 4.2, our accelerated CSFD/MCSFD is much faster than AD packages. 99

5.1 Time performance of the examples reported in the chapter. **Factorization** is the time needed to pre-factorize the system matrix of the linear elasticity. We use `SimplicialLLT` solver shipped with `Eigen`. During the simulation, we only need to solve the system once. **FPS** reports both CPU and GPU performance. 140

Chapter 1

Introduction

Solid dynamics is a system that describes and simulate 3D solid objects. The research in this field helps scientists analyze and test their models for better understanding and helping our world. In computer graphics, solid modeling is a popular tool for visualization and synthesis of realistic animation. Many solid objects in real life can be described and simulated very well. For example, cloth, tree, building structure and human body tissue that can be approximated by linear or nonlinear solid models. Therefore, the concept of solid modeling is widely used in industrial design, games and medical scenes, and that makes the solid modeling becomes an important research topic. However, building a fast, accurate, and easy-to-use real-time interactive solid simulator is still a very challenging problem. This dissertation will present a set of algorithms to develop fast 3D solid simulation based on the linear and nonlinear elastic model, and we will also discuss how to simplify the implementation and present the application.

In this dissertation, we use elastic deformation strain-stress relationship to simulate the solid objects, which based on the partial differential equations of the continuum mechanics over the objects. Due to the lack of analytic solutions of complex geometry

shape, we use the Finite Element Method (FEM) to numerically solve the equations. The 3D domain of the solid object will be discretized with a set of elements (e.g., tetrahedron elements) and the equations will be evaluated piece wisely. The simulation results are more accurate if the discretization process generates higher resolution volumetric mesh. However, the solid simulation could be extremely computationally expensive due to a large number of degrees of freedom (DOFs) of the system. Furthermore, the partial differential equations can be nonlinear in solid simulations, this slows the computation and makes implementation difficult. We will discuss how to achieve a faster and easier implementation simulation in this dissertation.

In Chapter 2 and Chapter 3, we will combine model reduction with linear and nonlinear elastic simulation together to achieve a faster deformable model. The model reduction technique helps us to find a smaller set of unknowns that can also be used to prescribe or approximate the original system. For a 3D deformable object, the original system size will be $3n$, 3 DOFs per vertex of 3D volumtric mesh, which will be a very large-size system to solve. Model reduction projects the $3n$ system into a much smaller reduced system with a properly chosen subspace with the equation:

$$\boldsymbol{u} = \boldsymbol{\Phi}\boldsymbol{q}, \tag{1.1}$$

where $\boldsymbol{u} \in \mathbb{R}^{\boldsymbol{3n}}$ is the displacement vector of 3D mesh, $\boldsymbol{\Phi} \in \mathbb{R}^{\boldsymbol{3n\times q}}$ is the subspace matrix, and $\boldsymbol{q} \in \mathbb{R}^{\boldsymbol{r}}$ is the reduced coordinates.

The main question about model reduction is how to properly choose subspace matrix $\boldsymbol{\Phi}$. In chapter 2, for tubular structure, we construct component-level subspace based on component mode synthesis(CMS) [8, 9]. In chapter3, we considered using spectral subspace or spatial subspace methods. Spectral subspace methods computed a set of global representative modal shapes by using PCA or modal analysis [10], [11]. However, it is known that a globally constructed modal subspace lacks the local deformations. To fix the limitation, we use a domain decomposition technique to construct subspace mode locally to capture more local deformation. Another issue is

the non-overlapping domain decomposition method could be unstable under large deformation because of explicitly handling of coupling between adjacent domains. The simulation would fail on the large deformed domain interfaces with most coupling methods like damped springs and rigid binding.

Another idea of subspace construction is the spatial reduction method. The method scatters DOFs sparsely over the deformable 3D mesh domain and utilizes blending functions to express the deformation. Blending methods like Cage-based or Free-form are widely used for shape modeling. The choice of DOFs for each domain could be variety like a linear transformation field, a local rigid frame, or an integration unit. The blending or weight functions smoothly blend the deformations across domains and implicitly deal with the domain coupling. Thus, the spatial reduction method presents a more stable result under extreme deformations.

In chapter 4, we will introduce complex step finite difference (CSFD) to simplify the implementation of nonlinear solid simulation. In a typical nonlinear solid simulation, we have to solve a nonlinear optimization problem at every time step. Normally, we choose Newton's method for fast convergence rate and evaluate first and second-order derivatives with an exact formula for best efficiency and accuracy. However, the implementation of the actual high order derivative is too complicated and difficult that guide us to use numerical derivative in some cases. For example, the finite difference method can be used. The forward difference scheme estimates the derivative as:

$$f'(x_0) = \lim_{\Delta x \to 0} \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x} \approx \frac{f(x_0 + h) - f(x_0)}{h}, \tag{1.2}$$

where the small perturbation $h \in \mathbb{R}$. With the finite difference method, it seems that the smaller perturbation is, the more accurate derivatives delivers. However, there is a lower bound of perturbation h because of the subtractive cancellation. The subtraction between two nearly equal numbers, such as $f(x_0 + h) - f(x_0)$, could

eliminate many of their significant digits and contaminate the result in the computer. Compare to the finite difference method, CSFD applies the perturbation h in the imaginary domain after promoting f to be a complex function. This can avoid the subtraction of the first-order terms in the complex Taylor expansion, and allows us to use the very small perturbation h without worrying about the subtractive cancellation. CSFD provides a highly accurate numerical derivative without deriving the actual formulation. We also provide acceleration techniques for CSFD and applications to show the convenience of CSFD to solve the difficult computation of derivatives.

In chapter 5, a neural network-based method named *NNWarp* is presented to simplify and accelerate the simulation of nonlinear deformable model. Our method is conceptually similar to stiffness warping [11] and modal warping [12], in which a linear solver is used after rotating the deformed shape back to its undeformed orientation. We leverage the neural network to map or *warp* a simplified constitutive law to a nonlinear one. In this dissertation, we choose the linear elasticity as the simplified constitutive law. Compare to nonlinear elasticity, The linear elasticity can be much faster during the run-time simulation with the same number of DOFs because of the constant stiffness matrix. Then the NNWarp can augment displacement per node to approximate the nonlinear deformation very fast with the advantages of parallel. During the warping, we compute three novel discriminative features, namely the *geodesic*, *potential* and *digression*, for each node to train a fairly shape- and tessellation- independent network which can be re-used for different geometries.

In chapter 6, we present an application of nonlinear solid simulation based on our novel techniques for visualization of the speaking tongue. The framework converts acoustic signals of human language (Chinese) to realistic 3D tongue animation sequences in real-time. Visualization of the tongue is an important tool for understanding human speech and produce realistic speech animation in AR/VR scene. To achieve a high quality of animation of the tongue, the system must provide precise in-

put information from sensors and a real-time deformable tongue model. The challenge is that the tongue is an interior organ inside of the oral cavity. Therefore, most optical sensors like video cameras can not be used to capture the input signal. Consider the high dependence between the motion of the tongue and the related pronunciation, we use voice signals as our input in the system which can be captured precisely with general microphones. The proposed framework will convert the voice signal to pre-defined bio-mechanical parameters of the tongue to reconstruct the motion by enforcing certain constraints during the simulation. We verified our results by comparing our deformed tongue with MRI reconstructed tongue model side-by-side.

Chapter 2

Interactive Design and Simulation of Tubular Structure

2.1 Introduction

In this chapter, we present a novel technique to accelerate the tubes simulation based on model reduction and linear elasticity. The accelerated simulator allows us to provide a real-time interactive design and simulation software for tubular structure.

Tubes serve as a type of important supporting structure and are commonly used in people's everyday life (Fig. 2.1). Traditionally, such supporting structures are often hollow to conserve the manufacture cost and self-weight. They mostly consist of regular cylinders, which are more budget-friendly for mass production with traditional manufacturing techniques. On the other hand, the rapid development of prototype technology (e.g. 3D printing) makes personalized and customized tube fabrication using generalized cylinders conveniently possible, which greatly expands the designing space of supporting tubes.

Figure 2.1: Two furniture designs using supporting tubes.

Although most existing computer-aided design (CAD) software (e.g. AutoCAD) well support the geometric design of such tubular structures. Users still need to manipulate many geometric degrees of freedom (DOFs) to model free-form tubes. Interpolatory and tangential controls at the boundary cross-sections are two widely-adopted mechanisms to control the shape of the tube. However, profile control [13] or solving higher-order differential equations [14] is still necessary to prevent shape distortion, which is often tedious or time-consuming. On the other hand, existing CAD packages merely focus on the aspect of shape editing while the structural properties of the 3D model remain unknown to novice users. Following the trend of design-simulation integration, current commercial produces start to enable user to analyze their design using finite element method. Unfortunately, an accurate simulation of the structural characteristic of a customized tubular structure is expensive because generalized shell element with high-order shape functions is usually required to avoid shear-locking artifact [15] and a simulator often possesses a large number DOFs that

is prohibitive to regular desktop computers, not to mention performing interactive structural analysis of the 3D model being edited.

As a response to the aforementioned challenges, we present a system for interactive design and simulation of supporting tubular structures. Tube components can be intuitively edited using boundary and skeletal controls and a complex tube system can be handily created by assembling tube components at their open interfaces. The underlying simulation is carried out using general quadratic nine-node quadrilateral element. A constraint subspace is constructed at each component, which serves as the primary subspace for the follow-up structural analysis. On the top of the constraint subspace, we build a load-dependent secondary sub-space named residual subspace, which is able to precisely capture the detailed intracomponent deflection due to the regional external loads without resorting to expensive full-space simulation. As a result, our system is able to provide interactive yet accurate structural analysis along with the editing operation of the tube.

Contribution In general, the contribution of our work can be briefly summarized as follows:

- This chapter presents a system integrating parametric shape editing and finite element method based structural analysis into a unified environment for the design and simulation of free-form tubular supporting structures.
- We provide user an intuitive shape design mechanism with lower geometric DOFs by using the boundary and skeletal controls to manipulate the geometry of each tube component.
- A new simulation strategy is proposed based on the fact that the supporting tube is often of light self-weight comparing to its external loads. We use a two-level subspace simulation that is able to accurately capture the deflection induced by external loads while still keep the simulator compact.

2.2 Related Work

Swept surface is often used to model general cylinders [16] by transforming cross-section curves along a smooth rotation field on a swept trajectory. Topics such as how to design smooth rotation field on a given trajectory [17, 18], how to interpolate cross-section curves [19, 20] and how to support profile editing [21, 22] are all well studied in the literature. However, it is tedious to manipulate lots of control vertices of swept surfaces represented by standard tensor product spline surfaces. Recently, You et al. [13] suggest modeling swept surfaces by solving ordinary differential equations (ODE). They showed that interpolatory and tangential boundary controls are available by using fourth-order ODE, which leads to lower DOFs in controlling the shape of swept surfaces. They also derived analytical solutions to six-order partial differential equations and gave extra curvature control to swept surfaces [23]. The similar idea is also exploited in shape modeling using meshed surfaces. In [14,24], the authors showed that generalized cylinders can be obtained by solving harmonic and higher-order harmonic equations. In our system, the geometric design of free-form supporting tubes is motivated by these existing studies.

Thin shell element is a natural choice for the tubular structure, which has a high width-thickness ratio. Such degeneracy motivates researchers, especially in graphics community, to seek for alternative energy models to capture the deformation of thin shell in a more efficient and intuitive manner such as spline/NURBS [25–27], hinge-based bending [28–31], or meshless method [32–34], rather than resorting to classic strain theory [35]. Zhang et al. [36] proposed to use 1D orientated rod element with incremental strain theory to model the thin shell structure, which could be considered as an extended version of mass-spring system. While compelling results have been reported, these methods only produce physically plausible animations while we are looking for an accurate simulation that directly serves for potential follow-up fabrication (e.g. via 3D printing).

Design-simulation integration has received increased attention recently and fabrication purposed design system becomes an active research topic. Simulation based optimization has been widely applied to make sure the fabricated object possesses the desired structural robustness [37–39], kinematic constraints [40, 41], and deformable behavior [42–44]. There are also many contributions trying to unify the simulation and the design processing. Umetani et al. [45] present a garment designing system that allows an interactive editing between 2D patterns and 3D simulated draped forms. Cirak et al. [46] propose to use subdivision surface for the design-simulation integration for thin-shell objects.

Simulation acceleration stands out a grand technical challenge for the integration of design simulation because an accurate finite element method (FEM) [15] simulation is often expensive while timely coupled design simulation environment is always favored. To accelerate the FEM simulation of thin shell, Seth et al. [47] employ a multi-resolution framework. In regular FEM simulation of 3D solid volume, subspace modal reduction is a widely used technique [48–50] and it can also be applied to accelerate thin-shell simulation [51].

Our method well complements exiting contributions by developing a design-simulation framework based on do-main decomposition [52] and finite element tearing and interconnect (FETI) method [53], as we notice that tubular structures are often component-wise and the geometric symmetry commonly exists. The geometry of each tube component is dealt with using boundary and skeletal controls. The structural behavior is simulated using quadratic nine-node quadrilateral mesh automatically generated via the surface parametrization, which will assign five DOFs for each free node. To accelerate the simulation, we construct the component-level subspace based on an engineering technique named component mode synthesis (CMS) [8, 9]. Improved simulation accuracy is achieved by computing the residual deflection within a load-dependent secondary subspace. As a result, our system is able to provide

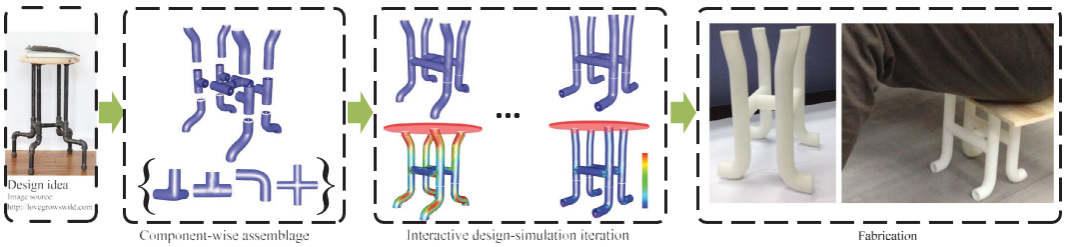


Figure 2.2: An overview of the proposed design-simulation system.

accurate stress analysis while keeping the simulator compact and efficient.

2.3 System Overview

Fig. 2.2 sketches an overview of the proposed design simulation system. The entire structure is composed of multiple tubular components which are inter-connected at their interfaces. The shape of each component is modeled as a swept surface that can be freely edited with an intuitive interface that allows users to manipulate key cross-section curves and their trajectory (Section 2.4). Based on the parameterization, a quadrilateral finite element mesh is automatically generated. Each of its element is a nine-node quadratic shell element (Section 2.5), where the mid-edge nodes are determined using cubic Hermite interpolation. We adopt a two-level subspace simulation strategy to accelerate the simulation so that an interactive structural analysis is made possible (Section 2.6) and the stress distribution can be timely visualized by the designer to ensure the tubular model is robust and stable under the prescribed external loads.

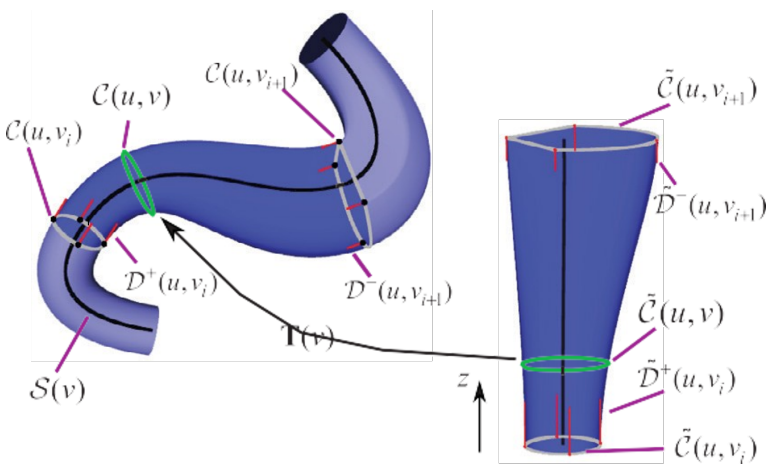


Figure 2.3: The boundary conditions and sweep trajectory of a free-form tube. The designed free-form tube is on the left and its shape space of cross sections is shown on the right.

2.4 Geometric Design of Free-form Tubes

Our goal of the geometric design is to provide users intuitive and flexible controls of free-form tubular components, which are modeled using swept surfaces in our system. Similar to previous works [13], we use boundary constraints to manipulate geometric variation of the cross-section of the tube along the neutral axis instead of using control vertices for the tensor product spline surfaces, since it is intuitive and requires fewer control parameters. Instead of profile editing introduced in [13], we employ the sweep trajectory to control the design in our system, or namely skeletal control. Comparing to multiple profile curves, one free-form tube only have one sweep trajectory, which significantly eases the overall shape editing operation. Specifically, as shown in Fig. 2.3, our system allows users to control (1) the sweep trajectory $S(v_i)$, (2) a few key cross-section curves $C_i(u) = C(u, v_i)$ at $S(v_i)$, and (3) each key cross-section curve's variation at both extensions $D^-(u, v_i)$ and $D^+(u, v_i)$, such that $D^-(u, v_i)$ and $D^+(u, v_i)$ are the tangent directions. The output swept surface $C(u, v)$

at $[v_i, v_{i+1}]$ must satisfy the following conditions:

$$\begin{aligned} &C(u, v_i) = C_i(u),\ \frac{\partial C(u, v_i)}{\partial v} = D^+(u, v_i) \\ &C(u, v_{i+1}) = C_{i+1}(u),\ \frac{\partial C(u, v_{i+1})}{\partial v} = D^-(u, v_{i+1}). \end{aligned} \tag{2.1}$$

You et al. [13] presented a formulation to construct the swept surface with the above constraints by solving a fourth-order ODEs. Unfortunately, we found that this approach yields unpleasant shape distortion if the boundary tangents are not in parallel, as shown in Fig. 2.4(a). To prevent such distortion, extra control information must be provided from the user such as profile control [13] or curvature control [14, 23],which inevitably induces more editing freedoms and could potentially make novice users confusing.

Alternatively, we combine the rotation minimizing frames (RMF) [18] together with the existing ODE-based techniques. The geometry of the swept surface is decoupled into two parts: the shape defined by cross-sections and their 3D embedding defined by sweep trajectories. In our system, we constrain each cross-section curve $C(u, v_i)$ to be planar and the editing operations associated with cross-section curves are performed with an intuitive 2D user interface. Suppose a rigid transformation $\mathbf{T}(v)$ maps a planar curve $\tilde{C}(u, v)$ to $C(u, v)$, we call $\tilde{C}(u, v)$ the shape of cross-section at $v = v_t$. Given a sparse set of key cross-section shapes $\tilde{C}_i(u, v) = \tilde{C}(u, v_i)$ for $i = 1, \cdots, n$ and a sweep trajectory $S(v)$, we are computing for interpolated cross-section shapes $\tilde{C}(u, v)$ and transformations $\mathbf{T}(v)$ for all $v_i \in [v_1, v_n]$. The transformation $\mathbf{T}(v)$ is composed of a translation $\mathbf{t}(v)$ and a rotation $\mathbf{R}(v)$. We define the translation $\mathbf{t}(v) = S(v)$, to make the cross-section curves sweep coincide with the specified trajectory $S(v)$. The rotation $\mathbf{R}(v)$ is computed by using the RMF on $S(v)$. Besides interpolatory constraints, we also allow users to control the tangent in the shape space of cross-sections. Specifically, we embed all planar cross-section shapes in 3D by defining v as the height (z direction) of $\tilde{C}(u, v)$, as shown in Fig. 2.3.

The tangential controls $\tilde{D}^+(u, v_i)$ and $\tilde{D}^-(u, v_i)$ in the shape space of cross-sections are also defined in this embedding. $\tilde{C}(u, v)$ is solved with the following tangential constraints:

$$\begin{aligned} &\tilde{C}(u, v_i) = C_i(u), \ \frac{\partial \tilde{C}(u, v_i)}{\partial v} = \tilde{D}^+(u, v_i) \\ &\tilde{C}(u, v_{i+1}) = C_{i+1}(u), \ \frac{\partial \tilde{C}(u, v_{i+1})}{\partial v} = \tilde{D}^-(u, v_{i+1}). \end{aligned} \tag{2.2}$$

We take the shape on the plane $z = v_j$ as the interpolated cross-section at $v = v_j$. Note that in our system, $\tilde{D}^-(u, v_i)$ and $\tilde{D}^+(u, v_i)$ provide only tangential control in the shape space of cross-sections. They are not the tangents on the swept surface if the sweep trajectory is not straight. In our implementation, we set $\tilde{D}^{\pm}(u, v_i) = \mathbf{R}^\top(v)\tilde{D}^{\pm}(u, v_i)$, where $\mathbf{R}(v)$ is the rotation of the RMF at v.

As shown in Fig. 2.4(b), with the same boundary conditions, swept surfaces using our method are free of distortion. In this example, the sweep trajectory is defined by a cubic Bezier curve whose end points and tangents are the same with the input boundary conditions. If profile control is still required, users can simply insert extra key cross-section curves to control the profile. Note that although existing commercial products support cross-section sketching followed by sweep path design, tangential control is mostly applied on the sweep trajectory but not on the surface. An alternative way is to directly manipulate on the control points, which leads to heavier data and interactions.

Since the cross-sections are interpolated analytically and the RMFs are computed explicitly, the computation associated with editing operations is negligible. Please note that our current system does not explicitly handle the self-intersection, which could be possible for highly curved tubes. An alert will be sent as soon as the user's editing leads to any self-collisions or intersections.

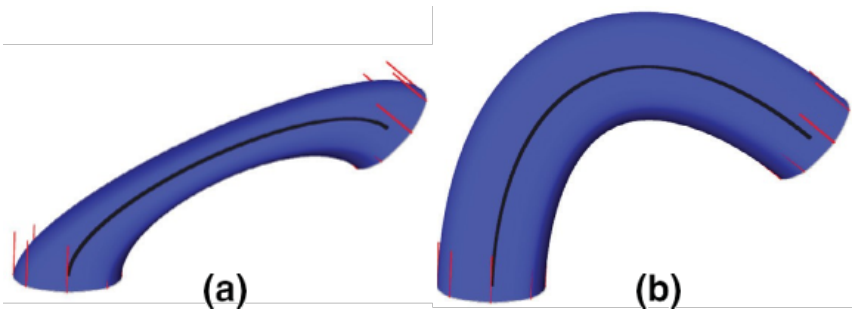


Figure 2.4: Boundary constrained swept surfaces using ODE-based techniques without (a) and with (b) sweep trajectory control.

2.5 Formulation of General Shell Element

Shell element is a degenerated structural element. Unlike regular 3D solid volumetric elements such as brick or tetrahedra, the dimension along its thickness is much smaller (e.g. over 50 times) than the other two dimensions. Such degeneracy leads serious numerical stability issue. As a result, necessary geometric/kinematic constraints and simplifications must be assumed. This section will briefly explain

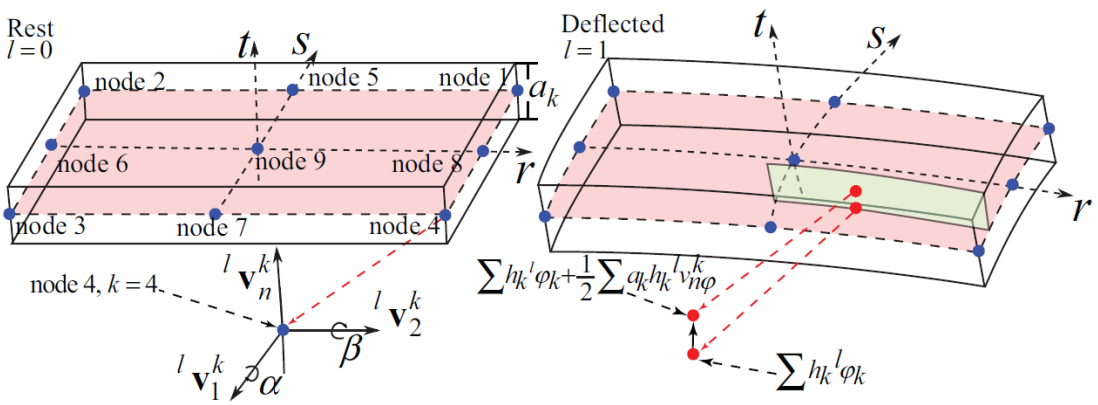


Figure 2.5: The boundary conditions and sweep trajectory of a free-form tube. The designed free-form tube is on the left and its shape space of cross sections is shown on the right.

the finite element formulation of the nine-node quadrilateral shell element. We refer readers to the related literature [15, 54], for a more detailed derivation.

The parametrization of the swept surface handily generates a quadrilateral mesh. Each four-node quadrilateral will be converted into a nine-node quadratic shell element. The extra mid-edge nodes are determined by using cubic Hermite interpolation so that the resulting nine-node element has smoothly-curved edges. For the ease of derivation, we adopt the isoparametric formulation, which begins with a standard shell element (Fig. 2.5) defined in a virtual rst coordinate frame or natural coordinate frame. This element spans from -1 to 1 in both r and s directions. For elements with arbitrary location and geometry, we take use of the Jacobian matrix to map it back to the real coordinate system.

For an arbitrary mass point within the element, all of its kinematic terms are interpolated using nodal shape functions. Shape functions always have ones at their host nodes and vanished values at the other nodes. High-order shape function can be considered as the superposition of the scaled low-order ones. Based on this property,

the shape function of this standard element can be easily written as:

$$
\begin{aligned}
h_1 &= \frac{1}{4}(1+r)(1+s) - \frac{1}{2}h_5 - \frac{1}{2}h_8 - \frac{1}{4}h_9 \\
h_2 &= \frac{1}{4}(1+r)(1+s) - \frac{1}{2}h_5 - \frac{1}{2}h_6 - \frac{1}{4}h_9 \\
h_3 &= \frac{1}{4}(1+r)(1+s) - \frac{1}{2}h_6 - \frac{1}{2}h_7 - \frac{1}{4}h_9 \\
h_4 &= \frac{1}{4}(1+r)(1+s) - \frac{1}{2}h_7 - \frac{1}{2}h_8 - \frac{1}{4}h_9 \\
h_5 &= \frac{1}{2}(1+r)(1+s) - \frac{1}{2}h_9 \\
h_6 &= \frac{1}{2}(1+r)(1+s) - \frac{1}{2}h_9 \\
h_7 &= \frac{1}{2}(1+r)(1+s) - \frac{1}{2}h_9 \\
h_8 &= \frac{1}{2}(1+r)(1+s) - \frac{1}{2}h_9 \\
h_9 &= (1-r^2)(1-s^2).
\end{aligned}
\tag{2.3}
$$

Fig. 2.5 shows a nine-node shell element in the rest (left) and deflected (right) configurations respectively, indicated with the superscript l. When $l = 0$, the corresponding variable is at the rest configuration; when $l = 1$, the corresponding variable is at the deflected configuration. A unit vector ${}^l\mathbf{v}_n^k = [{}^l v_{nx}^k, {}^l v_{ny}^k, {}^l v_{nz}^k]^\top$ is defined in the regular xyz coordinate frame at node k, which corresponds to the tangent direction of t axis in rst frame. ${}^l\mathbf{v}_1^k$ and ${}^l\mathbf{v}_2^k$ are two mutually perpendicular unit vectors sitting the plane normal to l ${}^l\mathbf{v}_n^k$. The infinitesimal rotations around ${}^l\mathbf{v}_1^k$ and ${}^l\mathbf{v}_2^k$ are denoted as α and β, which serve as extra two DOFs of node k.Therefore, each node possesses five DOFs i.e. x_k, y_k, z_k, α_k and β_k. Let $p(r, s, t)$ be an arbitrary mass point within the element. Its rest/deflected position can be interpolated as:

$$
{}^l\phi(r,s,t) = \sum_k h_k^l \phi_k + \frac{1}{2}\sum_k a_k h_k^l v_{n\phi}^k,\ \phi = x, y\ or\ z, \tag{2.4}
$$

where a_k is the thickness of the shell at node k. The first term in Eq. 2.4 corresponds to the regular shape function interpolation and the second term (i.e. $\frac{1}{2}\sum_k a_k h_k^l v_{n\phi}^k$

assumes that r and s displacements at $p(r,s,t)$ are linearly proportional to its t coordinate when r and s are fixed. The displacement vector $\mathbf{u}(r,s,t) = [u,v,w]^T$ at p can be interpolated in a similar fashion:

$$\begin{aligned} u(r,s,t) &= \sum_k h_k u_k + \frac{1}{2}\sum_k a_k h_k v_{nx}^k \\ v(r,s,t) &= \sum_k h_k v_k + \frac{1}{2}\sum_k a_k h_k v_{ny}^k \\ w(r,s,t) &= \sum_k h_k w_k + \frac{1}{2}\sum_k a_k h_k v_{nz}^k, \end{aligned} \tag{2.5}$$

where v_{nx}^k, v_{ny}^k and v_{nw}^k are the three components of the displacement vector $\mathbf{v}_n^k$ from ${}^0\mathbf{v}_n^k$ to ${}^1\mathbf{v}_n^k$ (i.e. $\mathbf{v}_n^k \triangleq {}^1\mathbf{v}_n^k - {}^0\mathbf{v}_n^k$). It can be expressed using nodal DOFs α_k and β_k such that

$$\mathbf{v}_n^k = -{}^0\mathbf{v}_2^k\alpha_k + {}^1\mathbf{v}_1^k\beta_k. \tag{2.6}$$

Substituting Eqs. 2.5 and 2.6 into Eq. 2.4 leads to the final interpolation of the shell element

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \sum_k \mathbf{H}_k \begin{bmatrix} u_k \\ v_k \\ w_k \\ \alpha_k \\ \beta_k \end{bmatrix}, \tag{2.7}$$

where

$$\mathbf{H}_k = \left[\begin{array}{ccc|c|c} h_k & & & -h_k\tfrac{t}{2}a_k^0 v_{2x}^k & h_k\tfrac{t}{2}a_k^0 v_{1x}^k \\ & h_k & & -h_k\tfrac{t}{2}a_k^0 v_{2y}^k & h_k\tfrac{t}{2}a_k^0 v_{1y}^k \\ & & h_k & -h_k\tfrac{t}{2}a_k^0 v_{2z}^k & h_k\tfrac{t}{2}a_k^0 v_{1z}^k \end{array}\right]. \tag{2.8}$$

The element stiffness matrix is in the format of

$$\mathbf{K}_e = \int_{V_e} \mathbf{B}^\top \mathbf{C}\mathbf{B} dv, \tag{2.9}$$

where $\mathbf{B}$ is the strain-displacement matrix obtained by applying the partial derivative to nodal displacements using Eq. 2.7. $\mathbf{C}$ is the stress-strain matrix determined by the mate-rial property. In this chapter, linear elasticity is assumed and $\mathbf{C}$ is constant.

Due to the degeneracy of the shell element along its t direction, the stress components normal to the midsurface must always be zero. As a result, we need transforms the $\mathbf{C}$ matrix to make it aligned with the orientation of the shell element. As shown in Fig. 2.6, let $\mathbf{r}$, $\mathbf{s}$, $\mathbf{t}$ be the unit vectors corresponding to three axes in the natural coordinate frame. They may be distorted and no longer orthogonal to each other when being viewed from the regular xyz coordinate frame. Let $\mathbf{r}'$ $\mathbf{s}'$ $\mathbf{t}'$ be the unit vectors at the tangent directions of the corresponding axis (note that $\mathbf{t}' = \mathbf{t}$). We can extract a new set of basis vectors $\bar{\mathbf{r}}$, $\bar{\mathbf{s}}$ and $\bar{\mathbf{t}}$ such that: $\bar{\mathbf{r}} = \frac{\mathbf{s}' \times \mathbf{t}'}{\|\mathbf{s}' \times \mathbf{t}'\|_2}$, $\bar{\mathbf{s}} = \frac{\mathbf{t}' \times \bar{\mathbf{r}}}{\|\mathbf{t}' \times \bar{\mathbf{r}}\|_2}$, and $\bar{\mathbf{t}} = \frac{\mathbf{t}'}{\|\mathbf{t}'\|_2}$. Afterwards, a transformation matrix $\mathbf{Q}_{sh} \in \mathbb{R}^{6\times 6}$ can be reconstructed from the direction cosines of the $\mathbf{r}$, $\mathbf{s}$, and $\mathbf{t}$ measured in xyz coordinate frame, such that:

$$\mathbf{Q}_{sh} = \begin{bmatrix} l_1^2 & m_1^2 & n_1^2 & l_1 m_1 & m_1 n_1 & n_1 l_1 \\ l_2^2 & m_2^2 & n_2^2 & l_2 m_2 & m_2 n_2 & n_2 l_2 \\ l_3^2 & m_3^2 & n_3^2 & l_3 m_3 & m_3 n_3 & n_3 l_3 \\ ll_{1,2} & mm_{1,2} & nn_{1,2} & lm_{1,2} & mn_{1,2} & nl_{1,2} \\ ll_{2,2} & mm_{2,3} & nn_{2,3} & lm_{2,3} & mn_{2,3} & nl_{2,3} \\ ll_{3,2} & mm_{3,1} & nn_{3,1} & lm_{3,1} & mn_{3,1} & nl_{3,1} \end{bmatrix}, \tag{2.10}$$

where

$$\begin{aligned} &l_1 = \cos(\mathbf{x}, \bar{\mathbf{r}});\ m_1 = \cos(\mathbf{y}, \bar{\mathbf{r}});\ n_1 = cos(\mathbf{z}, \bar{\mathbf{r}}); \\ &l_2 = \cos(\mathbf{x}, \bar{\mathbf{s}});\ m_2 = \cos(\mathbf{y}, \bar{\mathbf{s}});\ n_2 = cos(\mathbf{z}, \bar{\mathbf{s}}); \\ &l_3 = \cos(\mathbf{x}, \bar{\mathbf{t}});\ m_3 = \cos(\mathbf{y}, \bar{\mathbf{t}});\ n_3 = cos(\mathbf{z}, \bar{\mathbf{t}}); \end{aligned} \tag{2.11}$$

The notation $ab_{i,j}$ denotes $a_i b_+ a_j b_i$, for instance $lm_{1,2} = l_1 m_2 + l_2 m_1$. The transformed $\mathbf{C}$ matrix is computed as

$$\mathbf{C}_{sh} = \mathbf{Q}_{sh}^\top \mathbf{C} \mathbf{Q}_{sh}. \tag{2.12}$$

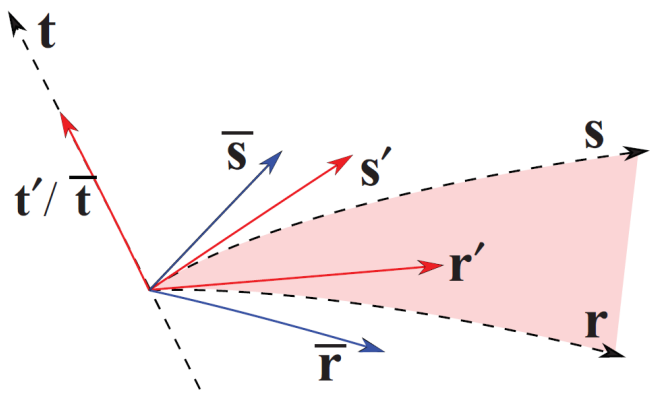


Figure 2.6: Extracting orthogonal basis vectors $\bar{\mathbf{r}}$, $\bar{\mathbf{s}}$ and $\bar{\mathbf{t}}$.

The volumetric integral in Eq. 2.9 is evaluated numerically at 18 sampling points $p_i(r_i, s_i, t_i)$ according to *Gauss-Legendre* integration strategy

$$\mathbf{K} \approx \sum_i \det(\mathbf{J}_i) w_i \mathbf{B}^\top \mathbf{C}_{sh} \mathbf{B}, \tag{2.13}$$

where $\mathbf{J}_i$ is the Jacobian matrix encoding how rst frame is transformed to xyz frame at p_i, w_i is the constant sampling weight of p_i. The r, s and t coordinates of p_i are selected from the combinations of $r = \pm 0.77460, 0$, $s = \pm 0.77460, 0$, and $t = \pm 0.57735$.

2.6 Model Reduction of Tubular structure

We project the FEM simulator into a preconstructed sub-space to accelerate the associated computation in order to provide an interactive design-simulation interplay. It is well-known that the cost of subspace simulation acceleration is the accuracy compromise as the system response beyond the predefined subspace cannot be captured. This problem is dealt with by using a secondary residual subspace, in

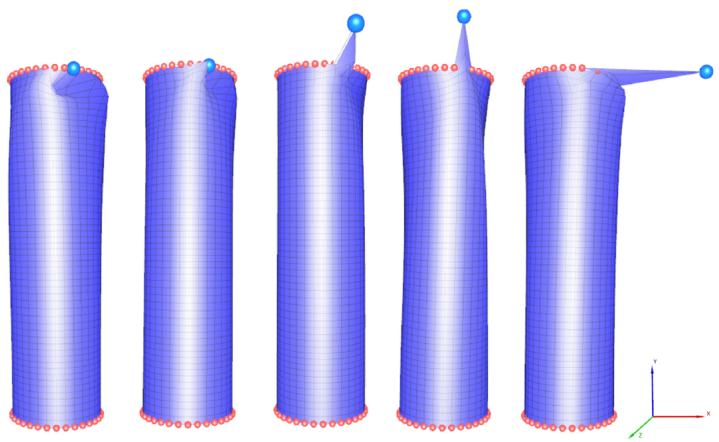


Figure 2.7: The shapes of five constraint modes associated with a boundary node (highlighted as blue node). Other restrained boundary DOFs are marked as red nodes. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

our system, to accurately obtain the necessary intracomponent deflection due to the external load.

Our subspace construction strategy is devised based on the following three important observations/assumptions: The entire tube system is composed of multiple small-size components, many of which are of the same geometry because of the geometric symmetry.

Accordingly, we compute the subspace basis vectors or modes at each of the tubular components individually so that the expensive computation associated with global finite element mesh is avoided. For tubular components of the same geometry but different locations/orientations, the modes can be directly synthesized by applying the corresponding rotation and translation.

Only a static equilibrium structural analysis is required in our case while vibrational response under highly accelerated velocity field (e.g. a launching rocket) is not our concern.

Correspondingly, we safely ignore the dynamical structural analysis and only focus on the static equilibrium analysis with the format of $\mathbf{Ku} = \mathbf{f}$.

Consequently, we build a complementary subspace at run-time to capture the residual deformation that is not included in the primary subspace. It can be mathematically proven that such two-level subspace simulation strategy is able to produce the same result as using the full-space.

2.6.1 Constraint Subspace - the Primary Subspace

Our subspace construction method is inspired by the boundary mode [55], which is essentially an extension of the classic CMS technique [9]. The basis vectors are computed per component by solving a static equilibrium system. For a given tubular component, we classify all of its DOFs into two categories namely, the internal DOF set and boundary DOF set, which are denoted using subscripts i and b respectively in the following formulation. We impose one unit displacement to each boundary DOF while restrain the rest boundary DOFs anchored, which leads to

$$\begin{bmatrix} \mathbf{K}_{ii}^{L} & \mathbf{K}_{ib}^{L} \\ \mathbf{K}_{ib}^{L\top} & \mathbf{K}_{bb}^{L} \end{bmatrix} \begin{bmatrix} \Phi_{i}^{L} \\ \Phi_{b}^{L} \end{bmatrix} = \begin{bmatrix} \mathbf{F}_{i}^{L} \\ \mathbf{F}_{b}^{L} \end{bmatrix} \tag{2.14}$$

where superscript L denotes the variables are local. Φ_b^L is an identity matrix corresponding to the unit boundary excitement imposed. $F_i^L = 0$ as no external loads are applied at internal DOFs. The unknown internal response Φ_i^L can be easily computed by expanding the first line of Eq. 2.14:

$$\Phi_{i}^{L} = -\mathbf{K}_{ii}^{L^{-1}}\mathbf{K}_{ib}^{L}, \tag{2.15}$$

and mode vectors at the component are assembled by concatenating the Φ_i^L and Φ_b^L such that $\Phi^L = [\Phi_i^{L\top}|\Phi_b^{L\top}]^\top$. We notice that the formulation of Phi^L is consistent with the constraint mode in CMS [8]. Therefore, we refer the subspace spanned by

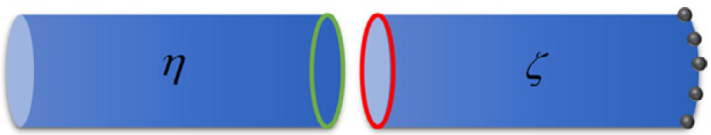


Figure 2.8: An illustrative example showing the coupling of tubular components η and ζ assuming that an appropriate boundary condition has been specified at the highlighted nodes.

Phi^L as the constraint subspace. Fig. 2.7 shows the shapes of five constraint modes associated with a boundary node while other boundary nodes (red nodes in the figure) are fixed. With constraint modes, the system's displacement can be expressed using the reduced coordinates at each component such that

$$\mathbf{u} = \bar{\Phi}\bar{\mathbf{q}}, \tag{2.16}$$

where $\bar{\Phi} = \text{diag}(\Phi^1, \Phi^2, \cdots, \Phi^k)$ and $\bar{\mathbf{q}} = [\mathbf{q}^{1\top}, \mathbf{q}^{2\top}, \cdots, \mathbf{q}^{k\top}]$ are the *global* subspace matrix and generalized displacement vector of the system with k tubular components.

2.6.2 Multiplier-free Component Coupling

All the tubular components are mutually connected at their interfaces. Such coupling can be formulated as the interface constraint (IC) between a pair of adjacency components. As shown in Fig. 2.8, IC requires that duplicated boundary DOFs from components η and ζ must always have identical values e.g. $\mathbf{u}_b^\eta = \mathbf{u}_b^\zeta$. It can be re-written using the reduced coordinate in constraint subspace

$$\mathbf{E}_b^\eta \Phi^\eta \mathbf{q}^\eta = \mathbf{E}_b^\zeta \Phi^\zeta \mathbf{q}^\zeta, \tag{2.17}$$

where $\mathbf{E}_b^\eta$ and $\mathbf{E}_b^\zeta$ are two elementary matrices extracting boundary DOFs from each component. A commonly-adopted approach to enforce Eq. 2.17 is to use the Lagrange multiplier method, which explicitly formulates the interface forces as the unknown

multipliers [53]. While our simulator can also be handled this way, there are two obvious drawbacks associated with multiplier-based solution: (1) the dimension of the resulting linear system is greatly increased due to the existence of the multipliers and the redundancy of the boundary DOFs and (2) the system matrix is no longer a symmetric positive definite (SPD) matrix as we have vanished diagonal elements at locations corresponding to the IC. Consequently, the effectiveness of subspace acceleration is compromised.

Alternatively, we enforce the interface constraint without replying on the Lagrange multiplier method to maintain a more compact and better-conditioned subspace solver.

Note that ICs are a set of linear constraints, which can be re-written as

$$\mathbf{C}\bar{\mathbf{q}} = \mathbf{0}, \tag{2.18}$$

where $\bar{\mathbf{q}} = [\mathbf{q}^{\eta\top}, \mathbf{q}^{\zeta\top}]^\top$ and $\mathbf{C} = [\mathbf{E}_b^\eta \Phi^\eta | - \mathbf{E}_b^\zeta \Phi^\zeta]$ as in the case shown in Fig. 2.8. $\mathbf{C} \in \mathbb{R}^{c\times d}$ is a rectangular matrix, where c is the number of ICs of the system and d is the total number of the reduced coordinates at components η and ζ including the duplicated interface DOFs. Obviously, $d > c$, therefore $\mathbf{C}$ can be further split into two parts

$$\mathbf{C} = [\mathbf{C}_1 | \mathbf{C}_2], \tag{2.19}$$

such that $\mathbf{C}_1 \in \mathbb{R}^{c\times c}$ is a full-rank square matrix. Eq. 2.18 can be re-written as

$$\mathbf{C}_1\mathbf{q}_d + \mathbf{C}_2\mathbf{q}_f = \mathbf{0}, \tag{2.20}$$

where $\mathbf{q}_f$ represents a subset of $\bar{\mathbf{q}}$ consisting of only independent or free DOFs and $\mathbf{q}_d$ represents a subset of dependent DOFs. In the example shown in Fig. 2.8, if the DOFs on the red interface are free DOFs, the DOFs on the green interface are dependent ones and vice versa.

Since $\mathbf{C}_1$ is full-rank, we can use $\mathbf{q}_f$ to represent the other "redundant" DOFs $\mathbf{q}_d$

$$\mathbf{q}_d = -\mathbf{C}_1^{-1}\mathbf{C}_2\mathbf{q}_f, \tag{2.21}$$

as well as the complete $\bar{\mathbf{q}}$ vector

$$\bar{\mathbf{q}} = \begin{bmatrix} \mathbf{q}_f \\ \mathbf{q}_d \end{bmatrix} = \begin{bmatrix} \mathbf{I} \\ -\mathbf{C}_1^{-1}\mathbf{C}_2 \end{bmatrix} \mathbf{q}_f. \tag{2.22}$$

Substituting Eq. 2.22 into Eq. 2.16 yields

$$\mathbf{u} = \bar{\Phi} \begin{bmatrix} \mathbf{I} \\ -\mathbf{C}_1^{-1}\mathbf{C}_2 \end{bmatrix} \mathbf{q}_f = \Phi\mathbf{q}, \tag{2.23}$$

where $\Phi \triangleq \begin{bmatrix} \mathbf{I} \\ -\mathbf{C}_1^{-1}\mathbf{C}_2 \end{bmatrix}$ and $\mathbf{q} \triangleq \mathbf{q}_f$. The full-space equilibrium $\mathbf{Ku} = \mathbf{f}$ can be directly projected onto the new subspace spanned by Φ where IC is implicitly encoded

$$\mathbf{K}_d\mathbf{q} = \mathbf{f}_q. \tag{2.24}$$

Here, $\mathbf{K}_q = \Phi^\top\mathbf{K}\boldsymbol{\Phi}$ and $\mathbf{f}_q = \Phi^\top\mathbf{f}$. This derivation can be easily extended for multiple components.

2.6.3 Residual Subspace - the Secondary Subspace

Using constraint modes is able to significantly improve the performance, yet it also sacrifices the accuracy of the simulation. Tubular supporting structure is often exerted concentrated regional loads of large magnitude. While the deflections at load-free components are accurately captured (when the gravity effect can be ignored) because all the inter-component stress propagation are losslessly passed via the interface whose DOFs are fully preserved within constraint subspace, deflection at the loading components where the forces are applied is not able to be well represented with constraint modes. To resolve this issue, a local secondary subspace is built to capture the residual deflection and improve the simulation accuracy at loading components,

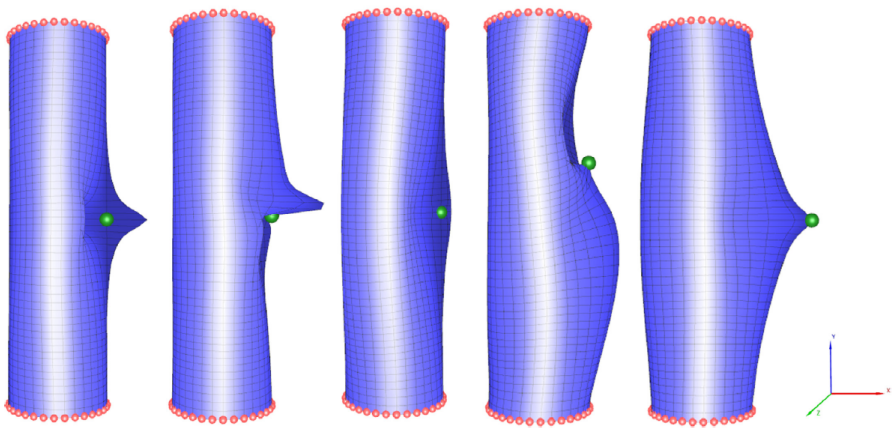


Figure 2.9: The shapes of five residual modes associated with an exciting node (green).

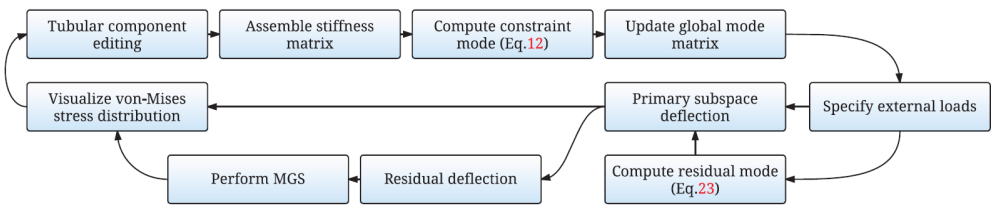


Figure 2.10: Overall computational procedures of the two-level subspace simulation method.

which is detailed in this subsection. As all the formulation is for a certain loading component, the superscript L is omitted.

We denote all the internal DOFs undertaking the external loads as the exciting DOFs while all the other internal DOFs as passive DOFs. They are symbolized using subscripts e and p respectively. Similar to constraint mode, a unit displacement is imposed to the each of the exciting DOF while keep other exciting DOFs and boundary DOFs fixed. We restrain the boundary DOFs so that the complementary deflection computed will not affect the status of other load-free components. A

equilibrium system can be listed accordingly:

$$\begin{bmatrix} \mathbf{K}_{ee} & \mathbf{K}_{ep} & \mathbf{K}_{eb} \\ \mathbf{K}_{ep}^{\top} & \mathbf{K}_{pp} & \mathbf{K}_{pb} \\ \mathbf{K}_{eb}^{\top} & \mathbf{K}_{pb}^{\top} & \mathbf{K}_{bb} \end{bmatrix} \begin{bmatrix} \Psi_e \\ \Psi_p \\ \Psi_b \end{bmatrix} = \begin{bmatrix} \mathbf{F}_e \\ \mathbf{F}_p \\ \mathbf{F}_b \end{bmatrix}, \tag{2.25}$$

where $\Psi_e = \mathbf{I}$ and $\Psi_b = \mathbf{0}$ correspond to the imposed unit displacement at exciting DOFs and anchored boundary DOFs. $\mathbf{F}_p = \mathbf{0}$ as no forces are applied at the passive DOFs. Again, the superscript L indicating the local variables is omitted here. The unknown system response at passive DOFs can be computed by expanding the second line of Eq. 2.25

$$\Psi_p = -\mathbf{K}_{pp}^{-1}\mathbf{K}_{ep}^{\top}. \tag{2.26}$$

We use $span(\Psi)$ to represent the subspace spanned by Ψ, name it as the residual subspace whose basis vectors are residual modes. Fig. 2.9 shows the shapes of five residual modes associated with an internal exciting node.

The residual modes are employed based on the fact that the response of a linear system of a composite input is equivalent to the superposition of the system's responses with respect to each individual input. In fact, it can be proven that the superset of Φ and Ψ is able to completely capture the deflection at loading components. We refer readers to Appendix A for mathematical proof details.

In other words, it means that an accurate result will be obtained if the system is solved within $span(\Phi) \cup span(\Psi)$

$$\begin{bmatrix} \Phi^{\top} \\ \Psi^{\top} \end{bmatrix} \mathbf{K}[\Phi|\Psi] \begin{bmatrix} \Phi^{\top}\mathbf{f} \\ \Psi^{\top}\mathbf{f} \end{bmatrix} \tag{2.27}$$

or

$$\begin{bmatrix} \mathbf{K}_{\Phi\Phi} & \mathbf{K}_{\Phi\Psi} \\ \mathbf{K}_{\Psi\Phi} & \mathbf{K}_{\Psi\Psi} \end{bmatrix} \begin{bmatrix} \mathbf{q} \\ \mathbf{p} \end{bmatrix} = \begin{bmatrix} \mathbf{f}_{\phi} \\ \mathbf{f}_{\Psi} \end{bmatrix}, \tag{2.28}$$

where

$$\begin{cases} \mathbf{K}_{\Phi\Phi} = \Phi^\top \mathbf{K} \Phi \\ \mathbf{K}_{\Phi\Psi} = \Phi^\top \mathbf{K} \Psi \\ \mathbf{K}_{\Psi\Phi} = \Psi^\top \mathbf{K} \Phi \\ \mathbf{K}_{\Psi\Psi} = \Psi^\top \mathbf{K} \Psi \end{cases}. \tag{2.29}$$

Here, $\mathbf{q}$ and $\mathbf{p}$ are the reduced coordinates of constraint modes and residual modes. While Φ can be pre-computed for each component, Ψ is a load-dependent matrix as different external loads would specify different exciting DOF sets and therefore, yield different mode matrices. It implies that as soon as the external loads are changed, the entire system must be re-computed, which significantly downgrades the usability of the system.

We notice that if the off-diagonal blocks (e.g. $\mathbf{K}_{\Phi\Psi}$ and $\mathbf{K}_{\Phi\Psi}^\top$) in Eq. 2.28 are zero, the constraint subspace and residual subspace will be decoupled and the computation of $\mathbf{q}$ and $\mathbf{p}$ are isolated. Therefore, we apply the *modified Gram-Schmidt process* (MGS) [56] towards Ψ with respect to $\mathbf{K}\Phi$, which yields a new set of residual modes $\tilde{\Psi}$ such that $\tilde{\Psi} \perp \mathbf{K}\Phi$ or $(\mathbf{K}\Phi)^\top \tilde{\Psi} = \mathbf{0}$. Boundary DOFs are always fixed during the computation of Ψ. On the contrary, there always exists one non-zero boundary DOF in the constraint mode. Such properties of constraint modes and residual modes guarantee that $span(\Phi) \cup span(\Psi) = \varnothing$. Therefore, $span(\Psi) = span(\tilde{\Psi})$ and $span(\Phi) \cup span(\Psi) = span(\Phi) \cup span(\tilde{\Psi})$. Substituting Ψ with $\tilde{\Psi}$, Eq. 2.28 is simplified to

$$\begin{cases} \mathbf{K}_{\Phi\Phi}\mathbf{q} = \mathbf{f}_\Phi \\ \mathbf{K}_{\tilde{\Phi}\tilde{\Phi}}\mathbf{q} = \mathbf{f}_{\tilde{\Phi}} \end{cases}. \tag{2.30}$$

The final component deflection is computed using

$$\begin{aligned} \mathbf{u} &= [\Phi|\tilde{\Psi}][\mathbf{q}^\top|\mathbf{p}^\top]^\top \\ &= \Phi\mathbf{q} + \Psi\mathbf{p} \\ &\triangleq \mathbf{u}_\Phi + \mathbf{u}_{\tilde{\Psi}}. \end{aligned} \tag{2.31}$$

Note that $\mathbf{u}_\Phi$ is the constrain subspace displacement. Therefore, we only need to calculate an incremental displacement $\mathbf{u}_\psi$ in order to obtain the exact full-space result at loading component. Fig. 2.10 summarizes the major computational procedures in our system.

2.7 Experimental Results

We report and discuss experiments we have conducted in this section. Please refer to the accompanying video for more results including the test use of the fabricated tubular models.

2.7.1 Hardware and Software Platform

Our experiments are carried out on a Dell Optiplex 9010 workstation computer equipped with an Intel i7-3770, 3.40 GHz CPU and 16G memory. The proposed system is implemented on 64-bit Microsoft Windows 7 using Visual Studio 2010. We use Eigen numerical library [57] for most linear system related calculations. Note that we only use the single core implementation however, many computations (e.g. per-component subspace construction) can be trivially parallelized using multi-threading.

2.7.2 Four-node Element vs. nine-node element

Existing FEM literature [15, 54] have mentioned that linear four-node element is not a good choice for general shell simulation. It is partially because the governing stress equilibrium is characterized using a second-order partial differential equation. The adoption of weak form (well-known as virtual work principle in the context of continuum mechanics) allows the usage of linear interpolation functions (e.g. linear

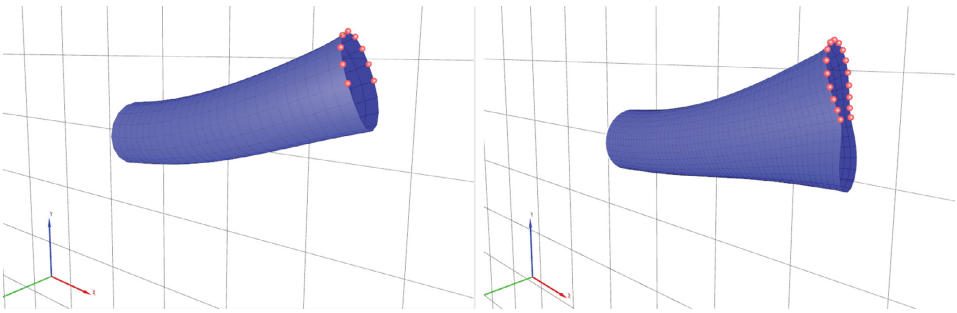


Figure 2.11: Shear locking using linear four-node element (left) which eliminates the bending deflection while quadratic nine-node element (right) does not have such artifact.

element) however, the accuracy of the simulation is compromised. The adoption of the linear element also leads to the shear locking artifact, which is shown in Fig. 2.11. The regular cylinder-shaped tube is simulated using the same number of four-node shell elements (left) and nine-node shell elements (right), respectively. The external forces are applied at the highlighted nodes in the positive y direction. The bending deformation can be well observed with nine-node element which is however, "locked" with four-node element.

2.7.3 User interface and Implementation Details

Fig. 2.12 shows a screen capture of the user interface of the proposed design-simulation system. Right to the main 3D view, our system provides an intuitive interface for the user to specify the boundary (top) and skeletal (bottom) controls of tube components. Our system maintains a tube library shown blew the main 3D view. The geometry of each tube component can be freely edited. Immediately after the geometric edit of a component is committed, the updated component will be inserted into the library as a new component. All the related pre-computation is also carried out at this stage. On average, a tube component holds about 4000-7000

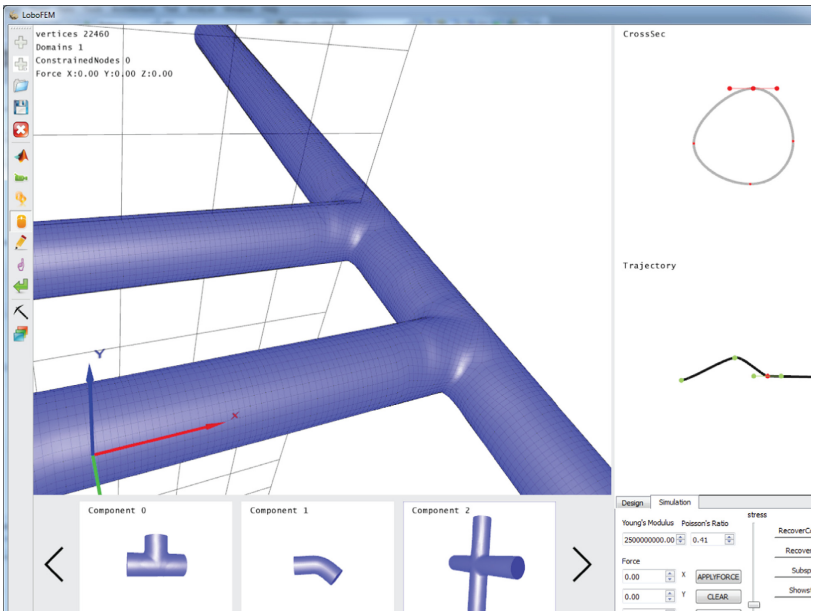


Figure 2.12: The user interface of the proposed system.

DOFs. The related computations like assembling the stiffness matrix $\mathbf{K}$, computing the constraint modes Φ as well as calculating the matrix-vector product of $\mathbf{K}\Phi$, which is for the potential MGS to be applied if this component is a loading component, can be done in real-time. During the model assemblage, each component can be freely copied and pasted. User is also able to specify the geometric symmetry during the editing so that the shape edits applied at a component will be automatically mapped to its geometrically symmetric counterparts like the stool legs shown in Fig. 2.2.

As soon as the entire structure is assembled, we need to build the global Φ matrix (Eq. 2.23) as the primary subspace basis vectors. Therefore matrices $\mathbf{C}_1$ and $\mathbf{C}_2$ (Eq. 2.19) must be identified. They can be efficiently found as each column in the original $\mathbf{C}$ matrix corresponds to a system DOF while all the row vectors in $\mathbf{C}$ are guaranteed to be linearly independent (as long as the IC are not redundantly defined in $\mathbf{C}$). Therefore, we only need to construct another elementary matrix encoding the necessary column permutation to move all the columns corresponding to independent

DOFs to the left-end of the matrix. As long as the topology of the tubular structure is not altered, this elementary matrix remains the same.

After the displacement is computed. The strain vector $\epsilon \in \mathbb{R}^{6\times 1}$ can be easily evaluated using the strain-displacement matrix $\mathbf{B}$, which is further converted to the stress vector σ according to the assumed linear elasticity: $\sigma = \mathbf{C}\epsilon$. Finally, we visualize the von Mises stress using the GLSL shader. The von Mises stress is a scalar and can be computed as

$$\sigma_{von}^2 = \frac{\sigma_{1,2}^2 + \sigma_{2,3}^2 + \sigma_{3,1}^2 + 6(\sigma_{12}^2 + \sigma_{23}^2 + \sigma_{31}^2)}{2}, \tag{2.32}$$

where $\sigma_{i,j} = \sigma_{ii} - \sigma_{jj}$ and σ_{ij} is the i, j component in the tensor representation of σ. We simulate the tubular structure of stereolithography (SLA) material with Young Modulus of $2.5e9$ and Poisson Ration of 0.41. Regions with high von Mises stress are likely to fail under the prescribed external loads as shown in Fig. 2.14.

Fig. 2.13 shows the snapshots of using our system for designing and simulating various tubular structures. Since authors are not professional designers, we just follow some design ideas searched from internet shown in the leftmost column of the figure. When high-stress regions is observed, we apply some further geometric edits to the model including altering the shapes of the cross-sections at critical region (row 1, the lamp stand model), reducing the curvature connecting neighbor components (row 2, the laptop holder model), adding extra supporting components (row 3, the bookshelf model) and reducing the force moment (row 4, the camera rack model). The edited regions are highlighted in the rightmost column in the figure.

2.7.4 Time Performance

Table 1 reports the detailed statistic of the 3D models we have tested. We compare the time performance of the proposed two-level subspace simulation method with the

Figure 2.13: Snapshots of using the proposed design-simulation system.

full-space simulation as well as the subspace simulator using the Lagrange multiplier method. While the full-space system can be solved using the sparse Cholesky solver (the built-in SimplicialLLT routine in Eigen library), the simulation is not interactive along with the design operations with lags of seconds. On the other hand, the multiplier based subspace solver often has doubled or tripled size comparing with our method, due to the dulcification of the boundary DOFs as well as the explicit formulation of unknown multipliers. In addition, the resulting system matrix is no longer SPD either and cannot be handled with LLT decomposition. Therefore, the performance data listed in Table 1 is the one using the LU decomposition(the built-in

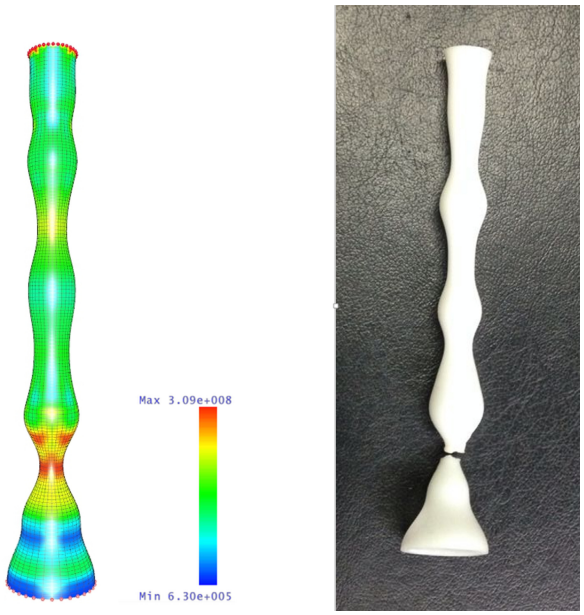


Figure 2.14: The 3D printed lamp stand fails and the failure location matches the area where high stress distribution is observed.

PartialPivLU solver in Eigen library), which is slower than LLT decomposition in most cases. As a result, even with much fewer DOFs, Lagrange multiplier based

Model	#Ele.	#Nodes	#DOFs	#S(L)DOFs	SDOFs	FT(s)	S(L)T(s)	ST(s)	#Com.
Lamp stand	5184	5216	25,920	1,120	480	0.406	0.093	0.01	3
Stool legs	6456	25,946	129,090	5,160	2,040	2.108	7.278	0.633	14
Bookshelf	25,824	19,547	97,750	5,120	2,080	1.532	6.960	0.66	11
Bookshelf edited	25,280	22,942	114,710	5,760	1,920	1.861	9.484	0.51	13
Camera rack	22,400	22,170	110,850	4,920	2,240	2.060	6.301	0.62	13
Laptop holder	12,096	12,128	60,480	3,040	1,120	0.784	1.472	0.103	7

Table 2.1: Time performance of our method, full-space simulator as well as subspace simulator using Lagrange multiplier method. #Ele.: the number of elements; #Nodes: the number of free nodes; #DOFs: the number of full-space DOFs: #S(L)DOFs the size of the simulator using constraint mode and Lagrange multiplier method; SDOFs: the size of the simulator using the propose multiplier-free coupling method; FT: time used to solve the system in full-space; S(L)T: time used to solve the multiplier-based subspace system; ST: time for our method; #Com: the number of the tube components.

subspace solver could be even slower than the full-space solver.

The proposed multiplier-free coupling mechanism will have a dense SPD matrix of much smaller size. Therefore, it is much more efficient than the above-mentioned two solvers. For 3D models with over 100k full-space DOFs, our method is still able to perform the accurate structural analysis at an interactive rate. We would like to remind the reader that unlike most existing subspace model reduction methods, our method does not compromise simulation accuracy while making the simulation an order of magnitude faster. Extra computations are required for applying the MGS, constructing the residual subspace and solving the secondary deflection. However, such computations are light-weight as they are conducted at the component level. In most cases, they can be finished within milliseconds.

Chapter 3

Nonlinear Elasticity with Overlapped Domain Decomposition

3.1 Introduction

In this chapter, we will continue discussion about model reduction techniques but for more general deformable nonlinear elastic model. Realistically simulating nonlinear deformable objects is known to be expensive, which drives a great amount of research efforts for developing accelerating techniques. An intuitive thought is to leverage the fact that deformations in reality are often of low rank, as elastic material models themselves effectively penalize high-frequency shape variations. Speedups of orders of magnitude can be obtained by removing less important degrees of freedom (DOFs). The core question for such *model reduction* method is how to utilize limited DOFs to achieve a better deformation expressivity. This objective is often dealt with either *spectrally* or *spatially*.

Figure 3.1: Overlapping quadratic domains make the simulation robust even under large deformations. The one-inch-tall bunny model is forced to pass a funnel whose inner diameter is only 0.3 inch. Our method yields plausible animations (the red bunny) at an interactive rate comparable to the fullspace simulation (the blue bunny). Most existing non-overlapping multi-domain simulators (i.e. [1, 2]) fail in this challenging test. Indeed, our simulator remains stable even when the funnel's diameter is reduced to 0.2 inch (Fig. 3.16). Please refer to the supplementary video and executables for more details.

Spectral subspace methods assign each DOF with a global representative modal shape or *mode*, often obtained using PCA or modal analysis [58, 59]. They rely on a dedicated pre-computation to select key modes. Some recent research further accelerates the pre-computation [60, 61] nevertheless, it is still at the order of $O(rN^2)$, where r stands for the number of modes and N is the size of the input model. It is also known that a globally constructed modal subspace lacks the capability of capturing local deformations. To remedy this limitation, the domain decomposition method (DDM) trends to be a more attractive option. It allows a domain-level mode customization and makes the local pre-computation much more efficient (i.e. $O(rN^2/d)$ for d domains, which is parallelizable and re-usable if domains are of the same geometry). When domains are *non-overlapping*, the influence of domain's subspace is analogous to the nodal *shape function* in the finite element method (FEM), which evaluates 1 locally and 0 elsewhere. As an unpleasant consequence, domains need to be explicitly coupled due to such boundary discontinuity. This gives rise to another concern regarding the simulation robustness under large deformations. Highly deformed domain interfaces could fail most coupling methods adopted in

existing nonlinear multi-domain simulators like rigid binding [1], damped springs [62], or coupling elements [2].

Another collection of acceleration techniques, referred to as *spatial reduction* here, scatters DOFs sparsely over the deformable body and utilizes blending functions to express the deformation in between, similar to the Cage-based [63] or the Free-from [64] schemes widely used for shape modeling. Here, the concept of DOF is not limited to the nodal displacement. It could be a linear transformation field [3], a local coordinate frame [4], or an integration unit [65]. The adopted blend or *weight functions* smoothly mix deformations across domains and unnecessitate an explicit domain coupling. As a result, the spatial reduction behaves more stably against extreme deformations. This framework is also better suited for local adaptivity and refinement [66, 67] than the spectral method. On the downside, since weight functions are typically calculated geometrically, they do not accommodate real material parameters like the Young's modulus and the Poisson's ratio. The deviation of the resulting deformation from the fullspace standard is often visually noticeable.

	Single-domain spectral reduction	Multi-domain spectral reduction	Spatial reduction	Our method
Fast pre-computation	⛔	✓	✓	✓
Good nonlinear expressivity	✓	✓	⛔	✓
Robust under large deformation	✓	⛔	✓	✓
Good local adaptivity	⛔	⛔	✓	✓

Figure 3.2: Pros and cons of existing single- and multi-domain reduction techniques for nonlinear deformable models.

As outlined in Fig. 3.2, our method supplements state-of-the-art spatial reduction techniques and tries to provide better answers to following three important how-tos:

- How to choose suitable deformation DOFs?
- How to assign limited DOFs in a more profitable way?
- How to design a good weight function?

We show that it is *essential*, for nonlinear models, to employ high-order DOFs in the spatial reduction, and we build our reduced simulator using overlapping quadratic domains so that it remains stable even under extreme-scale deformations. Orthogonal to existing geometric weighting methods, we propose a new physics-based strategy yielding local, smooth and material-respecting weight functions. We borrow the idea of multi-weight enveloping (MWE) for animation skinning [68] and fine-tune weight functions based on a few given representative deformations. Experiments (i.e. an example is given in Fig. 3.1) show that such augmentation enhances the expressivity of the reduced model significantly even with few input poses. This elastic weighting mechanism is efficient and adaptable so that adding new quadratic DOFs at the simulation runtime is possible.

3.2 Related Work

Physics-based deformable model has been extensively studied in computer graphics. We refer readers to excellent review articles [69, 70] for a comprehensive overview of classic deformable simulation algorithms. Speeding up a deformable simulation can be achieved using dedicated numerical treatments like the multigrid method [71, 72], an incremental matrix update [73], or parallelizable nonlinear solvers [5, 74]. These methods focus on improving the performance for the fullspace nonlinear optimization without condensing simulation DOFs. On the other hand, spectral reduction methods remove less important DOFs and create a reduced or subspace

representation of fullspace DOFs (i.e. $\mathbf{u} = \mathbf{U}\mathbf{q}$). Modal analysis [58, 75, 76] and its first-order modal derivatives [59] are often considered as the most effective way for the spectral subspace construction. Yang and colleagues [61] used Krylov iteration with reduced orthogonalization to further speed up this calculation. Displacement vectors from recent fullspace simulations can also be utilized as subspace bases [77].

Earlier spectral reduction techniques compute $\mathbf{U}$ globally, which become a bit awkward when localized deformations are desired unless the user includes a large number of modal bases. As a response to this limitation, domain decomposition methods, originally designed for large-scale numerical partial differential equations (PDEs), have been imported to graphics. As subspaces are constructed at domains, local deformations can be better handled. Many existing multi-domain solvers are non-overlapping. Consequently – domains must be explicitly constrained at boundaries, which stands as a primary challenge for state-of-the-art multi-domain deformable models. Roughly speaking, domain coupling can be achieved either geometrically [1,55] by enforcing the shape continuity at the interface, or physically [2, 62] by plugging in coupling forces between adjacent domains. Recently, overlapping domain decomposition has also been explored in graphics. Xu and Barbič [78] used bounded bi-harmonics weights (BBW) to blend local modal derivative bases for localized deformations. While targeting on character skinning, it implies that overlapping domain decomposition is a feasible solution for local deformation effects. Following this direction, our method can also be considered as an overlapping domain decomposition system. Unlike [78], which geometrically blends physically-computed subspace bases, our method physically blends geometrically-constructed bases.

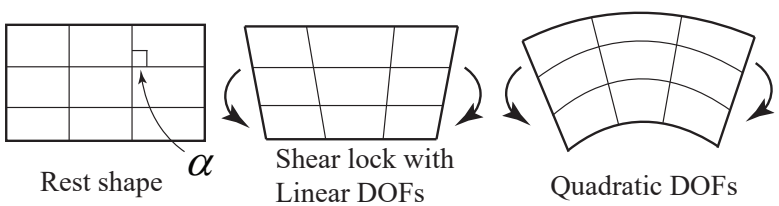
Alternatives are also possible for local deformations. For instance, Harmon and Zorin [79] made the fast simulation of contact-trigger deformations possible by adding local modal subspaces based on the Boussinesq solution. However, this method becomes less powerful when handling other types of local deformations. Teng and

colleagues [80] extended the linear condensation to handle unpredicted deformations by evoking the fullspace simulation locally.

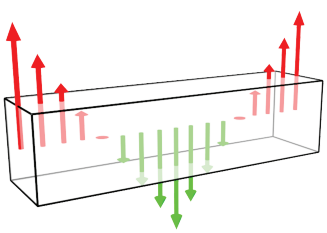
Our algorithm falls into another category of spatial reduction methods. Inspired by the superior accuracy of the higher-order finite element method [81, 82], we choose to build our deformable model based on overlapping quadratic domains, and each domain can be considered as a generalized *super element*. Our method also shares similar spirits of the shape match method [83]. Unlike shape matching however, our dynamics formulation is fully physics-based. Material parameters are fully incorporated in our reduced representation. This is achieved by encoding physically calculated shape functions, which is referred to as *elastic weighting* in this article. Calculating weight functions for shape interpolation has been widely studied in computer animation (see e.g. [84]). The harmonic coordinate [85], radial basis function (RBF) [86] and mean value coordinate (MVC) [87, 88] are a few classic paradigms. Similar techniques are also used in meshless simulations: Martin and colleagues [65] used the generalized moving least square (GMLS) for local deformation gradient evaluation. Gilles and colleagues [4] used harmonic kernels to blend rigid body motions for a skinning-like simulation.

We are not the first trying to accommodate material-awareness in the weight function calculation. Faure and colleagues [3] built shape functions using stiffness-scaled distance or the *compliance distance*. However, the other important material parameter of Poisson's ratio is disregarded. Nesme and colleagues [89] used static analysis to compute the weight function, which is similar to our approach. Yet, it is not clear how boundary conditions should be imposed. Meanwhile, it is difficult to rely on a single weight function to describe complex nonlinear deformations across the deformable body. Consequently, we calculate supplementary differential weight functions for quadratic DOFs based on few given representative deformation poses. This approach is similar to the multi-weight enveloping [68, 90].

3.3 Quadratic DOFs



Before starting a detailed discussion of our overlapping multi-domain simulator, we first show that quadratic DOFs are important in spatial reduction. Illustrated as the inset, think of simulating a simple 2D square under the *pure bending* using quadrilateral elements. Because only bending moments are applied, the angle α should be unchanged and retain right during the bending. Unfortunately, if the local subspace (i.e. shape functions of the quad-element) is linear, straight lines stay straight, and an artificial shear stress will be produced because α cannot be a right angle. More importantly, the shearing energy often increases one- or even two-order (depends on the element's geometry) faster than the real bending energy, which stiffens the deformable body. This artifact is known as the *shear locking* of linear elements. Shear locking is suppressed when the elements arrangement is dense as in most FEM based graphics simulations. However when simulation DOFs are spatially sparse (i.e. in our case), the locking issue becomes much more severe if we only have affine/linear [3] or rigid [4] DOFs.



To further illustrate this issue, we show an extreme example with side-by-side comparisons among several popular choices for local DOFs in Fig. 3.3. The beam model undergoes a pure bending test, where external forces applied are always perpendicular to its neutral axis. The force magnitude linearly varies along the neutral axis (as shown on the left). Under this circumstance, the deformable object will only have nonlinear bending deformation. This simulation is particularly challenging for linear elements. As shown in the figure, even with the correction of the invertible finite element (IFE) method [91],

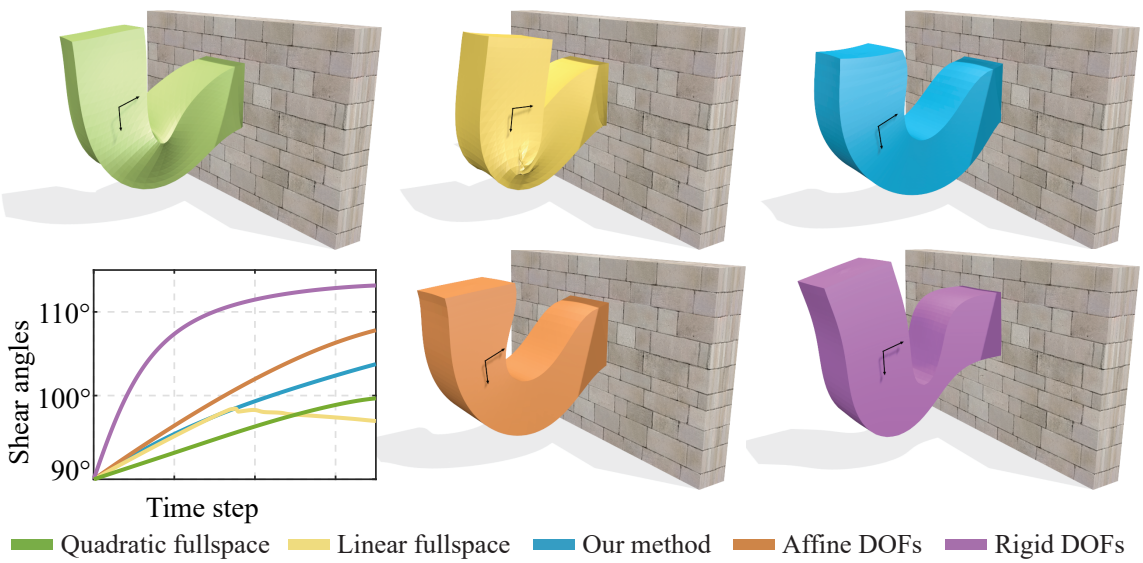


Figure 3.3: We apply pure bending moments along the neutral axis of the beam model. The bending quality is measured using the maximum shearing angle along the neutral axis. Our method yields a much smaller shearing locking artifact than other competitors including affine DOFs [3], and rigid DOFs [4]. The ground truth is the result using fullspace quadratic tetrahedral elements. Under such challenging bending test, even fullspace simulation using linear elements will fail.

the fullspace simulation using linear tetrahedral elements still fails this test. While quadratic 10-node tetrahedral elements produce a convincing ground truth result (with the cost of a much slower simulation). We evaluate the bending quality by examining the shearing angle as marked in the figure. A single quadratic domain (30 DOFs) captures the bending better than three affine domains [3] (36 DOFs) and five rigid domains [4] (30 DOFs).

3.4 Deformable Quadratic Model

We design our reduced model using overlapping quadratic domains. Each domain houses 30 DOFs grouped into 3 translation DOFs, 9 affine DOFs, 9 quadratic

homogenous DOFs, as well as 9 quadratic heterogenous DOFs. The kinematics of an individual domain is the same as in [65]. A domain only influences a local region, and the global deformation is obtained by combining contributions from multiple nearby domains.

Kinematics For a given material point P on the deformable body, we denote $\mathbf{x} = [x_1, x_2, x_3]^\top$ and $\mathbf{u} = [u_1, u_2, u_3]^\top$ as its rest shape position and displacement. A nearby domain imposes a quadratic influence to its displacement components such that $u_i = \mathbf{x}^\top \mathbf{Q}_i \mathbf{x} + \mathbf{a}_i^\top \mathbf{x} + t_i$ for $i = 1, 2, 3$. $\mathbf{Q}_i \in \mathbb{R}^{3\times 3}$ is a symmetric tensor encoding the iso-quadratic DOFs. We put its three diagonal DOFs into a vector $\mathbf{q}_{o_i} = [Q_{11}, Q_{22}, Q_{33}]^\top$ and name it as *homogenous* DOFs. Similarly, the vector $\mathbf{q}_{e_i} = [2Q_{12}, 2Q_{23}, 2Q_{13}]^\top$ containing off-diagonal entries of $\mathbf{Q}_i$ is referred to as *heterogenous* DOFs. The *affine* DOF $\mathbf{a} \in \mathbb{R}^3$ describes how u_i is linearly related to its rest position, and t_i is a *translation* DOF. Each type of deformable DOFs from different domains are convexly combined, and the ith displacement component of P can be written as:

$$u_i = \sum_j w^j \left(t_i^j + \mathbf{a}_i^{j^\top} \mathbf{x} + \mathbf{q}_{o_i}^{j^\top} \widetilde{\mathbf{x}} + \mathbf{q}_{e_i}^{j^\top} \widehat{\mathbf{x}} \right), \tag{3.1}$$

where w^j is the location-dependent weight coefficient indicating how much domain j affects the displacement of P. $\widetilde{\mathbf{x}} = [x_1^2, x_2^2, x_3^2]^\top$ and $\widehat{\mathbf{x}} = [x_1x_2, x_2x_3, x_1x_3]^\top$ are second-order homogenous and heterogenous vectors of P. By stacking all the DOFs from the jth domain into a single vector $\mathbf{q}^j \in \mathbb{R}^{30}$ such that $\mathbf{q}^j = [\mathbf{t}^{j^\top}, \mathbf{a}_1^{j^\top}, \mathbf{a}_2^{j^\top}, \mathbf{a}_3^{j^\top}, \mathbf{q}_{o_1}^{j^\top}, \mathbf{q}_{o_2}^{j^\top}, \mathbf{q}_{o_3}^{j^\top}, \mathbf{q}_{e_1}^{j^\top}, \mathbf{q}_{e_2}^{j^\top}, \mathbf{q}_{e_3}^{j^\top}]^\top$, the displacement of P can be concisely expressed as a matrix-vector product:

$$\mathbf{u} = \mathbf{G}^j \mathbf{q}^j = \left[\mathbf{G}_t^j | \mathbf{G}_a^j | \mathbf{G}_o^j | \mathbf{G}_e^j \right] \mathbf{q}^j, \tag{3.2}$$

where

$$\mathbf{G}_t^j = w^j \mathbf{I}, \quad \mathbf{G}_a^j = w^j \mathbf{I} \otimes \mathbf{x}^\top, \quad \mathbf{G}_o^j = w^j \mathbf{I} \otimes \widetilde{\mathbf{x}}^\top, \quad \mathbf{G}_e^j = w^j \mathbf{I} \otimes \widehat{\mathbf{x}}^\top.$$

We call matrix $\mathbf{G}^j$ the *geometric displacement matrix*, and the generalized coordinate

$\mathbf{q}^j$ prescribes P's kinematic configuration as:

$$\dot{\mathbf{u}} = \sum_j \mathbf{G}^j \dot{\mathbf{q}}^j, \qquad \ddot{\mathbf{u}} = \sum_j \mathbf{G}^j \ddot{\mathbf{q}}^j. \tag{3.3}$$

Reduced dynamics Let $\mathbf{e}_i$ denote canonical basis vectors of $\mathbb{R}^3$, and we drop the domain superscript $[\cdot]^j$ for succincter notations. Based on Eq.(3.1), each row of the deformation gradient tensor $\mathbf{F} = [\mathbf{F}_1, \mathbf{F}_2, \mathbf{F}_3]^\top \in \mathbb{R}^{3\times 3}$ can be written as $\mathbf{F}_i = \mathbf{F}_{t_i} + \mathbf{F}_{a_i} + \mathbf{F}_{o_i} + \mathbf{F}_{e_i} + \mathbf{e}_i$, where

$$\begin{aligned} \mathbf{F}_{t_i} &= \sum \nabla w t_i, & \mathbf{F}_{a_i} &= \sum \mathbf{a}_i^\top \mathbf{x} \nabla w + w \mathbf{a}_i^\top, \\ \mathbf{F}_{o_i} &= \sum \mathbf{q}_{o_i}^\top \widetilde{\mathbf{x}} \nabla w + w \mathbf{q}_{o_i}^\top \widetilde{\mathbf{X}}, & \mathbf{F}_{e_i} &= \sum \mathbf{q}_{e_i}^\top \widehat{\mathbf{x}} \nabla w + w \mathbf{q}_{e_i}^\top \widehat{\mathbf{X}}, \end{aligned}$$

and

$$\widetilde{\mathbf{X}} = \begin{bmatrix} x_2 & x_1 & 0 \\ 0 & x_3 & x_2 \\ x_3 & 0 & x_1 \end{bmatrix}, \qquad \widehat{\mathbf{X}} = \begin{bmatrix} 2x_1 & 0 & 0 \\ 0 & 2x_2 & 0 \\ 0 & 0 & 2x_3 \end{bmatrix}.$$

Here we assume that ∇w is a column 3-vector. On the top of $\mathbf{F}$, one can evaluate the nonlinear Green strain, $\mathbf{E} = \frac{1}{2}(\mathbf{F}^\top \mathbf{F} - \mathbf{I})$, and proceed to express the strain energy density Ψ as well as the first Piola-Kirchhoff stress tensor (PK1) based on the chosen material model. Our framework works with most hyperelastic materials, and in this chapter we choose to use the St. Venant-Kirchhoff (StVK) model since it is capable of producing most desired deformation effects for computer animation. With the StVK model, the energy density and PK1 are formulated as: $\Psi = \mu \mathbf{E} : \mathbf{E} + \frac{\lambda}{2}\mathrm{tr}^2(\mathbf{E})$ and $\mathbf{P} = \mathbf{F}[2\mu\mathbf{E} + \lambda \mathrm{tr}(\mathbf{E})\mathbf{I}]$ respectively, where λ and μ are the Lamé parameters. The per-domain reduced internal force $\widetilde{\mathbf{f}}_{int}$ and its gradient $\partial \widetilde{\mathbf{f}}_{int}/\partial \mathbf{q}$ are computed as:

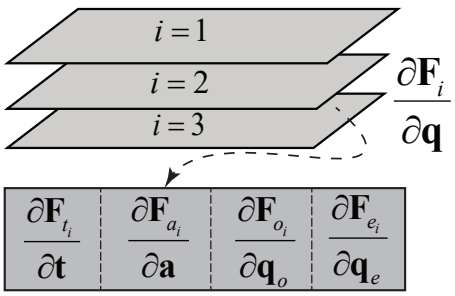
$$\widetilde{\mathbf{f}}_{int} = -\int \mathbf{P} : \frac{\partial \mathbf{F}}{\partial \mathbf{q}} \mathrm{d}V, \tag{3.4}$$

and

$$\frac{\partial \widetilde{\mathbf{f}}_{int}}{\partial \mathbf{q}} = -\int \left(\frac{\partial \mathbf{P}}{\partial \mathbf{F}} : \frac{\partial \mathbf{F}}{\partial \mathbf{q}} \right)^\top : \frac{\partial \mathbf{F}}{\partial \mathbf{q}} \mathrm{d}V. \tag{3.5}$$

Here, $\partial \mathbf{F}/\partial \mathbf{q} \in \mathbb{R}^{3\times 3\times 30}$ is a block-sparse 3-tensor, which can be

understood as the superposition of three layers as shown on the right. The ith layer represents the matrix $\partial\mathbf{F}_i/\partial\mathbf{q}$ and it hosts four sub-matrices: $\partial\mathbf{F}_{t_i}/\partial\mathbf{t}$, $\partial\mathbf{F}_{a_i}/\partial\mathbf{a}$, $\partial\mathbf{F}_{o_i}/\partial\mathbf{q}_o$ and $\partial\mathbf{F}_{e_i}/\partial\mathbf{a}_e$. These sub-matrices are block-sparse as the partial derivative is nonzero only when subscripts of generalized coordinates agree with each other. Each nonzero block can be easily calculated as:



$$\begin{aligned} \frac{\partial\mathbf{F}_{t_i}}{\partial t_i} &= \nabla w, & \frac{\partial\mathbf{F}_{a_i}}{\partial\mathbf{a}_i} &= \nabla w \otimes \mathbf{x} + w\mathbf{I}, \\ \frac{\partial\mathbf{F}_{o_i}}{\partial\mathbf{q}_{o_i}} &= \nabla w \otimes \widetilde{\mathbf{x}} + w\widetilde{\mathbf{X}}^\top, & \frac{\partial\mathbf{F}_{e_i}}{\partial\mathbf{q}_{e_i}} &= \nabla w \otimes \widehat{\mathbf{x}} + w\widehat{\mathbf{X}}^\top. \end{aligned} \tag{3.6}$$

Applying temporal discretization using the implicit Euler integration leads to the final nonlinear system to be solved at each time step:

$$(\widetilde{\mathbf{M}} - h\widetilde{\mathbf{C}} - h^2\frac{\partial\widetilde{\mathbf{f}}_{int}}{\partial\mathbf{q}})\Delta\dot{\mathbf{q}} = h\widetilde{\mathbf{f}}_{ext} + h^2\frac{\partial\widetilde{\mathbf{f}}_{int}}{\partial\mathbf{q}}\dot{\mathbf{q}}, \tag{3.7}$$

where $\widetilde{\mathbf{M}}$ is the reduced mass matrix, which can be evaluated block-wisely: $\widetilde{\mathbf{M}}^{ij} = \int \rho\mathbf{G}^{i\top}\mathbf{G}^j \mathrm{d}V$; $\widetilde{\mathbf{f}}_{ext}$ is the generalized external force; h is the time step size; and $\widetilde{\mathbf{C}}$ is the reduced damping matrix.

3.5 Physics-based Elastic Weighting

Analogous to FEM shape functions that blend nodal quantities volumetrically within an element, the weight function $w(\mathbf{x})$ interpolates local quadratic transformations to produce the final global result. An ideal weighting mechanism should be *material-customized* so that sparsely allocated DOFs well capture the nonlinear dynamics. To this end, we utilize the per-domain static equilibrium to retrieve the most physically meaningful weight distribution with carefully prescribed boundary conditions. It may be difficult to depict complex deformations with a single weight

function. To address this challenge, we use a method similar to the multi-weight enveloping [68] to customize weight distributions for quadratic DOFs using an alternating optimization. The block-sparse matrix brought by decomposed domains allows a block-Jacobi solver to update weight coefficients efficiently.

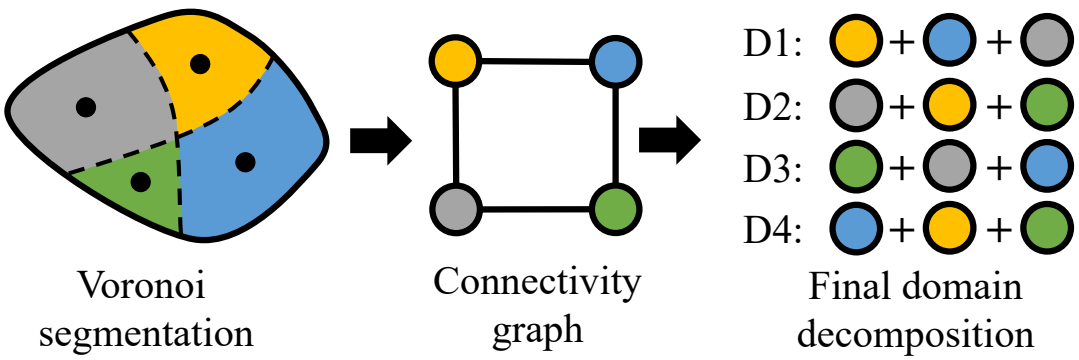


Figure 3.4: Decompose a deformable body into four domains.

Domain decomposition The input tetrahedral mesh is decomposed into overlapping domains. As illustrated in Fig. 3.4, the domain decomposition starts with subdividing the mesh into non-overlapping *segments* as in [3, 4]. While many well-established mesh segmentation algorithms are available [92], we found that a centroid Voronoi tessellation typically suffices. Initial seeds of each Voronoi cell are obtained by a regular sampling within the bounding box of the input model, followed by a few Lloyd iterations [93]. Users are allowed to manually specify segments with the provided interface too. After that, we can extract an undirected graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ encoding the connectivity information of the resulting Voronoi segmentation such that each vertex $v_i \in \mathcal{V}$ on the graph represents a Voronoi cell and $\langle v_i, v_j \rangle \in \mathcal{E}$ iff v_i and v_j share at least a triangle face. Finally, a domain is defined as a set of face-connected tetrahedrons from the ones in v_i and v_i's adjacent segments, and its seed is the seed of v_i. Note that it is possible that domains have the same collection of elements. For instance in Fig. 3.5 the red and purple, and the blue and green domains coincide with each other entirely, but they have complimentary weight functions.

Figure 3.5: The Voronoi segmentation and corresponding domain decomposition of the bunny model.

Principal direction The weight function of a domain ought to comply with the pattern describing how the deformation amplitude dissipates from its seed, where the maximum local displacement occurs. Following this thought, a reasonable way is to solve a static equilibrium [3, 89], by imposing an external nodal force $\mathbf{f}_s$ at the seed while retaining other neighbor seeds and domain's boundary. Unfortunately, this solution is ill-defined as we have infinite numbers of choices for applying $\mathbf{f}_s$ – obviously they lead to different weight distributions especially when the domain's geometry and material are irregular.

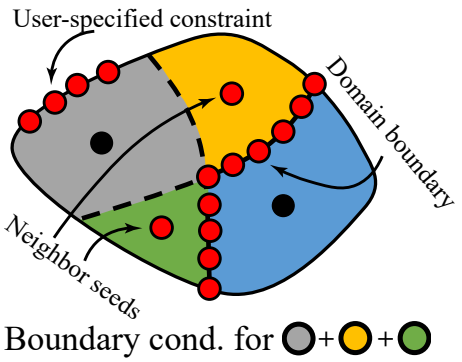
We resolve this ambiguity by restricting $\mathbf{f}_s$ along the *principal direction* $\mathbf{p}$. It can be understood as the *most deformable direction* such that domain's displacement is maximized when $\mathbf{f}_s = \mathbf{p}$. Let $[\cdot]_s$ and $[\cdot]_n$ denote domain's (three) seed DOFs and non-seed DOFs[1]. We partition domain's stiffness matrix accordingly and the principal direction of the domain can be mathematically formulated as a quadratically

[1]Seed DOFs are the x, y, and z displacement freedoms of the domain's seed node while non-seed DOFs are the DOFs of the non-seed nodes.

constrained quadratic program (QCQP) problem:

$$\begin{aligned} \arg\max_{\mathbf{p}} \quad & \|\mathbf{u}\| \\ \text{subject to} \quad & \underbrace{\begin{bmatrix} \mathbf{K}_{ss} & \mathbf{K}_{sn} & \mathbf{0} \\ \mathbf{K}_{sn}^\top & \mathbf{K}_{nn} & \mathbf{C}^\top \\ \mathbf{0} & \mathbf{C} & \mathbf{0} \end{bmatrix}}_{\mathbf{K}} \begin{bmatrix} \mathbf{u}_s \\ \mathbf{u}_n \\ \lambda \end{bmatrix} = \begin{bmatrix} \mathbf{p} \\ \mathbf{0}_n \\ \mathbf{0}_\lambda \end{bmatrix}, \\ \text{and} \quad & \|\mathbf{p}\| = 1. \end{aligned} \tag{3.8}$$

Here λ is the unknown multiplier vector. $\mathbf{C}$ is a constraint matrix prescribing necessary boundary conditions, which include: 1) user specified constraints like anchor nodes; 2) seeds of neighbor Voronoi cells; and 3) domain's boundary DOFs (as shown on the right). Doing so makes the resulting weight function always evaluate 1 at its own seed and 0 at others'. It is also local and has a vanished influence outside the domain. $\mathbf{K}$ is the domain's stiffness matrix (using linear elements).



In general, QCQP is NP-hard [94]. However as Eq. (3.8) only activates low-dimensional equality constraints, it can be efficiently solved. To do so, we first rewrite the linear constraint term in Eq. (3.8) using partitioned *compliance matrix* $\mathbf{L}$ (i.e. $\mathbf{L} \triangleq \mathbf{K}^{-1}$) as:

$$\begin{bmatrix} \mathbf{u}_s \\ \mathbf{u}_n \\ \lambda \end{bmatrix} = \begin{bmatrix} \mathbf{L}_{ss} & \mathbf{L}_{sn} & \mathbf{L}_{s\lambda} \\ \mathbf{L}_{sn}^\top & \mathbf{L}_{nn} & \mathbf{L}_{n\lambda} \\ \mathbf{L}_{s\lambda}^\top & \mathbf{L}_{n\lambda}^\top & \mathbf{L}_{\lambda\lambda} \end{bmatrix} \begin{bmatrix} \mathbf{p} \\ \mathbf{0}_n \\ \mathbf{0}_\lambda \end{bmatrix},$$

which leads to

$$\mathbf{u} = \begin{bmatrix} \mathbf{u}_s \\ \mathbf{u}_n \end{bmatrix} = \begin{bmatrix} \mathbf{L}_{ss} \\ \mathbf{L}_{sn}^\top \end{bmatrix} \mathbf{p} \triangleq \widetilde{\mathbf{L}}\mathbf{p}. \tag{3.9}$$

While evaluating the full $\mathbf{L}$ matrix is expensive, $\widetilde{\mathbf{L}}$ only has three columns and it can be quickly computed by solving:

$$\mathbf{K}\begin{bmatrix}\mathbf{L}_{ss}\\ \mathbf{L}_{sn}^{\top}\\ \mathbf{L}_{s\lambda}^{\top}\end{bmatrix}=\begin{bmatrix}\mathbf{I}_s\\ \mathbf{0}_n\\ \mathbf{0}_\lambda\end{bmatrix}. \tag{3.10}$$

Recalling that $\mathbf{KL}=\mathbf{I}$, it is easy to understand that the right hand side of Eq. (3.10) is simply the first three column of the identity matrix. After that, the target function to be maximized becomes:

$$\|\mathbf{u}\|=\sqrt{\mathbf{p}^{\top}\mathbf{B}\mathbf{p}},\qquad \mathbf{B}=\widetilde{\mathbf{L}}^{\top}\widetilde{\mathbf{L}}. \tag{3.11}$$

$\mathbf{B}$ is a symmetric positive definite (SPD) matrix and can be diagonalized with the eigenvalue decomposition as: $\mathbf{B}=\mathbf{R}^{\top}\Sigma\mathbf{R}$, where $\Sigma=\mathtt{diag}(d_1,d_2,d_3),\ d_1\le d_2\le d_3$ is the diagonal matrix of eigenvalues. $\mathbf{R}$ is an orthonormal matrix. Substituting $\mathbf{B}$ by $\mathbf{R}^{\top}\Sigma\mathbf{R}$ in Eq. (3.11) yields:

$$\|\mathbf{u}\|=\sqrt{(\mathbf{R}\mathbf{p})^{\top}\mathtt{diag}(d_1,d_2,d_3)(\mathbf{R}\mathbf{p})}\le\sqrt{d_3}. \tag{3.12}$$

It shows that $\|\mathbf{u}\|$ reaches the maximum value $\sqrt{d_3}$ when $\mathbf{p}$ is the eigenvector of $\mathbf{B}$ corresponding to its largest eigenvalue.

Principal weight & principle projection After $\mathbf{p}$ is ready, one can solve domain's static equilibrium prescribing $\mathbf{p}$ as the seed displacement and use the norm of the corresponding nodal displacement as its weight coefficient. Unfortunately, the resulting weight distribution leads to noticeable locking artifacts. Reasons are twofold. First, using the displacement norm as weight coefficients rules out the possibility of negative weight values, which are essential for high-order overlapping shape/weight functions. Second, when nodes are completely fixed, weight distributions among them are damped (as shown in Fig. 3.6 (a)) making the corresponding region artificially stiffened. The solution is simple: since the principal direction reveals the most

deformable direction of the domain, we should only consider the displacement along it other than incorporating information from "less important" directions.

Following this rationale, we allow all the constrained nodes to move on a plane perpendicular to the principal direction and only restrict their displacements along $\mathbf{p}$. The resulting per-node equilibrium displacement is also projected on $\mathbf{p}$ as the final *principal weight*. As illustrated in Fig. 3.6 (b), such principal projection is able to produce a natural and smooth weight distribution with necessary negative values across the domain. Clearly, the principal direction plays an essential role forming the principal weight function. Since different deformations propagate over the domain with different patterns, the principal direction effectively captures the most dominant one. Thus, animations produced using the principal weight are often distinguishably better. A simple test shown in Fig. 3.7 validates the importance of principal direction. In this test, the beam model only has one domain seeded at the middle. The principal direction is vertical to its neutral axis. We compare its deformation using weight functions calculated under a direction that is gradually away from the principal one (from 0° to 90° as shown in the figure). It can be clearly seen that the more it diverges from the principal direction, the more locking artifacts are observed.

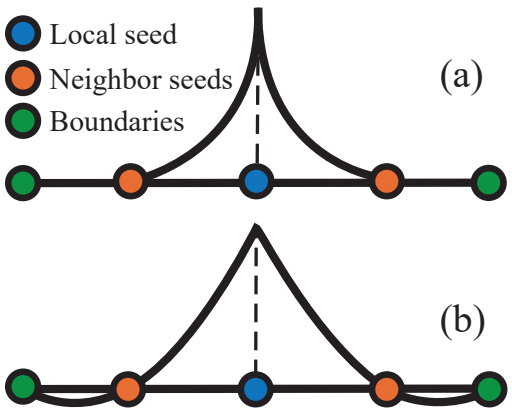


Figure 3.6: (a) Anchoring boundary nodes completely results in weight damping. (b) Principal projection yields smoother weight functions.

Elastic weighting encodes both domain's material and geometry information. Our experiment shows that the principal weight yields more realistic animations compared with geometry-based weights (e.g. harmonic coordinate [85], RBF [86] or

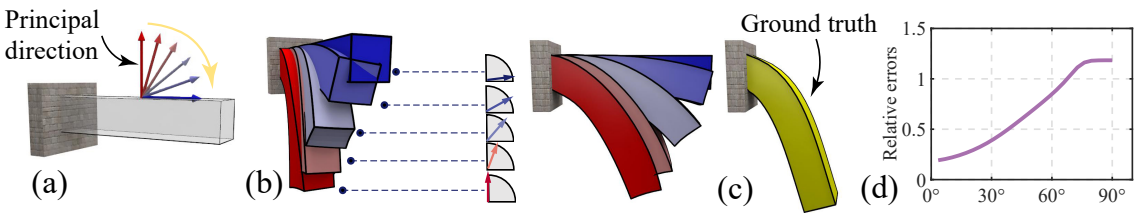


Figure 3.7: Computing weight function along different directions other than the principal direction leads to locking artifacts. (a) shows the rest shape of the beam, whose principal direction is along the y axis. We show its equilibrium shapes under the gravity using weight functions calculated with directions further and further away from the principal direction. The resulting shapes are aggregated for the comparison. The ground truth is also given in (c). In (d) we plot the relative shape difference for different directions.

MVC [87, 88]) especially when the material of the deformable body is heterogeneous (e.g. see §3.7, Fig. 3.12).

Elastic multi-weight enveloping While the principal weight function captures most visible deformations and produces natural results in general, the expressivity of our reduced model can be further enriched by using more customized weight functions at high-order DOFs, given a few representative shapes. Our method is similar to the multi-weight enveloping method [68], and we name this approach as elastic multi-weight enveloping (EMWE). Since the Cubature scheme [95] is also used for a fast runtime integration. Such shapes can be picked out of the Cubature training pose set if not specially provided.

We split domain's geometric displacement matrix $\mathbf{G}$ (i.e. Eq. (3.2)) into two sub-matrices $\mathbf{G} = [\mathbf{Y}|\mathbf{Z}]$ defined as $\mathbf{Y} = [\mathbf{G}_t|\mathbf{G}_a]$ and $\mathbf{Z} = [\mathbf{G}_o|\mathbf{G}_e]$ housing the linear and quadratic parts of $\mathbf{G}$ matrix respectively. Similarly, we subdivide domain's reduced coordinate into $\mathbf{y} = [\mathbf{t}^\top, \mathbf{a}_1^\top, \mathbf{a}_2^\top, \mathbf{a}_3^\top]^\top$ and $\mathbf{z} = [\mathbf{q}_{o_1}^\top, \mathbf{q}_{o_2}^\top, \mathbf{q}_{o_3}^\top, \mathbf{q}_{e_1}^\top, \mathbf{q}_{e_2}^\top, \mathbf{q}_{e_3}^\top]^\top$ so that $\mathbf{u} = \mathbf{G}\mathbf{q} = \mathbf{Y}\mathbf{y} + \mathbf{Z}\mathbf{z}$.

$\mathbf{Y}$ matrix is constructed using the principal weight function discussed previously.

For a given exemplar shape $\mathbf{u}_k$, we compute its residual error vector as:

$$\Delta\mathbf{u}_k \triangleq \left(\mathbf{I} - \mathbf{Y}(\mathbf{Y}^\top\mathbf{Y})^{-1}\mathbf{Y}^\top\right)\mathbf{u}_k, \tag{3.13}$$

which is a difference vector between the shape $\mathbf{u}_k$ and its best-fitting reduced representation in the column space of $\mathbf{Y}$ (i.e. $(\mathbf{Y}\mathbf{Y}^\top)^{-1}\mathbf{Y}^\top\mathbf{u}_k$). Our goal is to minimize $\|\Delta\mathbf{u}_k - \mathbf{Z}\mathbf{z}\|$ by assigning each quadratic DOF an independent isotropic weight function so that $\mathbf{u}_k$ can be well expressed in the subspace. Mathematically, this reflects an updated formulation for $\mathbf{G}_o$ and $\mathbf{G}_e$:

$$\mathbf{G}_o = \mathbf{I} \otimes \left(\mathbf{w}_o^\top \mathtt{diag}(\widetilde{\mathbf{x}})\right) \qquad \mathbf{G}_e = \mathbf{I} \otimes \left(\mathbf{w}_e^\top \mathtt{diag}(\widehat{\mathbf{x}})\right), \tag{3.14}$$

where $\mathbf{w}_o, \mathbf{w}_e \in \mathbb{R}^3$ are weight coefficients for homogenous (x_1^2, x_2^2, x_3^2) and heterogenous (x_1x_2, x_2x_3, x_1x_3) quadratic DOFs. We split $\mathbf{Z}\mathbf{z}$ into homogenous and heterogenous parts as $\mathbf{Z}\mathbf{z} = \mathbf{G}_o\mathbf{z}_o + \mathbf{G}_e\mathbf{z}_e$. A few manipulations extract the homogenous weight vector as:

$$\begin{aligned}
\mathbf{G}_o\mathbf{z}_o &= \left[\mathbf{I} \otimes (\mathbf{w}_o^\top \mathtt{diag}(\widetilde{\mathbf{x}}))\right]\mathbf{z}_o \\
&= (\mathbf{I} \otimes \widetilde{\mathbf{x}}^\top)(\mathbf{I} \otimes \mathtt{diag}(\mathbf{w}_o))\mathbf{z}_o \\
&= \underbrace{(\mathbf{I} \otimes \widetilde{\mathbf{x}}^\top)\left[\mathtt{diag}(\mathbf{q}_{o_1})|\mathtt{diag}(\mathbf{q}_{o_2})|\mathtt{diag}(\mathbf{q}_{o_3})\right]^\top}_{\mathbf{W}_o}\mathbf{w}_o.
\end{aligned} \tag{3.15}$$

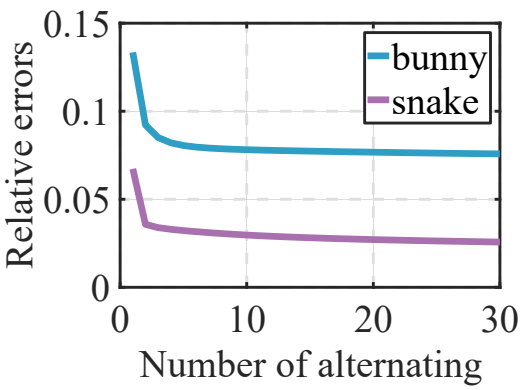
Together with $\mathbf{W}_e = (\mathbf{I} \otimes \widehat{\mathbf{x}}^\top)\left[\mathtt{diag}(\mathbf{q}_{e_1})|\mathtt{diag}(\mathbf{q}_{e_2})|\mathtt{diag}(\mathbf{q}_{e_3})\right]^\top$, we construct the matrix $\mathbf{W} = [\mathbf{W}_o|\mathbf{W}_e]$ such that $\mathbf{Z}\mathbf{z} = \mathbf{W}\mathbf{w}$, where $\mathbf{w} = [\mathbf{w}_o^\top, \mathbf{w}_e^\top]^\top$ is the quadratic weight vector. Clearly, both $\mathbf{z}$ and $\mathbf{w}$ are unknown and final weight coefficients should be calculated alternatingly. We initialize $\mathbf{w}$ as the principal weight, fix it, and compute the current optimal $\mathbf{z}$ using least square as: $\mathbf{z} \leftarrow (\mathbf{Z}^\top\mathbf{Z})^{-1}\mathbf{Z}^\top\Delta\mathbf{u}_k$. Afterwards, $\mathbf{z}$ is fixed, and we compute the optimal $\mathbf{w}$ respecting the updated $\mathbf{z}$. The iteration stops when $\|\Delta\mathbf{u}_k - \mathbf{W}\mathbf{w}\|$ converges.

To avoid irregular weight distributions, we also added a penalty term when solving $\mathbf{w}$. Let $\mathbf{L} \in \mathbb{R}^{6N\times 6N}$ be a graph-Laplacian matrix computing the weight difference

between a node and its local average. The augmented optimization for $\mathbf{w}$ becomes:

$$\arg\min_{\mathbf{w}} \|\Delta\mathbf{u}_k - \mathbf{W}\mathbf{w}\| + \alpha\|\mathbf{L}\mathbf{w}\|, \tag{3.16}$$

which leads to the final weight update as $\mathbf{w} \leftarrow (\alpha\mathbf{L}^\top\mathbf{L} + \mathbf{W}^\top\mathbf{W})^{-1}\mathbf{W}^\top\Delta\mathbf{u}_k$. Here, we set $\alpha = 0.1$ in all of our experiments. $\alpha\mathbf{L}^\top\mathbf{L} + \mathbf{W}^\top\mathbf{W} \in \mathbb{R}^{6N\times 6N}$ is a big matrix and explicitly factorizing it is expensive. Fortunately, it is also block dominant since $\mathbf{W}^\top\mathbf{W}$ is block diagonal. As a result, we use the iterative block-Jacobi solver to solve $\mathbf{w}$ efficiently. For instance for the bunny model, block-Jacobi can complete one weight update within tens of milliseconds while the `Pardiso` solver takes several seconds. As shown on the left, few (3 to 5 iterations) alternations are sufficient to produce good quadratic weights.



EMWE enriches the expressivity of the geometric displacement matrix and allows interesting deformable effects that could be challenging for exiting methods with similar numbers of simulation DOFs. Fig. 3.8 reports snapshots from a set of comparative simulations of a winding snake model. A circular force field is applied and the fullspace simulation with 10,800 DOFs winds the snake for about 800° (i.e. 360° + 360° + 180°) as shown in the first row in the figure. Applying the principal weight for all the 30 DOFs only yields a 360-degree wind (second row in the figure). This result is similar to what one could obtain using *modal derivatives* [59] with 30 modal bases. However, EMWE using only three poses is able to improve the resulting animation making it visually similar to the fullspace result (third row in the figure). This result is even more plausible than modal derivatives with 60 bases (forth row in the figure). Notice that training poses used are quite different from the final frame of the fullspace simulation. Indeed, these poses simply imply that larger weights should be assigned to quadratic DOFs at the middle part of the snake. The entire EMWE

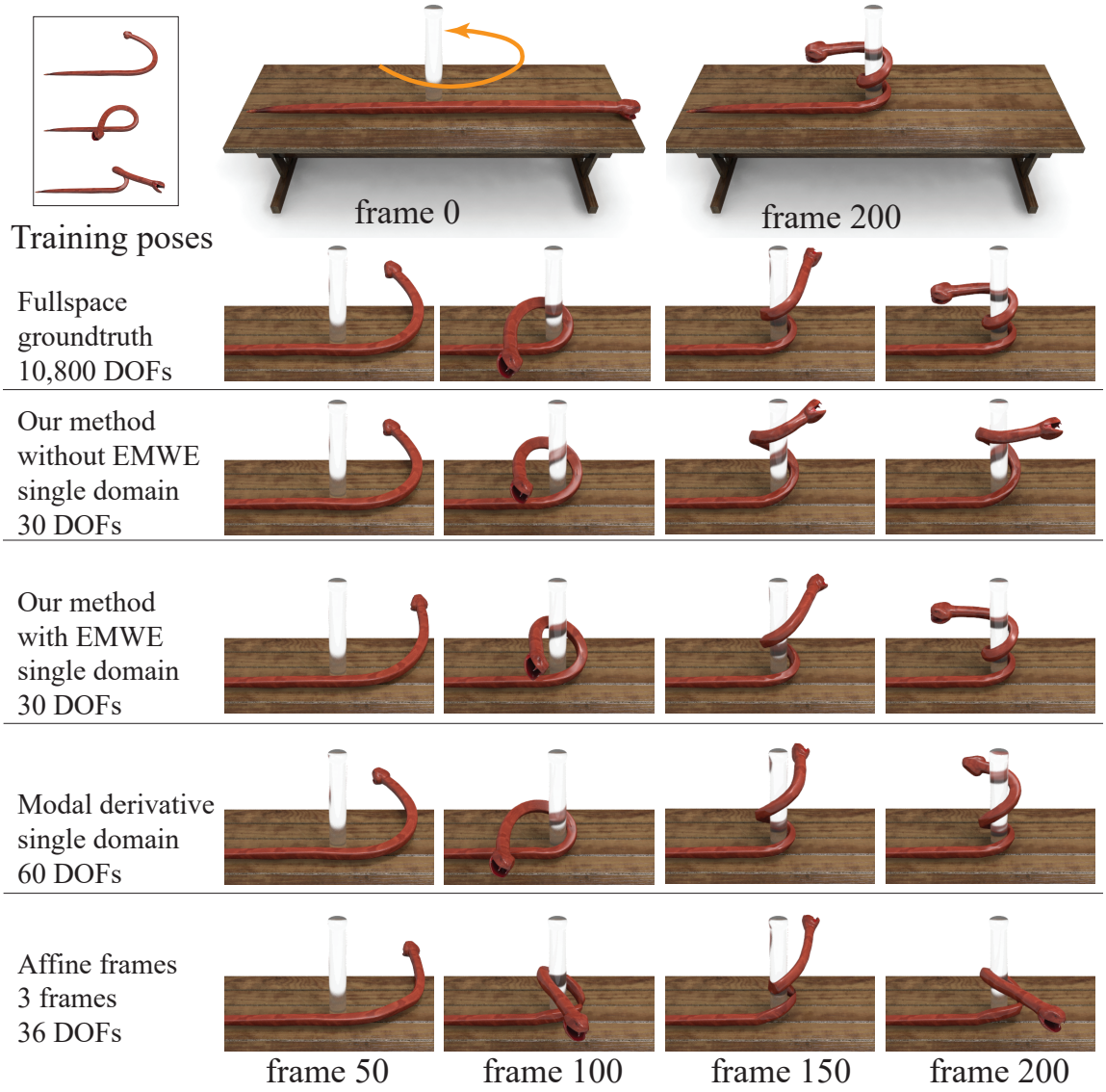


Figure 3.8: Comparative simulation of a winding snake.

training takes less than 300 ms. Results using multiple domains but with only affine transformations as in [3] are also reported in the bottom row. Clearly our method outperforms the spatial reduction using linear DOFs.

3.6 Adaptability and Extensibility

Assembling $\mathbf{G}$ only needs to solve domain's rest-shape stiffness matrix (i.e. for handling Eq. (3.10) and computing the principal weight), which is efficient and allows an interactive DOF adaption during the simulation runtime to incorporate novel deformations, for example induced by collisions. Besides, the locality of domain's subspace also makes the Cubature training orders of magnitude faster and parallelizable at each Voronoi segment. Overlapping domains do not need an explicit domain coupling treatment. Therefore, our method is able to simulate extreme-scale deformations robustly even when the mesh geometry is degenerated.

Runtime domain adaption It is known that geometrically constructed shape functions can be conveniently altered and adapted at the run time to accommodate new deformations [3, 96]. Our elastic weighting function also possesses this property. Suppose a novel deformation is triggered by a local contact on the deformable body. A reasonable reaction is to add a new domain $\mathcal{D}_a$ seeded at the node where the deepest inter-penetration is found. Due to the presence of $\mathcal{D}_a$, weight functions of existing domains that overlap with $\mathcal{D}_a$ need to be updated.

Consider a 1D example shown in Fig. 3.9. The original weight function of an existing domain $\mathcal{D}$ seeded at S, as well as the newly-plugged domains $\mathcal{D}_a$ seeded at S_a are known. The weight interpolating property requires that the updated weight w' of $\mathcal{D}$

must have vanished values at both B (the original domain boundary) and S_a while remaining 1 at its own seed S. In other words, we seek for a smooth function to offset w such that it becomes 0 at S_a while its original values at S and B are unchanged. Interestingly, w_a serves this purpose perfectly as it evaluates 0 at both S and B so that stacking w_a over w will not change w's original boundary conditions. As a result, the updated weight function of $\mathcal{D}$, after $\mathcal{D}_a$ is inserted, can be instantly

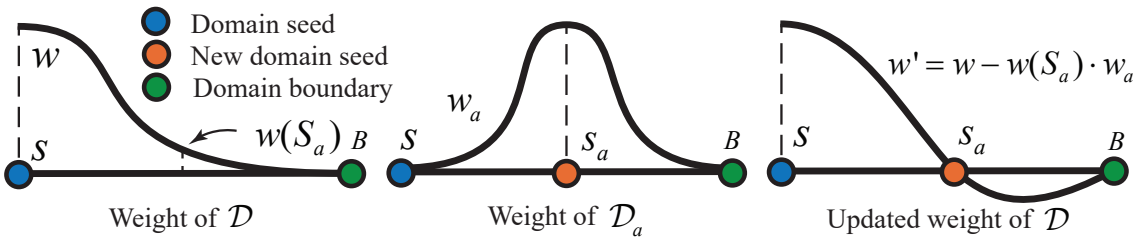


Figure 3.9: Left: the initial weight distribution w of an existing domain seeded at S. Mid: the weighting function of a newly inserted domain w_a. Right: the updated weighting function w' can be fast obtained as the linear combination of w and w_a.

obtained without resorting to the re-computation from scratch as:

$$w' = w - w(S_a) \cdot w_a. \tag{3.17}$$

It is noteworthy that such combination of weight functions also agrees with the superposition principle of linear elasticity. If principal directions of $\mathcal{D}$ and $\mathcal{D}_a$ align each other, it can be shown that the updated weight distribution w' is identical to the fresh-calculated elastic weight, under new boundary conditions. Eq. (3.17) also implies that the new weight from $\mathcal{D}_a$ *supplements* existing subspaces rather than *replacing* them. In the example shown in Fig. 3.10, a concentrated external force is applied at the facet center of a rubber brick. Inserting a new domain correspondingly yields a natural denting effect. The updated weight functions on the surface are also plotted.

Parallelized local Cubature An efficient integration to compute the reduced internal force and its gradient is important for interactive deformable models. Barbič and James [59] found that entries of $\widetilde{\mathbf{f}}$ and $\widetilde{\mathbf{K}}$ are low-degree polynomials of the reduced displacement for StVK materials, whose coefficients can be pre-computed. Another more general solution named Cubature [95] uses 3D quadrature to approximate the internal force at a few Cubature elements. Cubature was originally adopted for model reduction using global bases, and we notice that this procedure can be significantly accelerated under our framework due to the locality of the per-domain subspace.

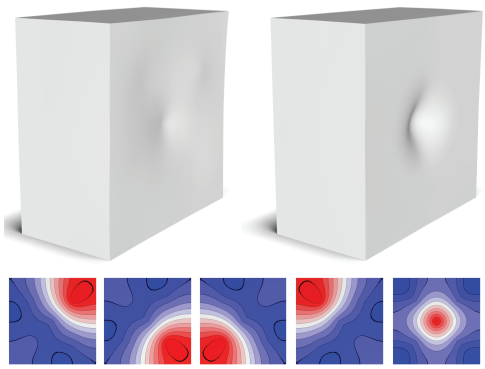


Figure 3.10: Adding new domain produces natural denting effects on the brick.

It is clear that (i.e. in Fig. 3.4) elements in the same Voronoi segment are affected by the same subset of reduced DOFs from adjacent domains. As a result, the Cubature training can be carried out segment by segment. Such local training is independent and can be trivially accelerated with multi-threading. More importantly because a segment has much fewer (typically less than 20) Cubature points, the associated NNLS (non-negative least square) solves run much faster than the global Cubature training. For instance, training the bunny model would take more than three hours with global bases, which is only less than two minutes when domain-decomposed. After the Cubature training is finished, the runtime evaluation of force and force gradient is simplified as: $\widetilde{\mathbf{f}}_{int} \approx \sum_l \eta_l \mathbf{P}_l \frac{\partial \mathbf{F}_l}{\partial \mathbf{q}}$ and $\widetilde{\mathbf{K}} \approx \sum_l \eta_l (\frac{\partial \mathbf{F}_l}{\partial \mathbf{q}})^\top (\frac{\partial \mathbf{P}_l}{\partial \mathbf{F}_l})^\top \frac{\partial \mathbf{F}_l}{\partial \mathbf{q}}$, where η_l is the non-negative Cubature weight at the element l.

Recovering domain degeneration Some materials such as the StVK model suffer the stability issue under a large compression. This is because the constitutive law does not produce necessary resisting forces to restore the volume from degeneration. This issue is often invisible for spectral reduction methods as the high-frequency displacements are already filtered by the subspace. Unfortunately, we do not have any mechanism preventing a domain from inversion. To deal with the domain degeneration, we transplant the invertible finite element or IFE method [91, 97] into our framework. IFE alters singular values of $\mathbf{F}$ if they are smaller than a certain threshold so that an element always produces restoring internal forces. Doing so modifies the differential relation between PK1 and the deformation gradient. We follow the formulation in [98] to update $\partial \mathbf{P}/\partial \mathbf{F}$. While $\mathbf{F}$ is linearly related to the reduced coordinate, clamping its singular values does not alter this relation. Therefore, $\partial \mathbf{F}/\partial \mathbf{q}$ remains unchanged.

Figure 3.11: Use reduced IFE simulation to imitate the inflation of a hot-air balloon. In this example, we simply use Cubature points as restorative elements. Three domains are defined.

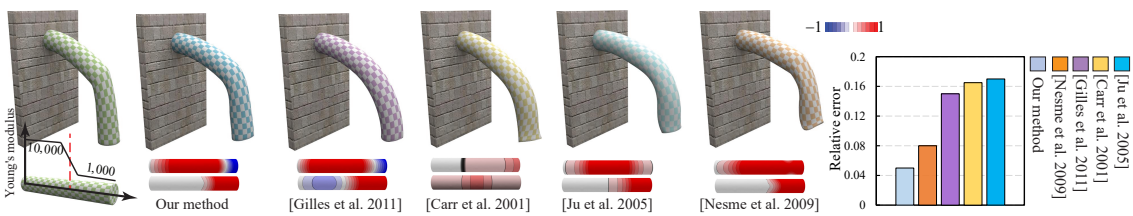


Figure 3.12: Compare different weighting algorithms with a cylinder beam of heterogenous materials: the Young's modulus of the beam varies along its neutral axis. Two quadratic domains are set. The weight distributions from different algorithms are also plotted.

In general, IFE is slow because the deformation gradient at each element must be checked and adjusted if necessary. This leads to $O(N)$ runtime efforts, and an interactive IFE simulation is hardly possible for large meshes, where N stands for the total number of elements on the mesh. We note that such calculation could also be accelerated following the idea of numerical quadrature, which has also been successfully utilized to resolve self-contacts [99] recently. We follow the standard Cubature procedure to find a small number of *restorative elements* $\mathcal{R}$ so that the

domain-wise restoring force can be well approximated at restorative elements based on a set of degenerated training poses. The runtime corrective forces resolving the volume inversion can then be computed via:

$$\widetilde{\mathbf{f}}_r = \sum_{i \in \mathcal{R}} \upsilon_i \mathbf{G}_i^\top \left[\mathbf{f}_i(\widehat{\mathbf{F}}_i) - \mathbf{f}_i(\mathbf{F}_i) \right], \tag{3.18}$$

where $\mathbf{f}(\mathbf{F}_i)$ and $\mathbf{f}(\widehat{\mathbf{F}}_i)$ are element internal forces calculated using the current deformation gradient $\mathbf{F}_i$ and the singular-value-corrected deformation gradient $\widehat{\mathbf{F}}_i$. υ_i is the non-negative weight. The approximation of Eq. (3.18) does not have to be precise, because the restoring force itself is ad-hoc when a material failure occurs. $\widetilde{\mathbf{f}}_r$ only provides a momentum to recover the degenerated volume, and it will be replaced by regular Cubature forces as long as the degeneration is resolved. In practice, the reduced IFE produces physically-plausible responses under extreme deformations and it is at least an order faster than the fullspace IFE method. Fig. 3.11 shows an example of using our reduced IFE to imitate inflating a collapsed hot-air balloon.

3.7 Experimental Results

Our system was implemented using `Visual Studio 2013` on a `Windows 10 X64` desktop PC equipped with an `Intel Core i7-5960` CPU and 12G RAM. Numerical algorithms were implemented on the top of `Eigen C++` template and `Intel MKL` library. Unless specified, the performance reported is with the single-core implementation. The statistics of tested 3D models and simulation benchmarks are summarized in Tab. 3.1. Since our method uses the model reduction, all the experiments run at an interactive rate, which is two to three orders faster than the fullspace simulation. Here we would like to remind readers that the EMWE is only used for funnelling the bunny (Figs. 3.1 and 3.16) and winding the snake (Fig. 3.8) with six and three training poses. All the other experiments discussed are based on the principal weight

Model	**#Ele**	**#D**	**#Cub**	**Pre**	**Sim**	**FPS**
Bunny	64K	5	389	2(4.2%)	26	16
Balloon	56K	3	248	1(3%)	55	25
Stay-Puft	49K	5	138	0.5(5%)	23	13
Cactus	23K	4	102	0.2(4.5%)	175	63
Armadillo	39K	6	348	1.5(7%)	31	23

Table 3.1: Model statistics and simulation benchmarks. **#Ele**: the number of elements on the simulation mesh; **#D**: the number of initialized quadratic domains; **#Cub**: the number of cubature elements in total; **Pre**: pre-computation time in minutes (with multi-threading enabled) and the accuracy of the Cubature training; **Sim**: average FPS for simulating the deformable bodies (collision handling not included). **FPS**: over all FPS including collision/self-collision handling and OpenGL rendering.

and local adaption. After all, it is not surprising that EMWE can produce highly stylized animations similar to [100] with carefully selected weight training, which makes the comparison unfair.

Weighting quality In the first experiment, we compare the proposed elastic weighting with other widely-used weighting algorithms including Harmonics weight [4], radial basis function (RBF) [86], mean value coordinate (MVC) [88] and the method used in [89]. As shown in Fig. 3.3, quadratic DOFs produce better nonlinear bending than linear or rigid ones. Therefore, we use two quadratic domains (i.e. 60 simulation DOFs) for all the tests in order to eliminate the interference brought by using different simulation DOFs.

The result is summarized in Fig. 3.12. Here, the material distribution of the cylinder beam is not uniform. The fixed end of the cylinder beam is stiffer where the Young's modulus is set as $10,000$. The stiffness linearly decreases to $1,000$ at the middle region. The deformed shapes under the gravity are shown along with the corresponding weighting distributions. One can see that geometrically constructed

weight functions (Harmonics, RBF and MVC) yield results that do not reflect the material variation. Our method with principal weights well handles such heterogenous elastic object. In [89], the weight function is computed based on a high-resolution equilibrium analysis which also takes the material into account. Similar results can be obtained using *compliance distance* [3], which augments the weight function with Young's modulus. However without computing the principal direction and performing the principal projection, the locking artifact (near the free end of the beam) is discernible. The fullspace ground truth (using un-reduced FEM simulation of linear elements) is the leftmost and the relative error of the free end displacement is also plotted.

One may also notice from Fig. 3.12 that our weighting function will have negative values occasionally. In fact, in the context of overlapping domain decomposition having negative weighting coefficient is essential to avoid the locking artifact. To explain this argument, let us look at an illustrative toy example of a 1D element with two nodes A and B. Under this configuration, only *interpolation* is needed as shown in the leftmost subfigure of Fig. 3.13. Here, interpolation means A's weighting function W_A is defined within its nearest boundary condition: $W_A(B) = 0$, and negative weight should be avoided. When a new node C is inserted into this 1D element between A and B[2], it induces a new boundary condition: $W_A(C) = 0$. That also means W_A need to be *extrapolated* beyond its nearest boundary condition in order to allow A to influence the entire element. If one chooses to design a smooth shape function, in order to incorporate boundary conditions at B and C, the lowest-degree polynomial solution is a quadratic curve with negative values after C (the rightmost subfigure in Fig. 3.13). If one chooses to clamp the functions values as the bounded biharmonic weights (BBW) [101], the weighting function becomes discontinuous and locking artifacts could occur (mid sugfigure in Fig. 3.13). In this case, the element is degenerated to be a linear one. Without negative weight values, no smooth shape

[2]Doing so actually makes this element nonlinear.

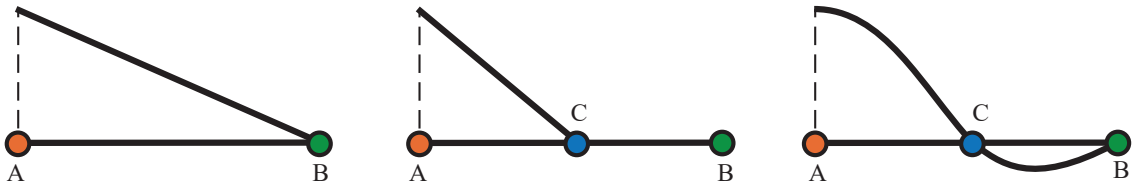


Figure 3.13: Shape functions of a simple 1D element.

functions can satisfy both boundary conditions at B and C simultaneous.

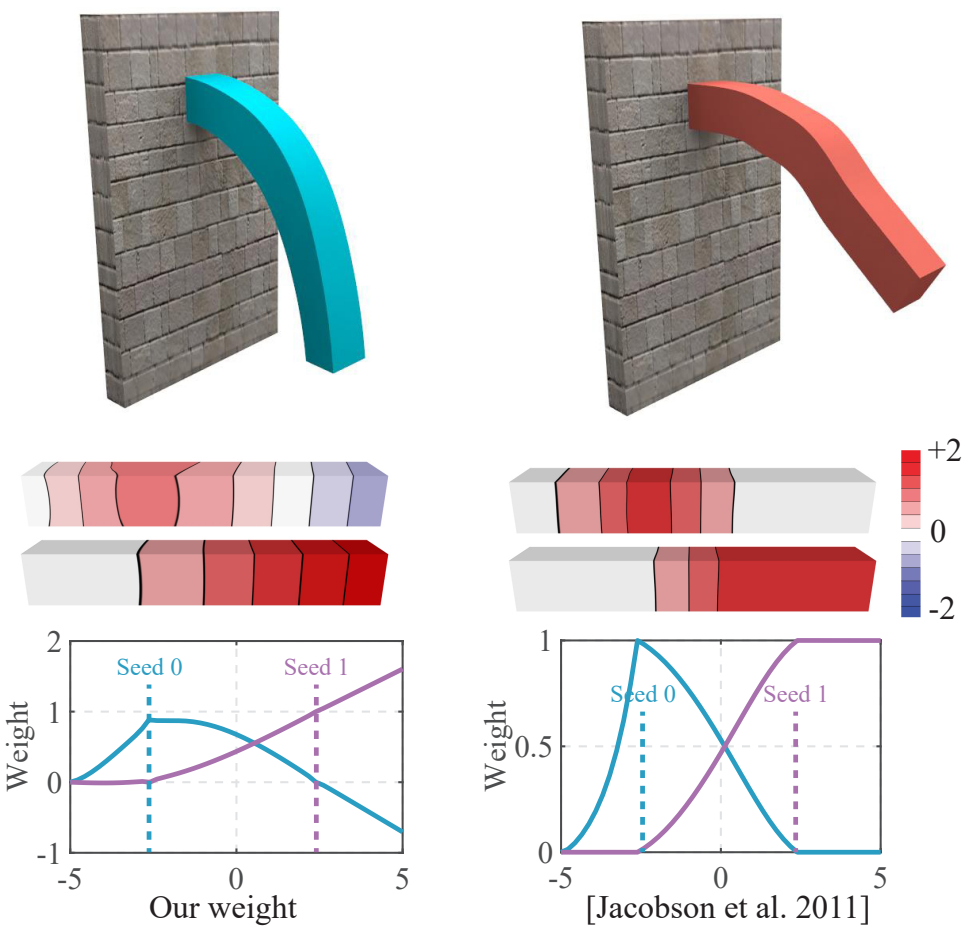


Figure 3.14: BBW may induce locking artifacts.

Whether or not we should have negative weighting functions depends on whether

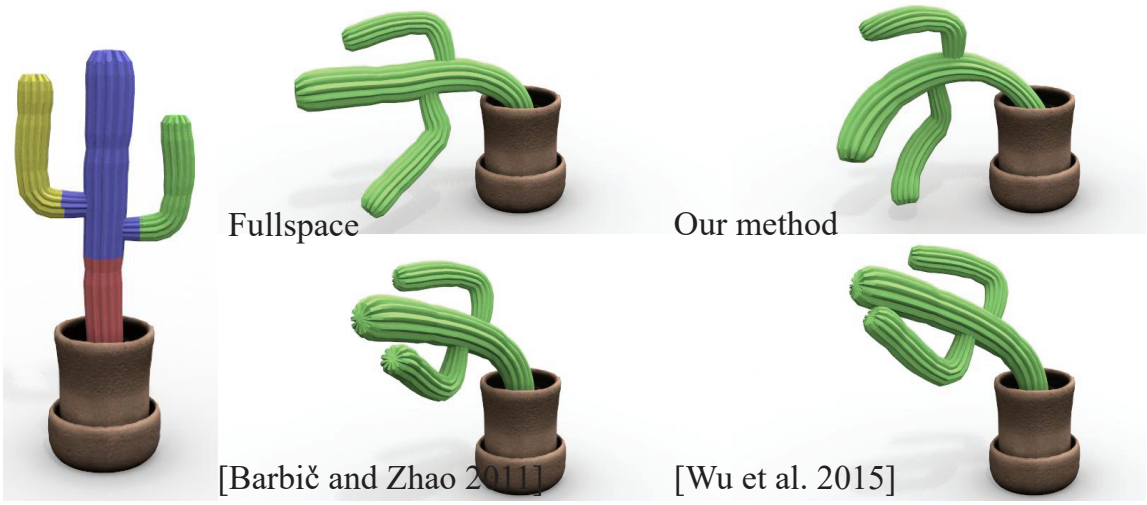


Figure 3.15: Simulate a swinging cactus using deformation substructuring [1], unified domain decomposition [2], our method, and the fullspace solver.

or not the weighting function needs to be extrapolated beyond its nearest boundary conditions. In many existing graphics literature, weighting functions are not supposed to influence areas outside of its neighboring boundary conditions. As a result, such blending is just interpolation and should be convex. However, our system is designed for the overlapping domain decomposition and negative weights become essential. Fig. 3.14 gives a 3D example. The beam model has two domains both covering the entire model. Their seeds are at the 1/4 and 3/4 along the neutral axis of the beam. If BBW is used, weight values of both domains at nodes right to seed 1 will be 1 and 0. This leads to locking effect. However, our method does not have such problem.

Robust nonlinear expressivity Next, we evaluate the capability of the proposed simulator capturing large nonlinear deformations. We compare our method with two paradigmatic state-of-the-art multi-domain nonlinear simulators using deformation substructuring [1] and coupling elements [2]. As shown in Fig. 3.15, the cactus model is decomposed into four domains. The Voronoi segments (left in the figure) are used for the non-overlapping domain decomposition for [1] and [2] with 30 modal

derivatives per domain. Therefore all the reduced models have 120 simulation DOFs. From snapshots reported in the figure and the supplementary video, we can see that all the simulators produce plausible deformable animations comparable to the fullspace result.

On the other hand, our method does not require an explicit domain coupling. This advantage makes our system robust against large-scale deformations. Figs. 3.1 and 3.16 show snapshots of a challenging scenario: a one-inch-tall bunny model is forced to pass through a thin funnel. The Young's modulus of the bunny is 500 and the Poisson's ratio is 0.4. Five domains are used in this example and all the solvers use 150 simulation DOFs. When the funnel is relatively wide (i.e. 0.4) as shown in Fig. 3.16 top, all the simulators produce plausible and interesting animations. However, if we reduce the size of the funnel to 0.3, non-overlapping solvers fail. This is because when domains' interfaces are highly distorted, the rigid interface assumption [1] does not hold and the coupling elements [2] are degenerated. Our method is still able to produce a similar animation compared with the fullspace simulation (Fig. 3.1) and remains stable even the funnel is further shrunk to 0.2 (Fig. 3.16 bottom).

A more extreme case highlighting the robustness of our solver is shown in Fig. 3.17. In this test, we collapse the Armadillo model into a small 2D disk initially. When this strong geometry constraint is released, our method quickly restores the model back to the rest shape with the help of reduced IFE simulation. While the IFE contributes the calculation of necessary internal forces, the main reason behind such good numerical stability is the overlapping domain decomposition. A fullspace IFE [91] simulated animation is also available in the video for readers' reference.

Local adaptivity Lastly, we test the adaptivity of our algorithm. Fig. 3.18 reports results using our method, local subspace [79] and the fullspace solver when we push the Stay-Puft with a spiky board. The Stay-Puft model originally has five domains and extra two domains are inserted corresponding to the external collision

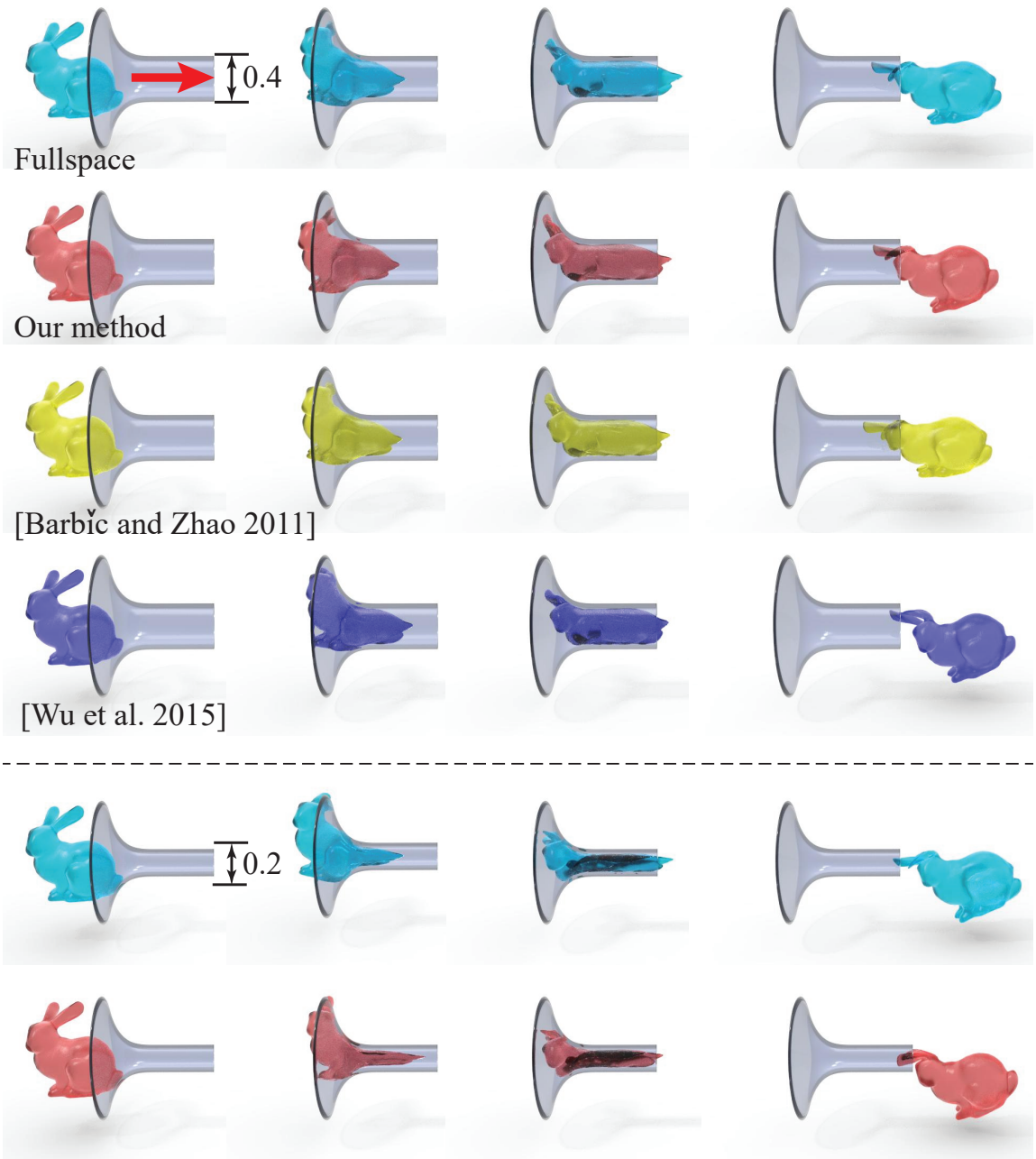


Figure 3.16: Drag the bunny through a thin funnel (available as the supplementary executable too).

with spikes. We can see from the figure that, newly-added domains provide necessary deformable freedoms to simulate local deformation, and realistic results comparative

Figure 3.17: Our solver remains stable under severe geometry constraints.

to the fullspace ground truth are produced.

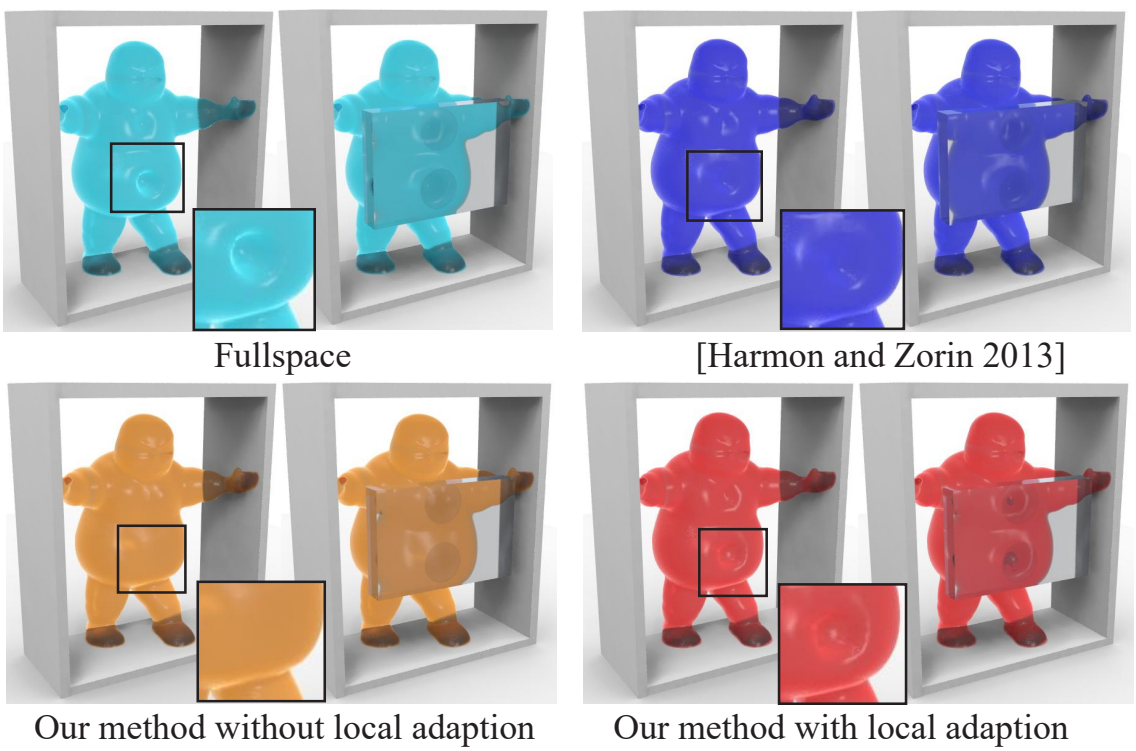


Figure 3.18: Adding new domains according to the contacts from a moving spike board greatly enriches detailed local deformation.

It is noteworthy that similar denting effects can also be obtained by building a local subspace using Boussinesq equation [79] as shown in Fig. 3.18. However, our method is able to deal with a much wider range types of deformation. As shown

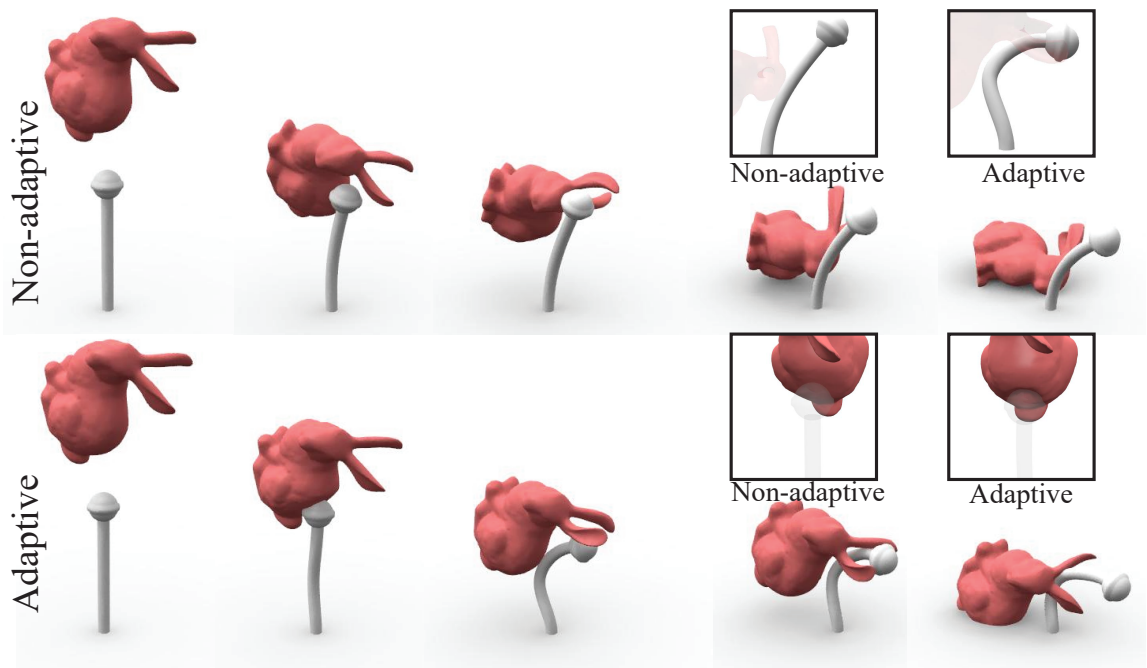


Figure 3.19: Runtime domain adaption does not only yield better denting effects on the bunny, but also enriches local deformations at the lollipop.

in Fig. 3.19, the falling bunny hits an elastic lollipop. The local domain insertion does not only help for a better denting effect on the bunny's body, it also enriches the local deformation for the lollipop. Initially, the bunny has five domains and the lollipop has only one domain seeded at the middle.

Chapter 4

Complex Step Finite Difference for Solid Dynamics

4.1 Introduction

In this chapter, we will deal with another essential computing task in physics-based simulation, the evaluation of various derivatives like total derivative, partial derivative, directional derivative, second- or high-order derivatives, etc. often stands out as a significant technical or implementation obstacle. Normally, people incline to infer an exact formula of derivative functions, which gives the best efficiency and accuracy. However, there are also many situations where a closed-form expression of the target function is not available, or deriving its actual derivative is too involved for just performing preliminary proof-of-concept trials. The numerical derivative is then preferred.

The commonly used strategy for numerical derivative is the finite difference

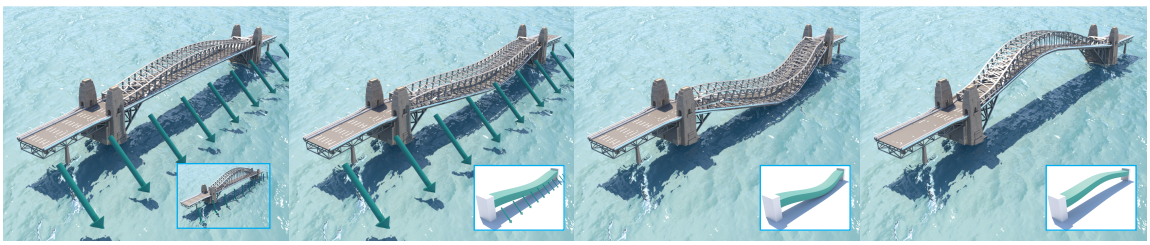


Figure 4.1: We present an accelerated complex-step finite difference algorithm, which efficiently computes highly accurate numerical derivative. This method can be coupled with the Cauchy-Riemann formula to allow us to fully exploit existing (real-valued) linear algebra libraries to evaluate derivatives of tensor-valued functions. This figure reports an example of designing vibration frequency of a bridge model ($21,414$ elements) by changing its geometry. The target frequency is visualized on a rectangular beam. Given an external force field, the bridge oscillates under the same frequency as the beam model does.

method. For instance, the *forward difference* scheme estimates the derivative as:

$$f'(x_0) = \lim_{\Delta x \to 0} \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x} \approx \frac{f(x_0 + h) - f(x_0)}{h}, \tag{4.1}$$

where a small perturbation $h \in \mathbb{R}$ is used to approximate $\lim_{\Delta x \to 0}(\cdot)$. It appears that the smaller h is, the better approximate Eq. (4.1) delivers. However, we are not allowed to make h arbitrarily small to improve the precision of Eq. (4.1). This is because the subtraction between two nearly equal numbers, such as $f(x_0 + h) - f(x_0)$ in Eq. (4.1) when h is very small, could eliminate many of their significant digits and contaminate the result. This issue is often known as the *subtractive cancellation*. During the simulation, finite difference would accumulate numerical errors along the time integration and crash the solver quickly.

Fortunately, this numerical instability can be averted using the so-called *complex-step finite difference* [102–104] or CSFD. The trick is to apply the perturbation h in the imaginary domain after promoting f to be a complex function. The subtraction of the first-order terms is skipped in the complex Taylor expansion [105], and we can make the perturbation h very small to accurately approximate the derivative without

worrying about the subtractive cancellation. CSFD allows us to conveniently obtain a highly accurate numerical derivative without deriving its actual formulation, which could be otherwise tedious and error-prone.

On the downside, CSFD has several fundamental limitations. First of all, promoting a real-valued function to be a complex-valued one induces significant computation overheads. A naïve CSFD implementation as in existing literature [102, 103] is often orders-of-magnitude slower than the analytic derivative. Secondly, complex-version Taylor expansion only circumvents the subtractive cancellation of the first-order derivative. Second- and higher-order derivatives still suffer with this issue and cannot be robustly approximated. In many simulation problems, we also need to deal with tensor functions, whose outputs are based on complicated numerical routines like Cholesky decomposition or SVD. Original CSFD becomes awkward as those numerical procedures are difficult to be explicitly formulated. Promoting such tensor functions is rather involving, if not impossible. As an echo to those drawbacks, we augment classic CSFD scheme making it more efficient, generalizable, and robust. While our extensions utilize some known techniques like multicomplex number [106, 107] and Cauchy–Riemann equation [108], to the best of our knowledge, we are the first to optimize CSFD performance making it nearly as efficient as the analytic derivative, re-engineer it to be a handy off-the-shelf numerical differentiation solution for physics animation, and thoroughly validate its feasibility in the context of deformable simulation. Specifically, we would like to summarize our contributions as follows:

- **Analysis** CSFD is a relatively new numerical method. We provide an extensive explanation of its numerical mechanism, error source, and theoretical foundation.
- **Acceleration** We systematically optimize the original CSFD scheme. Without losing accuracy, we obtain multifold speedups, and our accelerated CSFD is as efficient as the analytic derivative. This is achieved by discarding high-order error terms, decoupling real and imaginary calculations, replacing expensive functions

(e.g. trigonometric functions), and isolating the propagation of the perturbation in composite and nesting functions.

- **Adaptation** Instead of resorting to high-order Taylor expansion or Fourier expansion, we choose to promote a real-valued function with multicomplex arithmetic, which leads to a *multicomplex-step finite difference* scheme (MCSFD) for high-order finite difference. Our acceleration techniques naturally synergize with this generalization. In addition, we leverage the Cauchy-Riemann formula to further adapt CSFD/MCSFD for tensor functions.
- **Application** We thoroughly validate CSFD/MCSFD in both numerical experiments as well as in complicated nonlinear deformable simulations. Without knowing the actual formulation of the internal force and the tangent stiffness matrix, nonlinear deformation can be robustly and accurately simulated with CSFD/MCSFD. First- and high-order modal derivative bases can also be constructed for sophisticated materials. Many challenging inverse simulation problems now can be easily tackled using standard gradient/Hessian-based optimization approaches such as the Newton's method (e.g. see Fig. 4.1).

4.2 Related Work

Calculating the differentiation of a function is an important computational procedure in many graphics research problems. For instance, in physics-based animation [109], such as rigid body dynamics [110, 111], fluid/smoke animation [112], and cloth simulation [28, 113], etc, the key challenge is to solve the unknown ordinary/partial differential equations, and one needs to use numerical approaches to discretize the differential operation. In computational fabrication, the optimal design is often obtained via following the gradient direction of the inverse simulation [44, 114, 115], not to mention a vast volume of research involving various

optimization procedures, many of which rely on the information of the gradient and/or Hessian of the objective function.

Evaluating the derivative is also a key ingredient in deformable simulation [116] especially for hyperelastic models [117]. Those materials are fully characterized by the strain energy density $E(\mathbf{F})$ of the local deformation gradient $\mathbf{F}$. Modeling such materials requires the first- and/or second-order spatial derivatives of E to establish the equilibrium equation. Dynamic simulation can also be casted as an optimization problem of the *variational form* [118, 119]. Newton's method [120], quasi-Newton method [121] or gradient descent method [5] can then be used when gradient information is provided. The closed-form formulation of derivatives of E for some materials can be found in the literature [7, 117, 122]. However, for other more complicated models like phenomenological and user-crafted materials [123, 124], obtaining the analytic derivative is non-trivial and labor-intensive. For principal stretch based nonlinear materials, such as Ogden and spline-based materials [125], careful numerical thresholding is required, even at the rest configuration, to obtain the actual tangent stiffness matrix. Deriving those derivatives analytically could be tedious and seemingly unworthy, if the user just wants to toy with a new hyperelastic energy to see how it behaves in a given animation scenario. Even with the help of *symbolic differentiation* packages like `Mathematica` [126] and `Maple` [127], the implementation efforts are still considerable. Besides, there are also many cases where the target function's formulation is not even accessible, and one has to use the numerical derivative to infer the underlying kinematics [128–130]. In model reduction, it is known that linear modes are not sufficient to capture large nonlinear deformation, and the modal derivative [11, 131] should be used. Those *derivative modes* are computed through evaluating the third-order gradient of the energy function (i.e. the Hessian of the internal force), which makes this technique less popular for more sophisticated materials other than the St. Venant-Kirchhoff (StVK) model.

The finite difference method is a standard procedure of computing the numerical derivative [132]. Its variances include forward difference (FD), backward difference (BD), and central difference (CD). CD is twice as expensive as FD or BD, but it is also the most accurate among them. Nevertheless, all of these schemes suffer from the subtractive cancellation issue: decreasing the magnitude of the perturbation does not make the finite difference converging, and the result will oscillate around the correct value and explode eventually [133]. This numerical behavior prevents the adoption of the finite difference method for applications that are sensitive to the accuracy of the differentiation.

On the other hand, CSFD is a powerful finite difference scheme but often overlooked in classic numerical analysis textbooks [102]. This method is based on the complex version of Taylor series expansion of a function, which dates back to the 1960s [134]. Unlike regular finite difference method, CSFD obviates the subtractive cancellation problem (in the first-order approximation) so that a very small perturbation (e.g. 1.0×10^{-20} or even smaller) can be used making the resulting derivative approximation highly accurate. Indeed, we show that CSFD is able to completely replace the analytic gradient without any accuracy concerns in deformable simulation. Due to its superior accuracy, CSFD has been gradually recognized and used for the sensitivity analysis [135–138]. For nonlinear finite element method (FEM) simulation with high-order shape functions, CSFD has also been used to obtain the numerical tangent stiffness matrix [139–141].

Because the target function is promoted to be a complex one, a naïve implementation of CSFD involves much heavier computations than the real-valued finite difference. We show that this limitation can be ameliorated by carefully manipulating the promoted target function and discarding high-order perturbation terms. Our results show that we are able to achieve a multifold speedup, making CSFD nearly as efficient as using the exact derivative. Instead of referring to the Fourier differ-

entiation [142, 143], we use the multicomplex-step finite difference [144] to handle high-order derivative. Doing so allows our acceleration scheme to be seamlessly integrated for high-order numerical derivatives.

CSFD vs. automatic differentiation Another relevant and widely known differentiation technique is the automatic differentiation (AD) [145–147], which decomposes complicated functions with the *chain rule*. AD has been used in graphics [30, 148, 149]. Indeed, the back propagation optimization [150] commonly used for neural network training is a special implementation of the *reverse AD*.

A key difference between CSFD and AD lies in the fact that *"AD uses exact formulas along with floating-point values"* [151], and it is *"not numerical differentiation"* [152]. CSFD, on the other hand, is a numerical approach seeking for the derivative approximation. AD is more sensitive to the smoothness of the function and could fail at discontinuities. CSFD behaves more robustly in such cases: because the complex perturbation is orthogonal to the real domain, CSFD always returns the derivative as long as the target function exists. AD also has practical difficulties for high-order generalization [153]. For instance, some existing AD packages (e.g. `Adept` [154]) only deals with the first-order derivative. While one may perform first-order differentiation multiple times to obtain a high-order derivative, it has been argued that recursively applying AD leads to inefficient and numerically unstable code [153, 155]. High-order AD is seldom well supported and could be extremely slow. On the other hand, MCSFD extension generalizes our acceleration scheme to high-order cases with excellent robustness and accuracy. Our accelerated CSFD/MCSFD is over 30× faster than commonly used AD packages even for the first-order case. Tensor functions that involving complicated numerical procedures are also problematic with AD. It remains unclear if the Cauchy–Riemann generalization [108] can be applied in AD. Our accelerated CSFD is orthogonal to and complements the AD technique.

Because CSFD is highly accurate (as accurate as the analytic result), it is possible to harness CSFD/MCSFD for calculating derivatives along the chain rule that could be otherwise troublesome to AD.

4.3 Background

In order to make the chapter more self-contained, we start our discussion with a brief review of the error source of the finite difference method and the numerical issue of the subtractive cancellation.

Suppose that the function $f : \mathbb{R} \to \mathbb{R}$ is differentiable around $x = x_0$. After a small perturbation h is applied, it can be Taylor expanded as:

$$\begin{aligned} f(x_0 + h) &= f(x_0) + f'(x_0) \cdot h + \frac{1}{2} f''(x_0) \cdot h^2 + \cdots \\ &= f(x_0) + f'(x_0) \cdot h + \mathbf{O}(h^2), \end{aligned} \tag{4.2}$$

which leads to the forward finite difference of Eq. (4.1). Eq. (4.2) also suggests that h should be as small as possible for a good approximation. In the meantime, because the total number of bits used to represent a real number is limited on a computer, all the floating-point arithmetics have the round-off error [156], which is a small relative error also known as *machine epsilon* ϵ. For the double precision of `IEEE 754` floating-point standard [157], $\epsilon \approx 1.11 \times 10^{-16}$. Normally, the round-off error does not seriously impair the stability or the accuracy of a numerical procedure. However, when h gets smaller, $f(x_0 + h)$ and $f(x_0)$ become nearly equal to each other. Subtraction between them would eliminate many significant digits, and the result after rounding could largely deviate from the actual value of $f(x_0 + h) - f(x_0)$.

We elaborate this issue using a simple four-digit decimal floating-point system. Here, a real number $a = 1999.99$ is represented as $\widetilde{a} = 1.999 \times 10^3$ (because we only have four digits for the mantissa), and we use $\widetilde{(\cdot)}$ to denote a digitalized number in a

floating-point system. In this example, we simply choose the round-by-chop rule that discards all the out-of-precision digits, and the corresponding round-off error is:

$$E_{round} = \frac{|a - \widetilde{a}|}{|a|} = \frac{|1999.99 - 1.999 \times 10^3|}{|1999.99|} \approx 4.95 \times 10^{-4}. \tag{4.3}$$

Next, let $b = 1998.88$, which is represented as $\widetilde{b} = 1.998 \times 10^3$. The error of calculating $a - b$ with this toy floating-point system is:

$$\begin{aligned} E_{subtraction} &= \frac{|(\widetilde{a} - \widetilde{b}) - (a - b)|}{|a - b|} \\ &= \frac{|(1.999 - 1.998) \times 10^3 - (1999.99 - 1998.88)|}{|1999.99 - 1998.88|} \\ &\approx 0.1. \end{aligned} \tag{4.4}$$

We can see from Eqs. (4.3) and (4.4) that rounding loses us the least important significant digit, and it only yields an error at the order of the floating-point precision (10^{-4}). However, the subtraction between $\widetilde{a}$ and $\widetilde{b}$ eliminates three leading significant digits, which yields a much more substantial error. If we set b even closer to a as $b = 1999.88$, $E_{subtraction}$ increases to 100% as all the significant digits are eliminated. This is why the cancellation of subtracting numbers of similar magnitude is also called *catastrophic cancellation*.

Some numerical literature (e.g. [146]) shows that CD with the form of:

$$f'(x_0) \approx \frac{f(x_0 + h) - f(x_0 - h)}{2h}, \tag{4.5}$$

has a better accuracy with a quadratic error, while FD and BD have an error term of $\mathbf{O}(h)$. This conclusion is based on the assumption that subtractive cancellation does not occur. As to be discussed in the next section, CD could be even more sensitive to a smaller h (because of its faster convergent rate).

4.4 Complex-Step Finite Difference

CSFD is based on the complex Taylor series expansion [105]. Let $(\cdot)^*$ denote a complex variable, and suppose $f^* : \mathbb{C} \to \mathbb{C}$ is differentiable around $x_0^* = x_0 + 0i$. If a perturbation h is applied at the imaginary domain, f^* can be expanded as:

$$f^*(x_0 + hi) = f^*(x_0^*) + f^{*'}(x_0^*) \cdot hi + \mathbf{O}(h^2). \tag{4.6}$$

For any smooth and real-valued function f, we can always lift it to be a complex one f^* by allowing complex input while retaining its computation procedure unchanged. Under this circumstance, both $f^*(x_0^*) = f(x_0) \in \mathbb{R}$ and $f^{*'}(x_0^*) = f'(x_0) \in \mathbb{R}$ do not have imaginary parts. Taking the imaginary part (i.e. using the operator $\texttt{Im}(\cdot) \in \mathbb{R}$) of both sides of Eq. (4.6) leads to $\texttt{Im}\big(f^*(x_0 + hi)\big) = \texttt{Im}\big(f^*(x_0^*) + f^{*'}(x_0^*) \cdot hi\big) + \mathbf{O}(h^3)$. We can then have the first-order CSFD approximation:

$$f'(x_0) = \frac{\texttt{Im}\big(f^*(x_0 + hi)\big)}{h} + \mathbf{O}(h^2) \approx \frac{\texttt{Im}\big(f^*(x_0 + hi)\big)}{h}. \tag{4.7}$$

Compared with Eq. (4.1) or Eq. (4.5), we can see that Eq. (4.7) does not have a subtractive numerator meaning it only has the round-off error regardless of the size of the perturbation h. In addition, the operation of $\texttt{Im}(\cdot)$ removes the $(hi)^2$ term in Eq. (4.6), making the actual approximation error $\mathbf{O}(h^2)$. Thus, we can employ a very small h to obtain a highly accurate numerical derivative.

In addition to complex-step finite difference of Eq. (4.7), it is also possible to apply the perturbation in the *dual domain*, which corresponds to the *dual number* method [158, 159]. Similar to the complex number, a dual number $d = a + b\epsilon$ has a real part a and a dual part $b\epsilon$ such that $\epsilon \neq 0$ but $\epsilon^2 = 0$. This property makes all the higher-order terms of ϵ vanished. As a result, the dual version Taylor expansion of a given function leads to $f(x_0 + h\epsilon) = f(x_0) + f'(x_0) \cdot h\epsilon$, and one can obtain the *exact* derivative by exacting dual part of $f(x_0 + h\epsilon)$ and setting $h = 1$ as: $f'(x_0) = \texttt{Du}\big(f(x_0 + \epsilon)\big)$. The question here is how can we evaluate the dual

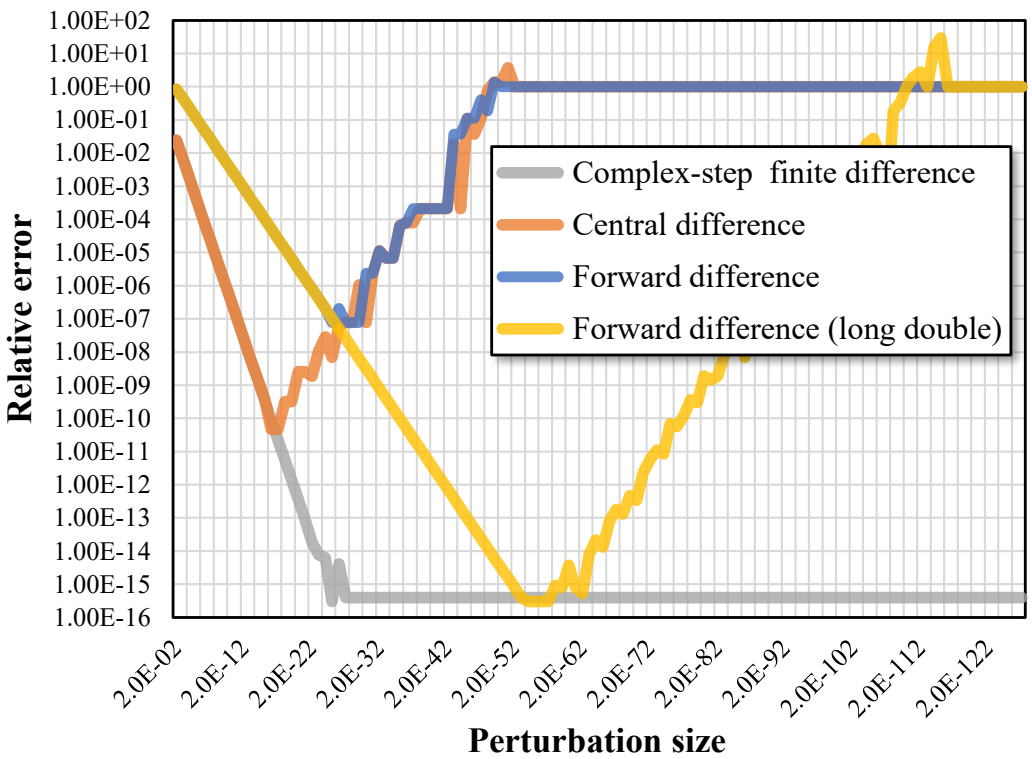


Figure 4.2: We use FD, CD, and CSFD to calculate the first-order derivative of $f(x) = e^x/(x^4 + x^2 + 1)$ at $x = 4$. The resulting numerical derivative is compared with the analytic derivative, and the relative error is plotted against the size of the perturbation, ranging from 2^{-2} to 2^{-127}.

function $f(x_0 + \epsilon)$. The literature of dual number arithmetic is far less extensive than the complex arithmetic. Normally, a dual function can be evaluated as a dual polynomial [160], which is based on the analytic derivative $f'(x_0)$. In other words, the dual number formulation is equivalent to using the analytic differentiation (because you need to compute $f'(x_0)$ to obtain the promoted dual function).

Fig. 4.2 reports a numerical experiment of $f(x) = e^x/(x^4 + x^2 + 1)$. We compute the first-order numerical derivative at $x = 4$ using FD, CD and CSFD. The analytic derivative of this simple function can be derived as: $f'(x) = (x^4 - 4x^3 + x^2 - 2x + 1)e^x/(x^4 + x^2 + 1)^2$, and $f'(4) = 0.006593183194438$ is considered as the

ground truth. In this example, both $f(x)$ and $f'(x)$ are well scaled, and the issue of subtractive cancellation starts to take place when $h \approx 2^{-26} \sim 1.0 \times 10^{-8}$ with FD and $h \approx 2^{-17} \sim 1.0 \times 10^{-6}$ with CD. We do see CD converges faster than FD before hitting the threshold of subtractive cancellation. However, both CD and FD explode quickly after the cancellation occurs. When h becomes smaller than $2^{-47} \sim 1.0 \times 10^{-15}$, the subtractive cancellation eliminates all the significant digits making $f(x_0 + h) - f(x_0)$ and $f(x_0 + h) - f(x_0 - h)$ vanished by the rounding. In this case, we cannot obtain any useful information of the derivative out of the finite difference approximation, and the relative error stays 100%. The numerical performance of FD can be improved by extending the floating number precision. As indicated in the figure, after doubling the precision from `double` to `long double` (128 bit), the subtractive cancellation is delayed. If we choose the perturbation size carefully, FD is also able to yield good accuracy in this particular example. In real applications however, f often takes a high-dimension vector $\mathbf{x}$ as the dependable variable. f and $\partial f / \partial x_i$ may also be badly scaled. These circumstances make subtractive cancellation happen much earlier. As a result, the numerical derivative of FD and CD is fallible: a conservative h has a big approximation error, while an aggressive h could be even worse due to the cancellation. In physics-based simulations, FD/CD is always problematic and often the demon behind the numerical instability. On the other hand, CSFD shows a superior performance in terms of both convergence rate and numerical stability. As CSFD does not have the subtractive cancellation problem, the relative error decreases consistently with a smaller h. When h is sufficiently small (i.e. $h < 2^{-26} \sim 1.0 \times 10^{-8}$), CSFD delivers a result with an error below 1.0×10^{-15}. Note that the "ground truth" itself also has a round-off error at the order of 10^{-16}. In other words, CSFD is as accurate as the analytic derivative for a sufficiently small h.

Naïve complex promotion In order to apply CSFD, we must promote the real function $f(x)$ to be a complex one $f^*(x^*)$. While the specific form of $f(x)$ could

be complicated, it is always constructed with binary operators of $+$, $-$, $\times$, $\div$ and unary operators including power function (x^a), exponential function (e^x), logarithmic function ($\ln x$), and trigonometric functions ($\sin x$ etc.). Promoting these elementary functions follows the standard complex number arithmetic [161]. For a quick reference, we also list the complex promotion of some commonly used functions in Appendix B.

If efficiency is not the primary concern, CSFD can be quickly implemented via overloading existing floating-point arithmetic operators with the corresponding complex version. `C++ Template` provides a flexible mechanism for this purpose: one can code $f(x)$ using a generic data type and choose the complex-type template specialization when CSFD is needed. Standard `C++ STD` library has a collection of stable complex number routines. Besides, there are also a few third-party opensource complex number libraries such as `Boost` [162] and `Eigen` [163]. Nevertheless, such naïve CSFD implementation induces a significant overhead. In many cases, CSFD runs orders-of-magnitude slower than the analytic derivative. One of our major contributions is to optimize CSFD computation to regain the efficiency of the finite difference. This is to be detailed in the next section.

4.5 CSFD Acceleration

Using general-purpose complex number arithmetic to promote $f(x)$ is actually "overkill" for just using CSFD to compute numerical derivatives. We show that CSFD approximation can be substantially simplified and accelerated, and our accelerated CSFD is as efficient as using the analytic derivative. Our strategy is based on the following three important observations:

- According to Eq. (4.7), it is clear that calculating the real part of f^* is unnecessary for CSFD, therefore nearly half of the computation brought by the complex promotion can be discarded.

- Complex number arithmetic for CSFD is quite different from a general complex operation. The imaginary part of f^* comes from the applied perturbation hi, which is a very small value (i.e. $h < 1.0 \times 10^{-20}$). Many calculations can be simplified by treating h as an infinitesimal: for instance we can have $\sin h \sim h$ to avoid the expensive evaluation of the trigonometric function of $\sin h$.
- Because h appears as the denominator of Eq. (4.7), all the quadratic or higher-order terms of h in $\texttt{Im}\big(f^*(x_0+hi)\big)$ can be discarded, which only leads to an approximation error up to $\mathbf{O}(h)$.

4.5.1 Accelerate CSFD of a Single Elementary Function

We start our discussion by assuming that $f(x)$ is an elementary function (i.e. listed in Appendix B), and take $f(x) = x^{1/m}$ an example to show how it can be much more efficiently evaluated for CSFD. First, the standard complex promotion (Eq. (B.4)) gives us:

$$\frac{\texttt{Im}\big(f^*(x_0 + hi)\big)}{h} = \frac{1}{h}\left(r^{\frac{1}{m}} \cdot \sin\frac{\phi}{m}\right). \tag{4.8}$$

Here, $r(\cos\phi + \sin\phi i)$ is the polar form of $x_0 + hi$. Recalling that h is a very small quantity, we have:

$$\sin\phi = \frac{h}{r} \Rightarrow \phi = \frac{h}{r}, \tag{4.9}$$

because $\langle \sin a \sim a \rangle$ is a pair of equivalent infinitesimals when $a \to 0$. With Eq. (4.9), the RHS of Eq. (4.8) can be greatly simplified as:

$$\frac{1}{h}\left(r^{\frac{1}{m}} \cdot \sin\frac{\phi}{m}\right) = \frac{1}{h}\left(r^{\frac{1}{m}} \cdot \frac{\phi}{m}\right) = \frac{1}{h}\left(r^{\frac{1}{m}} \cdot \frac{h}{rm}\right) = \frac{r^{\frac{1}{m}}}{rm}. \tag{4.10}$$

The performance improvement of Eq. (4.10) is substantial. We record the computation time of running Eqs. (B.4), (4.8), and (4.10) respectively as well as directly

Eq. (B.4)	Eq. (4.8)	Eq. (4.10)	Analytic
13.1s	9.49s(1.4x)	0.056s(233x)	0.064s

Table 4.1: Time statistics of using the optimized CSFD (i.e. Eqs. (4.8) and (4.10)) and the naïve CSFD implementation (Eq. (B.4)) of the exponential function $f(x) = x^{1/m}$ for 100 million times. The computation time using analytic derivative is also reported for the reference. Our CSFD simplification is over 200× faster than the naïve implementation. In this example, it is even faster than using the analytical derivative. The relative error is at the order of the machine epsilon (10^{-16}).

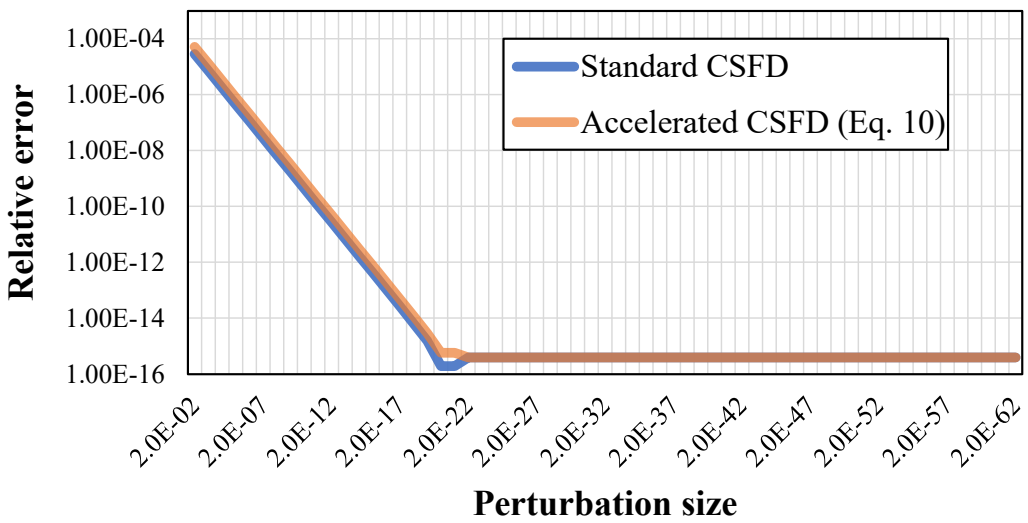


Figure 4.3: Our fast CSFD implementation has good numerical stability and accuracy. The relative error converges as quickly as the regular CSFD and remains at the order of machine epsilon after h is sufficiently small.

calculating the analytical derivative of $\left(x^{\frac{1}{m}}\right)' = \frac{1}{m}x^{\frac{1}{m}-1}$ for 100 million times on an `Intel i7` laptop. The result is reported in Tab. 4.1. As expected, we can see from the table that Eq. (4.8) modestly improves the calculation efficiency by discarding real part computation. The most significant speedup originates from equivalent infinitesimal based simplification, which frees us from performing expensive trigonometric function calculation. In this example, CSFD is even faster than using analytic derivative because Eq. (4.10) has a simpler exponential term of $(\cdot)^{\frac{1}{m}}$ than the exponential term

in the analytic derivative: $(\cdot)^{\frac{1}{m}-1}$. Meanwhile, our accelerated CSFD retains all the favored advantages of CSFD. As shown in Fig. 4.3, fast CSFD implementation has the same convergency and accuracy. For small h, the relative error reaches the machine epsilon stably. Interestingly, if one further simplifies r such that $r = \sqrt{x_0^2 + h^2} = x_0$ when $h \to 0$, Eq. (4.10) converges to the analytic derivative formulation. This finding reveals that, unlike regular finite difference, *the actual derivative of the function is essentially hidden in its complex promotion.* This is another important reason that explains why CSFD is able to achieve such high accuracy.

The strategy of leveraging equivalent infinitesimals can be readily applied to other elementary functions. For instance for trigonometric functions, the most expensive arithmetic is the evaluation of $e^{\pm h}$. Again, because h is an infinitesimal, we exploit the fact that $\langle e^{\pm h} \sim 1 \pm h \rangle$ is also a pair of equivalent infinitesimals. This simplification brings another orders-of-magnitude speedup.

4.5.2 Accelerate CSFD of Composite Binary Operators

In reality, $f(x)$ houses a chain of binary operators such that:

$$f(x) = f_1(x) \circ f_2(x) \circ f_3(x) \circ \cdots \circ f_k(x) \circ \cdots \circ f_N(x), \tag{4.11}$$

for $\circ \in \{+, -, \times, \div\}$. Each $f_k(x)$ could also be a nesting composite of multiple unary functions: $f_k(x) = f_{k,1}(f_{k,2}(f_{k,3}(...)))$. We defer the discussion of nesting operators to the next subsection and assume that the promoted form of each function along the chain is known.

Eq. (4.11) may be split into several sub-chains according to the parenthesization and operator priority. For instance, the example used in Fig. 4.2 can be understood as $f(x) = f_1(x)/\big(f_2(x) + f_3(x) + f_4(x)\big)$, where $f_1(x) = e^x$ is an exponential function; $f_2(x) = x^4$ and $f_3(x) = x^2$ are power functions; and $f_4(x) = 1$ is a constant. If a

sub-chain only consists of addition and subtraction operators, which are independent for real and imaginary parts, we just evaluate the imaginary part of each promoted function f_k^* along the chain for CSFD approximation and ignore the calculation for the real part.

However, if the sub-chain is concatenated with multiplication and/or division operators, we cannot discard the real part of each function because the real and imaginary parts are coupled in the multiplication operation – one can easily verify that the imaginary part of $f_1^*(x^*) \cdot f_2^*(x^*)$ contains the information of both real and imaginary parts of $f_1^*(x^*)$ and $f_2^*(x^*)$. Division is similar, which is regarded as the multiplication of the conjugate of the dividend.

We show that evaluating a multiplication chain can also be significantly accelerated based on a binary branching strategy. Let $f_k^*(x^*) = a_k + b_k$ denote a promoted function on the chain, where b_k is an imaginary quantity. Our base case is the chain of a single promoted function $f^*(x^*) = f_1^*(x^*) = a_1 + b_1$ with two addends. Putting an additional multiplying function after it leads to $f^*(x^*) = f_1^*(x^*) \cdot f_2^*(x^*) = (a_1 + b_1)a_2 + (a_1 + b_1)b_2$. In other words, each item of a_1 and b_1 is multiplied by a_2 and b_2 respectively. The multiplication of $f_2^*(x^*)$ thus doubles the total number of addends. This procedure can also be visualized with a binary tree shown in Fig. 4.4. Each complex function $f_k^*(x^*)$ along the chain increments the height of the tree by one, and we have 2^N addends at the bottom level for a chain of N functions. Recall that imaginary parts of b_k correspond to a very small perturbation $b_k = hi \sim 0$, and we can discard all addends that are quadratic or higher-order of b_k. The key question here is how can we directly identify those addends without actually expanding the multiplication chain?

From Fig. 4.4, we can see that each extra multiplication induces a binary branch towards the next level. A left branch appends an a_k after an existing addend while a right branch appends a b_k. The final form of a leaf addend depends on how many left and right branches at which levels it takes along the path from the root. Clearly,

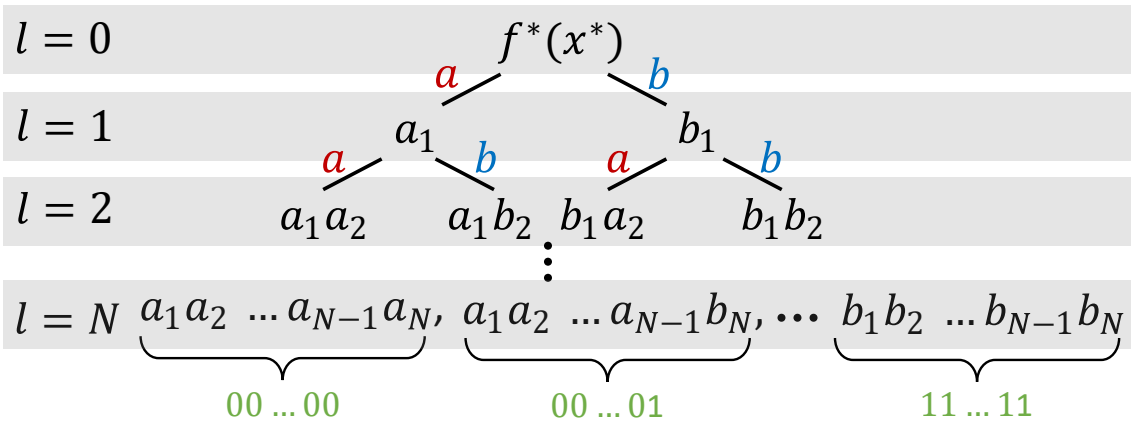


Figure 4.4: The procedure of evaluating a chain of multiplications (and divisions) can be visualized with a binary tree. The leaf nodes can be concisely encoded by a binary number. As a result, we can discard higher-order infinitesimals with two or more 1s (e.g. 011) at the bottom level.

the leftmost and rightmost leaves are always $a_1a_2...a_{N-1}a_N$ and $b_1b_2...b_{N-1}b_N$. The second leftmost leaf differs from the leftmost one because it takes a right branch at the last level. Accordingly, its final form becomes $a_1a_2...a_{N-1}b_N$. Interestingly, this branching mechanism mimics the ripple-carry addition of binary numbers. If we encode a_k with 0 and b_k with 1, all the addends at the bottom level, from left to right, can be concisely represented as a sequence of binary numbers $B_0, B_1, B_2, ..., B_{2^N-1}$ such that $B_k = (k)_{binary}$ is the binary representation of the decimal index k. For instance, if we have three functions along the chain, the eight leaf addends from B_0 to B_7 are: 000, 001, 010, 011, 100, 101, 110 and 111. The number of ones in B_k implies the order of h. Since anything higher-order than h^2 can be safely discarded, we only sum up addends with exact one 1-digit (the leftmost leaf is a real number, which is also discarded) such that: $\texttt{Im}\big(f^*(x^*)\big) = a_1a_2b_3 + a_1b_2a_3 + b_1a_2a_3 + \mathbf{O}(h^2)$. From Eq. (4.6), we can also understand that $f(x_0) = \texttt{Re}\big(f^*(x_0 + hi)\big) + \mathbf{O}(h^2)$ meaning replacing a_k by $f_k(x)$ only induces an approximation error of $\mathbf{O}(h^2)$. As a result, we can stick with our acceleration strategy of ignoring the real part of a promoted function. If a long multiplication chain is identified (e.g. more than 10 multiplicands)

whose derivative is to be evaluated via CSFD, we also pre-compute the product among all the a_k as:

$$A = \Pi_{k=1}^{N} a_k = \Pi_{k=1}^{N} f_k(x) + \mathbf{O}(h^2). \tag{4.12}$$

Therefore, a leaf node, say $a_1 b_2 a_3$ for instance, can be efficiently computed at $\mathbf{O}(1)$ time as:

$$a_1 b_2 a_3 = \mathtt{Re}\big(f_1^*(x^*)\big) \cdot \mathtt{Im}\big(f_2^*(x^*)\big) \cdot \mathtt{Re}\big(f_3^*(x^*)\big) \approx \frac{A}{f_2(x)} \cdot \mathtt{Im}\big(f_2^*(x^*)\big). \tag{4.13}$$

Note that using Eq. (4.12) potentially brings us the division-by-zero issue if $f_k(x)$ is zero or close to zero. This risk can be avoided by rolling back to the standard formula of $a_1 b_2 a_3$ instead of Eq. (4.13), if $f_2(x)$ is found smaller than a given threshold (say 1.0×10^{-16}). The timing benchmark shows that our strategy brings CSFD approximation an additional $5\times$ boost. After the value of each $f_k(x)$ is computed, the naïve implementation uses 21 ms to calculate the first-order CSFD derivative for $N = 100$, while our method only needs 4 ms on an `i7` laptop.

4.5.3 Accelerate CSFD of Composite Unary Operators

Real-world functions may also be in a nesting form of multiple unary operators:

$$f(x) = f_N(f_{N-1}(f_{N-2}(\cdots f_2(f_1(x))))), \tag{4.14}$$

where each f_k for $1 \leq k \leq N$ could be a power, exponential, logarithmic, or trigonometric function. We stick with the notation of $f_k^*(x^*) = a_k + b_k$, where b_k is an imaginary quantity, and $x_0 + hi = a_0 + b_0$ is the input of f_1^* i.e. the innermost function. Note that the CSFD approximation of $f'(x)$ is actually the ratio between b_N and b_0:

$$f'(x) \approx \frac{\mathtt{Im}\big(f_N^*\big)}{h} = \frac{\mathtt{Im}\big(f_N^*\big)i}{hi} = \frac{\mathtt{Im}\big(a_N + b_N\big)i}{hi} = \frac{b_N}{b_0}. \tag{4.15}$$

Similar to the multiplication case, the real and imaginary parts of an outer function are also coupled with the input real and imaginary parts from its inner function. The

algebraic relation between b_N and b_0 could be complicated, and expanding the entire composite equation to compute the actual value of b_N is expensive.

Fortunately we notice that in order to compute $f'(x)$ with CSFD, only the ratio between b_N and b_0 is needed, and their exact values are of less interest. Therefore, we convert b_N/b_0 to be:

$$\frac{b_N}{b_0} = \frac{b_1}{b_0} \cdot \frac{b_2}{b_1} \cdot \frac{b_3}{b_2} \cdot \ldots \cdot \frac{b_N}{b_{N-1}}. \tag{4.16}$$

A multiplicand in RHS of Eq. (4.16), b_k/b_{k-1}, describes how the imaginary perturbation is changed through f_k^*. An important observation here is *the imaginary part of a promoted function remains infinitesimally small after being applied with an infinitesimal imaginary perturbation.* This can be easily verified by the complex Taylor expansion: $f^*(x_0 + hi) \approx f^*(x_0^*) + f^{*'}(x_0^*) \cdot hi$, which leads to $\mathtt{Im}\big(f^*(x_0 + hi)\big) \approx f'(x_0) \cdot h = \mathbf{O}(h) \sim h$. In other words, all the b_k in Eq. (4.16) are small imaginary perturbations of the same order of hi. Therefore, we re-set each intermediate perturbation of b_{k-1} as h. In the meantime, its real part input a_{k-1} can be efficiently computed as f_{k-1} without resorting to f_{k-1}^* as:

$$\begin{aligned} \frac{b_k}{b_{k-1}} &= \frac{\mathtt{Im}\big(f_k^*(a_{k-1} + b_{k-1})\big)i}{b_{k-1}} \\ &\approx \frac{\mathtt{Im}\big(f_k^*(a_{k-1} + hi)\big)}{h} \approx \frac{\mathtt{Im}\big(f_k^*(f_{k-1} + hi)\big)}{h}. \end{aligned} \tag{4.17}$$

Eq. (4.17) literally breaks the coupling of the imaginary parts along the nesting chain – when computing b_k/b_{k-1}, the actual imaginary values from inner functions are not required, and the propagation of the initial imaginary perturbation $b_0 = hi$ is isolated.

Discussion Eq. (4.16) should look immediately similar to the chain rule, which forms the foundation of AD techniques. Indeed, one may also understand Eq. (4.17) as breaking Eq. (4.14) using the chain rule and applying CSFD to approximate each intermediate derivative afterwards (i.e. by setting $b_{k-1} = hi$ and $a_{k-1} =$

f_{k-1}). In other words, Eq. (4.16) is practically equivalent to *augmenting AD with accelerated CSFD* without referring to differentiation rules. Regular AD packages (e.g. `CppAD` [164] and `Adept` [154]) mainly aim on first- or second-order derivatives, and their generalization to high-order derivative is nonintuitive and inefficient, if not impossible. However, as we will see in the next section, CSFD can be elegantly generalized to handle high-order derivatives. All the acceleration techniques discussed in this section are naturally inherited.

4.6 Multicomplex-Step Perturbation

Regular finite difference evaluates high-order derivative by recursively applying the first-order approximation of Eq. (4.1). For instance, the second-order derivative is approximated as:

$$f''(x_0) = \frac{f(x_0 + h) - 2f(x_0) + f(x_0 - h)}{h^2} + \mathbf{O}(h^2), \tag{4.18}$$

which requires two extra function evaluations for both $f(x_0 + h)$ and $f(x_0 - h)$. The complex Taylor series expansion of Eq. (4.6) gives a real second-order term (with a factor of i^2), which yields:

$$f''(x_0) = \frac{2\left(f(x_0) - \mathtt{Re}\big(f^*(x_0 + hi)\big)\right)}{h^2} + \mathbf{O}(h^2). \tag{4.19}$$

Eq. (4.19) only needs one extra function evaluation of $f^*(x_0 + hi)$: its imaginary part can be used for the first-order CSFD while its real part is being used for the second-order CSFD. However, both schemes suffer with the subtractive cancellation. Besides, computing $f^*(x_0 + hi)$ could be even slower than computing both $f(x_0 + h)$ and $f(x_0 - h)$ due to the extra complexity induced by the promotion. Therefore, second-order CSFD is less appealing to us. A more numerical stable approach is based on *Fourier differentiation* [143], which generalizes the complex Taylor expansion to

Fourier expansion by not retaining the perturbation on the imaginary axis:

$$f^*(x_0 + he^{\theta i}) = f^*(x_0^*) + f^{*'}(x_0^*) \cdot he^{\theta i} + f^{*''}(x_0^*) \cdot \frac{h^2}{2} e^{2\theta i} \cdots . \tag{4.20}$$

High-order derivative can be computed by using different argument angles of θ to cancel out unwanted terms. For instance, setting $\theta = \pi/4$ and $\pi + \pi/4$ leads to one possible second-order approximation [142]:

$$f^{*''}(x_0^*) = \frac{\mathtt{Im}\big(f^*(x_0 + h \cdot i^{\frac{1}{2}}) + f^*(x_0 - h \cdot i^{\frac{1}{2}})\big)}{h^2} + \mathbf{O}(h^2). \tag{4.21}$$

While Fourier differentiation may be able to avoid the subtractive cancellation with a carefully chosen θ, its formulation is quite different from the first-order CSFD[1]. In practice, users have to use distinct implementations for different differentiation orders, and most calculations among them cannot be shared.

Alternatively, there is a more concise formula that generalizes the perturbation to be a *multicomplex* quantity, and we refer to this method as multicomplex-step finite difference (MCSFD). The detailed derivation of MCSFD formulation can be found in existing literature [144, 165]. MCSFD extends the regular complex number to have multiple mutual-orthogonal imaginary directions. The most attractive feature of multicomplex number to us is it can be defined recursively: its base cases are the real number $\mathbb{R}$ and the regular complex number $\mathbb{C}$, which are considered as zero- and first-order multicomplex sets $\mathbb{C}^0$ and $\mathbb{C}^1$. $\mathbb{C}^1$ extends the real set ($\mathbb{C}^0$) by adding an imaginary unit i as: $\mathbb{C}^1 = \{x + yi | x, y \in \mathbb{C}^0\}$, and the multicomplex number up to an order of n is defined as:

$$\mathbb{C}^n = \left\{ z_1 + z_2 i_n | z_1, z_2 \in \mathbb{C}^{n-1} \right\}. \tag{4.22}$$

The order of a multicomplex number matches the number of its imaginary directions, and all the imaginary units i_n have the property of $i_n^2 = -1$. Fully expanding the

[1]The fact is Fourier differentiation still has the subtractive cancellation issue. Explicitly avoiding the subtraction is not a real cure of cancellation. This is out of scope of this chapter, but numerical experiments clearly verify this.

recurrence of Eq. (4.22) yields:

$$\begin{aligned}\mathbb{C}^n = \; & x_0 + x_1 i_1 + x_2 i_2 + \cdots + x_n i_n \\ & + x_{1,2} i_1 i_2 + \cdots + x_{n-1,n} i_{n-1} i_n \\ & + x_{1,2,3} i_1 i_2 i_3 + \cdots + x_{n-2,n-1,n} i_{n-2} i_{n-1} i_n \\ & \; \vdots \\ & + x_{1,2,\ldots,n} i_1 i_2 \cdots i_n,\end{aligned} \tag{4.23}$$

where all of x_0, x_1, ..., x_n, $x_{1,2}$, $x_{2,3}$, ..., $x_{n-1,n}$, ..., $x_{1,2,\ldots,n}$ are real coefficients. For instance, setting $n = 2$ leads to $\mathbb{C}^2 = x_0 + x_1 i_1 + x_2 i_2 + x_{1,2} i_1 i_2$. A $\mathbb{C}^n$ number has 2^n x-coefficients: one x_0 for the real part, n coefficients x_1, x_2, ..., x_n for a single imaginary direction. All the other coefficients are for mixed imaginary directions with multiple i_j. Unlike quaternion [166], the product between different imaginary units is commutative such that $i_j \cdot i_k = i_k \cdot i_j$ for $j \neq k$.

Following the formulation in [144], the Taylor series expansion of $f^\star$ under a multicomplex perturbation is:

$$\begin{aligned}f^\star(x_0 + h i_1 + \cdots + h i_n) &= f^\star(x_0) + f^{\star(1)}(x_0) \cdot h \sum_{j=1}^{n} i_j \\ &+ \frac{f^{\star(2)}(x_0)}{2} \cdot h^2 \left(\sum_{j=1}^{n} i_j \right)^2 + \cdots + \frac{f^{\star(n)}}{n!} \cdot h^n \left(\sum_{j=1}^{n} i_j \right)^n + \cdots .\end{aligned} \tag{4.24}$$

Here, $f^{\star(n)}$ is the n-th-order derivative of $f^\star$. $(\sum i_j)^k$ can be expanded following the *multinomial theorem*, and it contains products of mixed k imaginary directions for the k-th-order term. We refer the reader to [144, 165] for a detailed step-by-step derivation. Because $(\sum i_j)^k \neq (\sum i_j)^l$ for $k \neq l$, Eq. (4.24) allows us to approximate an arbitrary-order derivative by directly extracting the corresponding imaginary combination, just as we did in CSFD. In order to do so, $\mathtt{Im}(\,\cdot\,)$ should also be generalized to $\mathtt{Im}_\kappa(\,\cdot\,)$ to handle multiple imaginary directions:

$$\mathtt{Im}_\kappa(z) = x_\kappa \in \mathbb{R}, \tag{4.25}$$

which picks a coefficient x_κ that matches the imaginary combination of κ (i.e. the subscripts combination of i_j).

The MCSFD approximation of the n-th-order derivative can then be concisely formulated as:

$$f^{(n)}(x_0) = \frac{\mathtt{Im}^{(n)}\big(f^\star(x_0 + hi_1 + hi_2 + ... + hi_n)\big)}{h^n} + \mathbf{O}(h^2). \tag{4.26}$$

Similarly, n-th-order partial derivative can be approximated as:

$$\frac{\partial^n f(x_1, \cdots, x_p)}{\partial x_1^{k_1} \cdots \partial x_k^{k_p}} \approx \frac{\mathtt{Im}^{(n)}\big(f^\star(x_1 + h\sum_{j\in\Pi_1} i_j, \cdots, x_p + h\sum_{j\in\Pi_p i_j})\big)}{h^n}, \tag{4.27}$$

where $\mathtt{Im}^{(n)} = \mathtt{Im}_{1,2,..,n}$ is a shortcut notation, which picks the coefficient of the mixed imaginary direction of $i_1 i_2 \cdots i_n$. $\Pi_j = \left\{\sum_{l=1}^{j-1} k_l + 1, \cdots, \sum_{l=1}^{j} k_l\right\}$. By setting $n = 2$ in Eqs. (4.26) and (4.27), elements of the Hessian matrix (of a function $f(x, y) : \mathbb{R}^2 \to \mathbb{R}$) can be easily obtained as:

$$\begin{cases} \dfrac{\partial^2 f(x,y)}{\partial x^2} \approx \dfrac{\mathtt{Im}^{(2)}\big(f(x + hi_1 + hi_2, y)\big)}{h^2}, \\ \dfrac{\partial^2 f(x,y)}{\partial y^2} \approx \dfrac{\mathtt{Im}^{(2)}\big(f(x, y + hi_1 + hi_2)\big)}{h^2}, \\ \dfrac{\partial^2 f(x,y)}{\partial x \partial y} = \dfrac{\partial^2 f(x,y)}{\partial y \partial x} \approx \dfrac{\mathtt{Im}^{(2)}\big(f(x + hi_1, y + hi_2)\big)}{h^2}. \end{cases} \tag{4.28}$$

The most pleasing advantage of MCSFD to us is its handy implementation. As long as CSFD is implemented, all the routines for CSFD can be recursively used for MCSFD. More importantly, all the acceleration techniques discussed in Sec. 4.5 are also inherited with MCSFD. The numerical performance of MCSFD is excellent as reported in Fig. 4.5, where we evaluate the second-order derivative of $f(x) = e^x/(x^4 + x^2 + 1)$ at $x = 4$, the same example used in Fig. 4.2. The actual derivative $f''(x) = (x^8 - 8x^7 + 22x^6 - 12x^5 + 21x^4 - 12x^3 - 4x^2 - 4x - 1)e^x/(x^4 + x^2 + 1)^3$ is used as the reference. In this example, second-order finite difference (Eq. (4.18)) has a similar behavior of its first-order counterpart. After h hits a certain threshold

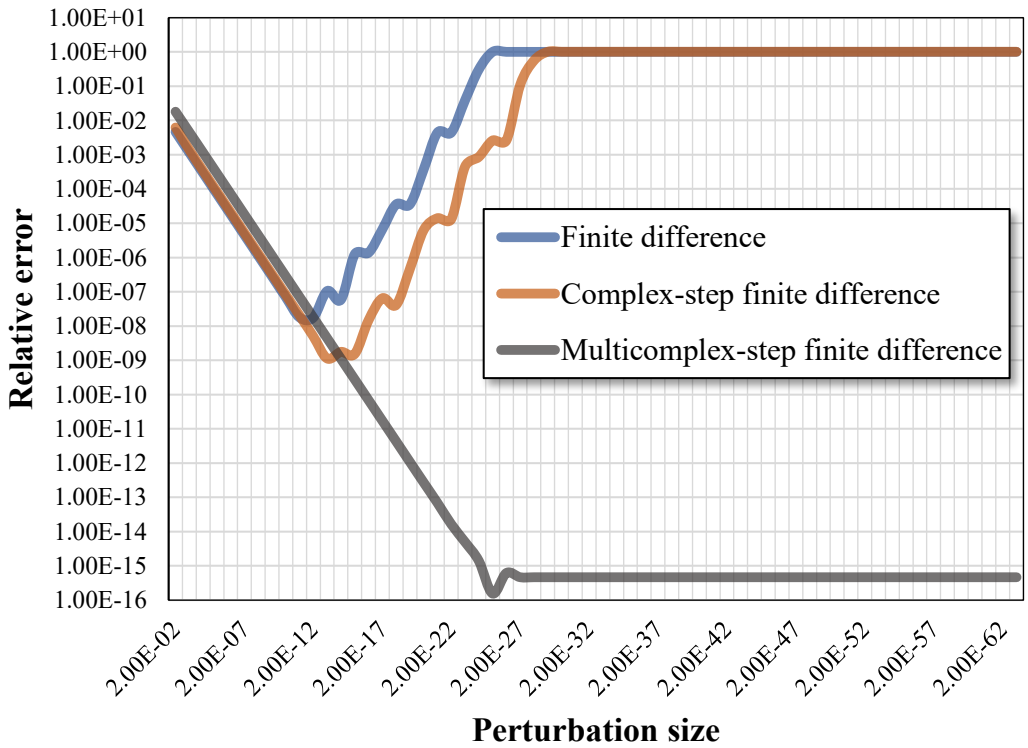


Figure 4.5: The performance of MCSFD approximation. We use the same test function of $f(x) = e^x/(x^4 + x^2 + 1)$ as in Fig. 4.2 and calculate its second-order derivative at $x = 4$. The relative error w.r.t to value of analytic derivative is plotted against the size of the perturbation, ranging from 2^{-2} to 2^{-63}.

($\sim 1.0 \times 10^{-13}$), the subtractive cancellation makes the numerator a numerical zero leading to a 100% relative error. The second-order CSFD approximation of Eq. (4.19) also suffers from this issue. MCSFD however, accurately approximates the second-order derivative. With a sufficiently small h, the relative error becomes comparable to the machine epsilon, and the approximation can be used to fully replace the analytic derivative.

4.7 Tensor Function

Most examples we have discussed so far are real functions taking a single real-valued input. In many simulation problems, however, we deal with functions with a tensor input. If we know how each component of the input tensor contributes to the output, we can simply overload the corresponding calculation to evaluate the promoted function value. For instance, $f(\mathbf{X}) : \mathbb{R}^{N\times N} \to \mathbb{R} = |\mathbf{X}|_F$ returns the Frobenius norm of the input matrix $\mathbf{X}$. As we know the exact form of this function is: $f(\mathbf{X}) = \sqrt{\sum\sum X_{i,j}^2}$, evaluating the partial derivative of $\partial f(\mathbf{X})/\partial X_{i,j}$ is nothing more than fixing unrelated tensor components to pose f as a scalar-input function.

However, there are also many functions that do not rely on an explicit formulation such as the one solving an input linear system:

$$f(\mathbf{X}) : \mathbb{R}^{N\times N} \to \mathbb{R}^N = \mathbf{X}^{-1}\mathbf{a}. \tag{4.29}$$

The exact inverse of a high-dimension matrix $\mathbf{X}$ is seldom given analytically. Instead, appropriate numerical routines like LU decomposition and forward-backward substitution are used to retrieve the function output. It is difficult for us to apply CSFD or MCSFD promotions without altering the underlying implementation of those numerical procedures.

An important advantage of CSFD/MCSFD is that one can exploit the *Cauchy-Riemann* (CR) formulation [108] to achieve (multi-)complex perturbations without overloading the complex arithmetic. CR form represents a multicomplex number in the form of a real matrix. Suppose $z^1 = z_0^0 + z_1^0 i$, its CR form is a 2×2 matrix:

$$z^1 = z_0^0 + z_1^0 i = \begin{bmatrix} z_0^0 & -z_1^0 \\ z_1^0 & z_0^0 \end{bmatrix}, \text{ where } z^1 \in \mathbb{C}^1 \text{ and } z_0^0, z_1^0 \in \mathbb{C}^0 = \mathbb{R}.$$

Here, we use the superscript $(\cdot)^n$ to denote the order of a multicomplex number. The CR matrix of z^n can be constructed recursively using the CR matrices of z_0^{n-1} and

z_1^{n-1} following the definition of the multicomplex number (Eq. (4.22)) as:

$$z^n = z_0^{n-1} + z_1^{n-1} i_n \in \mathbb{C}^n = \begin{bmatrix} z_0^{n-1} & -z_1^{n-1} \\ z_1^{n-1} & z_0^{n-1} \end{bmatrix}. \tag{4.30}$$

Each of the 2×2 blocks in Eq. (4.30) is a $(n-1)$-order multicomplex number, which can be further expanded with $(n-2)$-order multicomplex numbers and so on. Eventually, the CR form of z^n becomes a $2^n \times 2^n$ real matrix.

CR form can also be generalized for tensors i.e. z_0^0 and z_1^0 can be real-valued tensor quantities. As a result, $f(\mathbf{X})$ of Eq. (4.29) can be promoted as:

$$f^*(\mathbf{X}^*) = \begin{bmatrix} \mathtt{Re}(\mathbf{X}^*) & -\mathtt{Im}(\mathbf{X}^*) \\ \mathtt{Im}(\mathbf{X}^*) & \mathtt{Re}(\mathbf{X}^*) \end{bmatrix}^{-1} \begin{bmatrix} \mathtt{Re}(\mathbf{a}^*) & -\mathtt{Im}(\mathbf{a}^*) \\ \mathtt{Im}(\mathbf{a}^*) & \mathtt{Re}(\mathbf{a}^*) \end{bmatrix}. \tag{4.31}$$

Because all the tensors are now real quantities, Eq. (4.31) can be evaluated without involving any complex number calculations. The resulting function value is also the CR form of $f^*(\mathbf{X}^*)$, and we can extract its imaginary values from off-diagonal blocks. Fig. 4.6 reports another numerical experiment of using CR form to calculate first-order and second-order derivative of the inverse of a 3×3 matrix: $f(\mathbf{X}) = \mathbf{X}^{-1} \in \mathbb{R}^{3\times3}$ w.r.t $X_{2,2}$ (i.e. the element resides at the second row and second column of $\mathbf{X}$). In this example, we generate a random 3×3 non-singular matrix, and compute its inverse matrix analytically. The exact formulation of the first-order and second-order derivative of matrix inverse is:

$$\frac{\partial f}{\partial X_{2,2}} = -\mathbf{X}^{-1}\frac{\partial \mathbf{X}}{\partial X_{2,2}}\mathbf{X}^{-1}, \quad \frac{\partial^2 f}{\partial X_{2,2}^2} = -2\mathbf{X}^{-1}\frac{\partial \mathbf{X}}{\partial X_{2,2}}\mathbf{X}^{-1}\frac{\partial \mathbf{X}}{\partial X_{2,2}}\mathbf{X}^{-1},$$

and it is used as the reference.

The relative error of the numerical derivative computed using CR form as well as using the finite difference is plotted. We can see from Fig. 4.5 that CR form based CSFD and MCSFD also have excellent accuracy, while the regular finite difference still suffers with the subtractive cancellation.

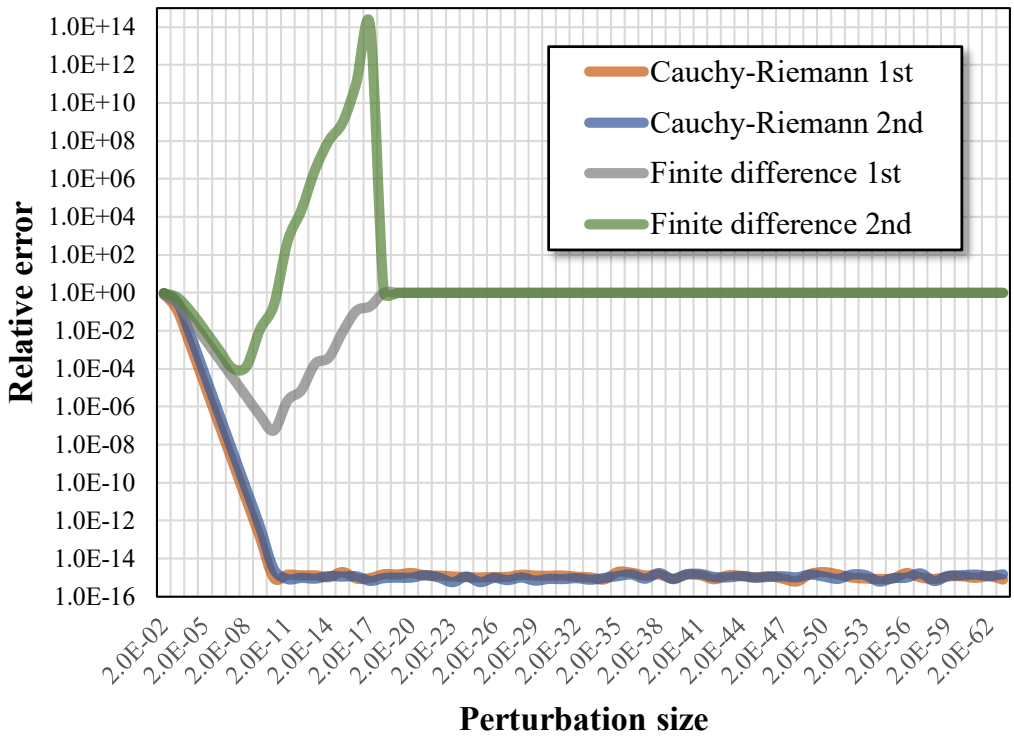


Figure 4.6: Cauchy-Riemann formula allows us to use existing linear algebra libraries to compute high-order numerical derivative without referring to an explicit complex promotion. In this example, we compute the first- and second-order derivative of 3×3 matrix inverse. CR-form based CSFD and MCSFD still have excellent accuracy compared with regular finite difference.

4.8 Experimental Results

We implemented CSFD/MCSFD on a desktop computer with an `Intel i7 8700K` CPU and 32 GB memory. Both regular complex arithmetic and the generalized multicomplex arithmetic were implemented using `C++ double` precision (64 bit on a x64 computer). While we believe CSFD/MCSFD will be useful for many graphics problems, in this chapter we demonstrate its applications in modeling and simulating elastic objects. Unless specified, we set h as 1.0×10^{-40} in our experiments. Our general observation is that one can fully rely on CSFD/MCSFD-based derivative without

	CSFD	`Adept` (s)	`CppAD` (s)	`ADOL-C` (s)	`ad` (s)
1st	114 ms	11.1 (97×)	8.2 (72×)	1.4 (13×)	72.1 (632×)
2nd	242 ms	NA	11.2 (49×)	5.9 (24×)	80.3 (332×)
3rd	768 ms	NA	NA	51 (62×)	NA

Table 4.2: Time statistics of computing first- (**1st**), second- (**2nd**), and third-order (**3rd**) derivatives for 1 million times of the function: $f(x) = e^x/(x^4 + x^2 + 1)$ using CSFD/MCSFD and some popular AD packages.

any accuracy concerns. Performance-wise, accelerated CSFD/MCSFD is almost as efficient as analytic derivatives. We also compared CSFD/MCSFD with some widely used AD packages. While both CSFD/MCSFD and AD produce accurate results in well-conditioned problems, CSFD/MCSFD excels at its robustness for nonsmooth functions, high-order generalization, and tensor extension. Accelerated CSFD/MCSFD is also much faster: it is over 20× faster than `C++` based AD packages and $\sim$ 300× faster than `Python` based AD packages.

Figure 4.7: The Armadillo model falls quickly and hits a glassy rod. Due to the sharp collision, gradient descent method [5] yields artifact because the residual is not sufficiently reduced. Regular finite difference method crashes instantly. Newton's method with MCSFD-based Hessian yields the same result as using the analytic Newton. Newton-PCG with CSFD-based directional derivative also has the same result.

Comparison with AD packages In the first experiment, we would like to examine the efficiency of our accelerated CSFD/MCSFD as well as some widely used AD packages including `Adept` [154], `CppAD` [164], `ADOL-C` [167], and `ad` [168]. The first

three libraries are in `C++`, and `ad` is a famous `Python` package. We record the time performance for evaluating derivatives (for 1 million times) of the function: $f(x) = e^x/(x^4+x^2+1)$ at $x = 4$ (i.e. the one used in Figs. 4.2 and 4.5). The computation time for the analytic first- and second-order derivatives is 104 ms and 238 ms respectively, which is quite close to our CSFD/MCSFD taking 114 ms and 242 ms. This function is smooth and differentiable, and all AD packages return accurate first-order derivative results successfully. Yet, our method is massively faster than AD packages as reported in Tab. 4.2. In general, the accelerated CSFD/MCSFD is dozens times faster than `C++` based AD packages and hundreds times faster than `Python` based ones. In this experiment, `Adept` does not support second-order derivative natively. `CppAD` and `ad` only support high-order derivative up to the second order. `ADOL-C` is the most sophisticated package, which has dedicated sub-routines for second-order and high-order derivatives. Nevertheless, it is still more than one order slower than our method. `ADOL-C` becomes even slower for higher-order derivatives as it calls the first-order routine repeatedly for high-order cases (i.e. with its `forward()` routune). `Python` package is the slowest.

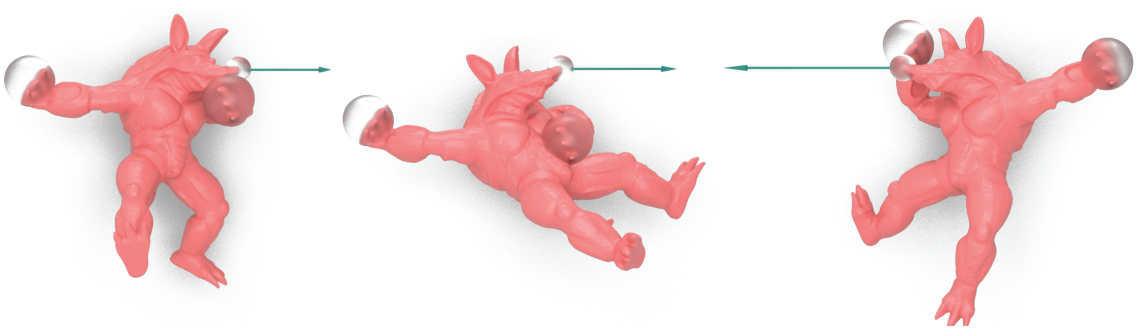


Figure 4.8: We simulate a Neo-Hookean Armadillo model using Newton's method. The Armadillo has 69,074 elements. The gradient and Hessian of the target function f (i.e. Eq. (4.35)) is approximated using numerical CSFD/MCSFD. The result is identical to the one computed using analytic gradient and Hessian.

We also assess the robustness of AD packages for nonsmooth functions. Consider

	CSFD	`Adept` (s)	`CppAD` (s)	`ADOL-C` (s)	`ad` (s)
StVK 1st	9 ms	1.1 (122×)	0.8 (90×)	0.7 (78×)	7.1 (786×)
StVK 2nd	101 ms	NA	5.4 (54×)	5.2 (52×)	29 (288×)
NH 1st	12 ms	1.2 (99×)	0.8 (66×)	0.8 (65×)	7.2 (580×)
NH 2nd	117 ms	NA	5.6 (48×)	5.7 (49×)	31 (268×)

Table 4.3: Computing the internal force and tangent stiffness matrix for $10k$ linear tetrahedral elements of StVK and Neo-Hookean materials. Similar to Tab. 4.2, our accelerated CSFD/MCSFD is much faster than AD packages.

$f(x) = \log^2\left(1 - \sqrt{(x-1)^2}\right)$. Its analytic first- and second-order derivative can be easily derived as:

$$f'(x) = -\frac{2(x-1)\log\left(1-\sqrt{(x-1)^2}\right)}{\left(1-\sqrt{(x-1)^2}\right)\sqrt{(x-1)^2}}, \tag{4.32}$$

and

$$f''(x) = -\frac{2\left(\log\left(1-\sqrt{(x-1)^2}\right)-1\right)}{\left(1-\sqrt{(x-1)^2}\right)^2}. \tag{4.33}$$

We notice that $x = 1$ is actually a singular point of the function. Without explicitly cancelling out $\sqrt{(x-1)^2}$ from Eqs. (4.32) and (4.33), AD packages that overload elementary arithmetic with differentiation rules tend to yield the division-by-zero error[2]. In this experiment, only `Adept` successfully returns the first-order derivative of this function, but It yields a `#IND` error for the second-order case. All other AD packages return either `NaN`, `#IND`, or `ZeroDivisionError` error. On the other hand, CSFD/MCSFD robustly handle this function derivative without any special treatments.

[2]We may be able to avoid this numerical instability of AD by expanding and simplifying the derivative function. But if we choose to do so, we are literally deriving the analytic formula of the derivative function, and why do we bother to use AD?

We observe similar results when applying CSFD/MCSFD and AD in deformable simulation computations. Tab. 4.3 lists the time performance of computing the internal force and tangent stiffness matrix for $10k$ linear tetrahedral elements of StVK and Neo-Hookean materials, which are the first- and second-order partial derivatives of the energy function. The analytic formulations of those two energies are known. We use `Vega` library [169] to compute the analytic force and stiffness matrix. For the StVK material, it takes 11.8 ms and 103.5 ms for the first- and second-order derivatives. For the Neo-Hookean material, the computation time is 12.1 ms and 112.5 ms respectively. This performance measure is close to our accelerated CSFD and MCSFD as shown in the table. In this experiment, most AD packages deliver correct results (expect for `Adept`) but they are all much slower than accelerated CSFD and MCSFD. It is also common, in practice, to resort to symbolic differentiation tools like `Mathematica` or `Maple`. For instance, `Maple` package takes ~ 1.5 s to yield the symbolic formulation of the first-order energy gradient for the StVK model, which consists of over 800 terms. Clearly, without further simplifications, directly importing them to the simulator is redundant and inefficient.

4.8.1 Application I: Accurate Nonlinear Optimization

Dynamic simulation of a deformable object requires solving a nonlinear system of the force equilibrium. For instance, the implicit Euler time integration scheme leads to:

$$\mathbf{M}\big(\mathbf{u}_{n+1} - \mathbf{u}_n - \Delta t\dot{\mathbf{u}}_n\big) = \Delta t^2\big(\mathbf{f}_{int}(\mathbf{u}_{n+1}) + \mathbf{f}_{ext}\big), \tag{4.34}$$

where $\mathbf{M}$ is the mass matrix. $\mathbf{f}_{int}$ and $\mathbf{f}_{ext}$ stand for the elastic internal force and the external force. The subscript $(\cdot)_n$ denotes the time integration step, and Δt is the time step size. $\mathbf{u}_{n+1}$ is the unknown displacement vector we want to compute. This equilibrium is often treated as an optimization problem known as its variational

form [118, 119] of:

$$\arg\min_{\mathbf{u}} f(\mathbf{u}), \quad f(\mathbf{u}) = \frac{1}{\Delta t^2}\left\|\mathbf{M}^{\frac{1}{2}}(\mathbf{u}-\mathbf{u}^*)\right\|^2 + E(\mathbf{u}), \tag{4.35}$$

where $\mathbf{u}^* = \mathbf{u}_n + \Delta t\dot{\mathbf{u}}_n + h^2\mathbf{M}^{-1}\mathbf{f}_{ext}$ is a known vector. E is the nonlinear elastic energy. Eq. (4.35) can be solved using the classic Newton's method, which approximates $f(\mathbf{u})$ with a quadratic form and calculates an incremental improvement of $\Delta\mathbf{u}$ as $\Delta\mathbf{u} = -\mathbf{H}\cdot\partial f/\partial\mathbf{u}$. Matrix $\mathbf{H}$ is the Hessian matrix, and it is the second-order partial derivative of f: $\mathbf{H} = \partial^2 f/\partial\mathbf{u}^2$. We simulate nonlinear dynamics of a Neo-Hookean Armadillo (with 69,074 elements) using Newton's method and drag its mouth back and forth. The gradient and Hessian of f are approximated with CSFD/MCSFD. The elastic energy density E of the Neo-Hookean material is

$$E_{NH} = \lambda(J-1)^2 + \mu(J^{-2/3}I_1 - 3), \tag{4.36}$$

where $J = |\mathbf{F}|$ is the determinant of the deformation gradient $\mathbf{F}$, and $I_1 = \mathtt{tr}(\mathbf{F}^\top\mathbf{F})$. λ and μ are Lamé constants. In our CSFD/MCSFD implementation, we treat E as a nested composite function $E_{NH} = E_1(J(\mathbf{F})) + E_2(I_1(\mathbf{F}))$. Snapshots of the deformed Armadillo are reported in Fig. 4.8. This animation is *identical* to the one obtained using analytic gradient and Hessian.

Alternatively, one may also use the Newton-PCG method, which replaces the direct solver used at each Newton iteration with an iterative PCG solver. As explained in [131], each Newton-PCG iteration calculates the product of $\mathbf{K}|_{\mathbf{u}_0}\cdot\mathbf{p}$, where $\mathbf{K}|_{\mathbf{u}_0}$ is the current tangent stiffness matrix at $\mathbf{u} = \mathbf{u}_0$, and $\mathbf{p}$ is a known displacement vector. This product can also be understood as the directional derivative of the energy function E and be numerically computed via CSFD as:

$$\mathbf{K}|_{\mathbf{u}_0}\cdot\mathbf{p} = \left.\frac{\partial^2 E}{\partial\mathbf{u}^2}\right|_{\mathbf{u}_0}\cdot\mathbf{p} = \nabla_{\mathbf{p}}\, E|_{\mathbf{u}_0} \approx \frac{\mathtt{Im}\big(E^*(\mathbf{u}_0 + h\cdot\mathbf{p}i)\big)}{h}. \tag{4.37}$$

As shown in Fig. 4.7, CSFD-based directional derivative is also highly accurate, which produces the same result of analytic Newton and MCSFD Newton. The regular finite difference crashes immediately when the Armadillo collides with the glassy rod.

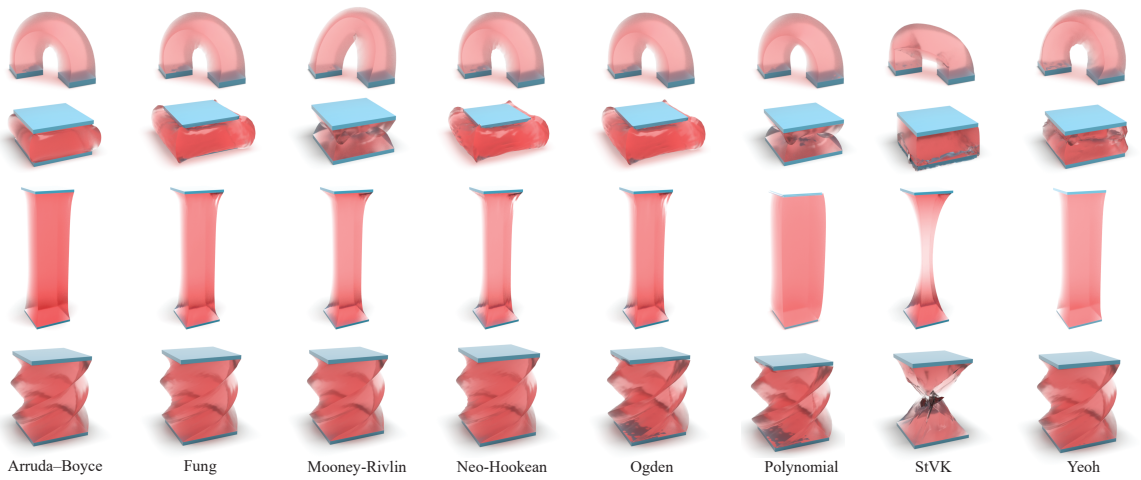


Figure 4.9: CSFD/MCSFD allows the user to easily simulate all kinds of hyperelastic materials. The figure reports the material behaviors under standard bending, compressing, stretching, and twisting tests of a box model with $14,678$ elements. From left to right, each column gives the result of Arruda–Boyce, Fung, Mooney-Rivlin, Neo-Hookean, Ogden, Polynomial, invertible StVK [6] (for improved stability), and Yeoh materials.

4.8.2 Application II: Intuitive Hyperelastic Simulation

For hyperelastic models, the form of the elastic energy (i.e. Eq. (4.35)) solely determines the deformed shape given inertial and external forces. Hyperelastic energy is typically defined based on three *isotropic invariants* of the deformation gradient: $I_1 = \mathtt{tr}(\mathbf{F}^\top\mathbf{F})$, $I_2 = \mathtt{tr}\big((\mathbf{F}^\top\mathbf{F})^2\big)$, and $I_3 = |\mathbf{F}^\top\mathbf{F}|^2$. Intuitively, I_1 measures the length change of the deformation; I_2 measures the area change of the deformation; and I_3 measures the volume change of the deformation. As long as the internal force $\partial E/\partial \mathbf{u}$ and the tangent stiffness matrix $\partial^2 E/\partial \mathbf{u}^2$ are available, the dynamic behavior of the deformable body can be simulated using standard FEM. The closed-form formulation of $\partial E/\partial \mathbf{u}$ and $\partial^2 E/\partial \mathbf{u}^2$ for some material models such as co-rotational model, StVK model, Neo-Hookean model are available in the literature [7, 117, 122]. However, there are many other materials such as Fung, Mooney-Rivlin, Ogden, Yeoh, Arruda Boyce models or the more general Polynomial model. Their energy structure

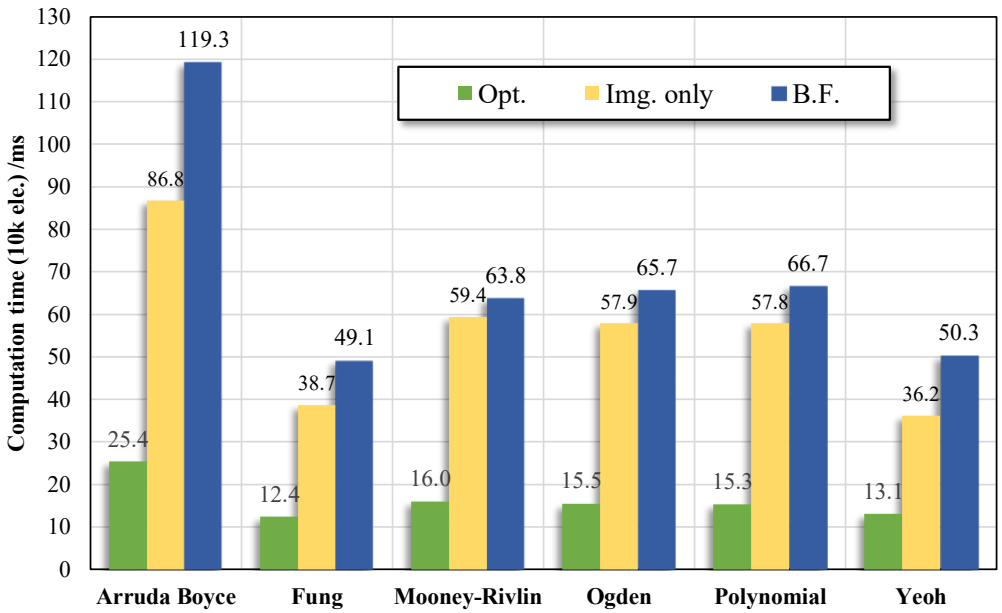


Figure 4.10: Timing information of CSFD/MCSFD derivative in simulating various hyperelastic materials. **Opt.** is the optimized CSFD/MCSFD computation time. **Img. only** is the time without computing the real part of the promoted energy functions. **B.F.** is the computation time using a brute-force CSFD/MCSFD implementation.

can be easily followed, but deriving the actual formulation of force and stiffness matrix prevents these materials from being more widely employed by the graphics community. CSFD and MCSFD allow us to conveniently simulate hyperelastic materials with light-weight implementation efforts. As reported in Fig 4.9, we simulate all of those materials using CSFD/MCSFD under standard bending, compressing, stretching and twisting tests. In this experiment, we use invertible StVK energy [6] to improve the stability of the regular StVK material. Timing information of different CSFD/MCSFD implementations is compared in Fig. 4.10.

In many situations, the user wants to use customized materials for specific needs in an animation scenario. For instance Smith and colleague [7] proposed a new Neo-Hookean-like hyperelastic energy for a stable integration and volume preservation

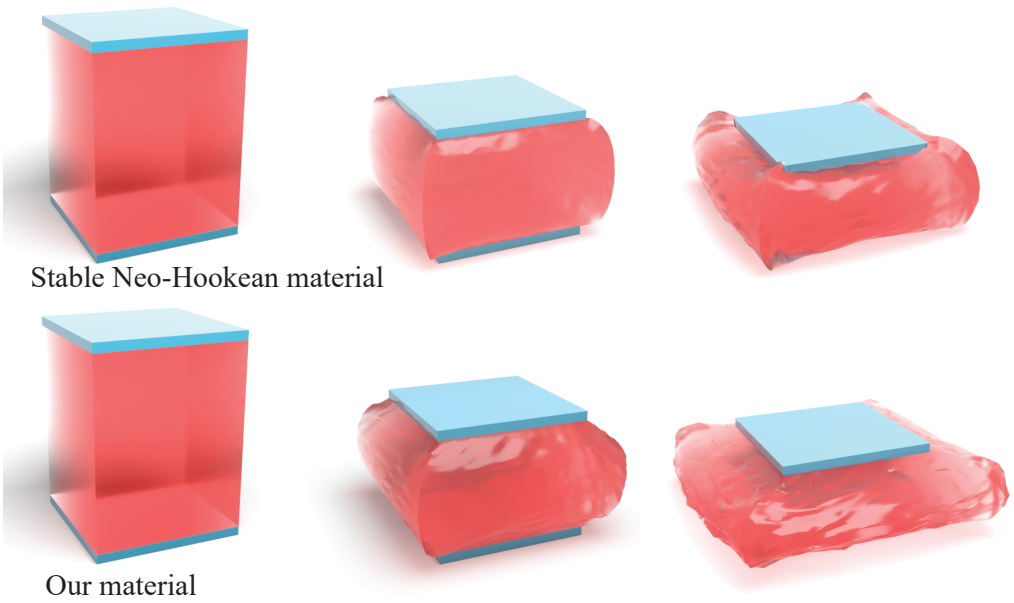


Figure 4.11: Our new material (Eq. (4.38)) with a more aggressive volume penalty term is able to better preserve the volume of this jelly box during the compression than the stable Neo-Hookean material [7].

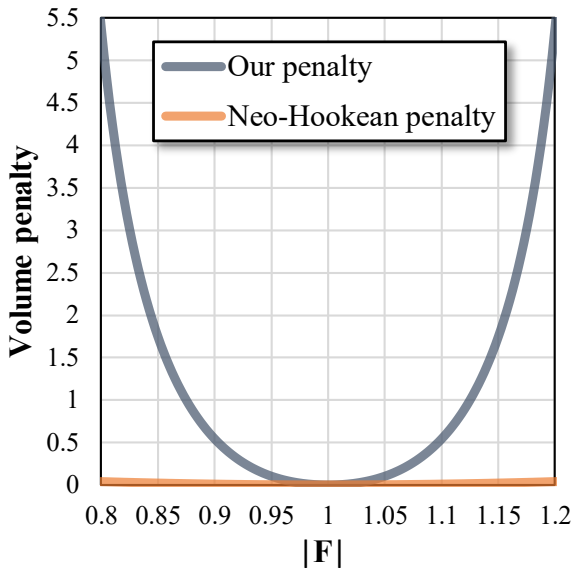


Figure 4.12: We design a new volume penalty term of $\log^2(1-4(J-1)^2)$, which yields much bigger internal forces when $J = |\mathbf{F}|$ deviates from 1 than the regular Neo-Hookean volume penalty of $(J-1)^2$ does.

under large deformation. Using CSFD/MCSFD, users can freely explore various such energy densities without tedious derivations for internal force and Hessian. For instance, we design a new hyperelastic model:

$$E_{volume} = \mu(J^{-2/3}I_1 - 3) + \frac{\lambda}{2}\log^2\left(1 - 4(J-1)^2\right). \tag{4.38}$$

As plotted in Fig. 4.12, E_{volume} triggers a much stronger resistance force to when $J = |\mathbf{F}|$ deviates from 1 and thus, better preserves the volume (i.e. see Fig. 4.11). In this example, the rest-shape volume of the jelly box is 0.64. After compressing its height by 65%, the new volume of the jelly box becomes 0.63 with E_{volume} and 0.61 with the stable Neo-Hookean material [7]. While numbers look close, we can clearly see that the compressed box is much wider spread out with E_{volume}.

CSFD/MCSFD can deal with even more complicated energies. Another example is reported in Fig. 4.13. In this example, we use an example-based hyperelastic energy as in [123], which has two target shapes, each of which embeds a smiling face or a sad face on the surface. We design this energy to be the function of the bending orientation so that corresponding internal forces arise when the box is bent to a certain direction. CSFD/MCSFD frees us from formulating the animation system and to quickly toy with many of such examples to achieve more interesting animations.

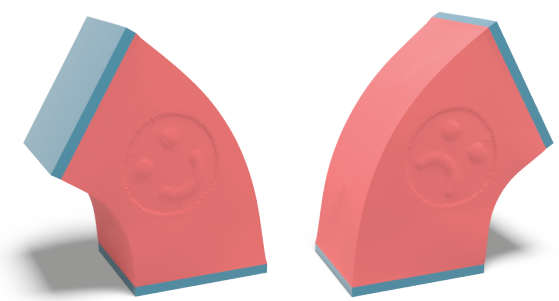


Figure 4.13: Example-based hyperelastic energy can also be easily handled with CSFD/MCSFD. We make the energy a the function of bending angle so that a smiling face appears when the box bends to left and a sad face appears when the box bends to right. This box model has 14,678 elements.

For a customized material, it is possible that the user-specified energy has some singularities due to its complex formulation. In this case, AD packages, regardless

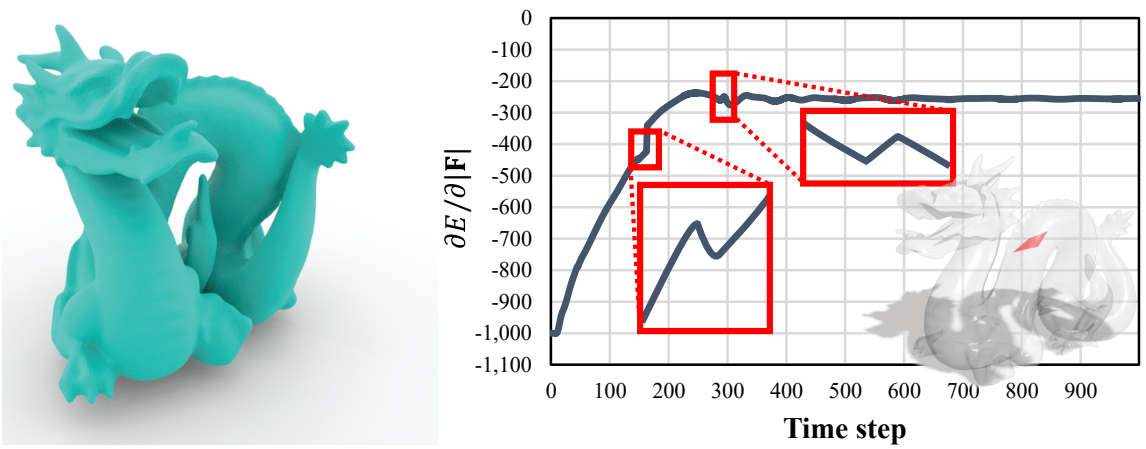


Figure 4.14: Complicated energy formulation as Eq. (4.39) could hide singularities that are unfriendly for AD. CSFD/MCSFD can tackle this issue robustly.

of their slow performance, could even fail the simulation if any element reaches the singularity. To better elaborate this, we create another energy with the form of:

$$E_{singular} = \mu(J^{-2/3}I_1 - 3) + \lambda(J-1)^2 + \sqrt{\cos^2 4(J-1)} - 1. \tag{4.39}$$

As shown in Fig. 4.14, if we slowly bend the dragon with this material using AD, the system crashes with the division-by-zero error when an element hits the singular point. CSFD/MCSFD is robust in such situations. Referring to Eq. (4.7), it is easy to see that as long as $f(x_0)$ exists, $f^*(x_0 + hi)$ also exists because it is perturbed orthogonally towards the real domain and never touches the real-valued singularity. Therefore, CSFD always returns a well-estimated derivative value because h is also nonzero.

4.8.3 Application III: Expressive Model Reduction

Model reduction is a widely-used technique to produce real-time deformable animation. This technique needs a pre-built subspace, which defines all the possible deformations of the deformable body. The standard method for subspace construction

is based on the modal analysis [10], which provides the optimal vibrational modes around the rest shape. For nonlinear models, we need to compute *derivative modes* that first-order approximate a low-frequency nonlinear vibration [11,131]. Computing derivative modes requires the calculation of the force Hessian (i.e. the third-order derivative of E). Therefore, this powerful technique is normally used only for the StVK material, whose stiffness matrix is quadratic w.r.t to the displacement vector. Applying nonlinear model reduction to other materials using modal derivative is less exploited due to the barrier of computing high-order energy gradients. CSFD/MCSFD allows us to build expressive and compact subspace easily for any given hyperelastic material. Fig. 4.15 shows snapshots of a real-time simulation of six falling dinosaur models using 30 first-order modal derivatives. Each dinosaur model has $356,48$ elements, and they are of Fung, Mooney-Rivlin, Ogden, Yeoh, Arruda Boyce, and Polynomial materials. Yang and colleagues [131] introduced a method that generalizes modal derivative to higher-order nonlinear shape approximation. This method can also be readily implemented with MCSFD. As shown in Fig. 4.16, we apply a circular force to bow the dinosaur model. Second-order modal derivatives are able to capture extreme bending effects. In this experiment, the hyperelastic material of Eq. (4.38) has a strong volume preserving term, which prevents this material from being extremely bent as other materials under the same external forces.

4.8.4 Application IV: Convenient Inverse Design

A lot of design problems tweak a collection of parameters to make sure that the simulated result matches certain specific measures like the maximum stress, deflection magnitude and so on. While there are many techniques (e.g. the well-known adjoint method) that are capable of handling those problems, we show that CSFD/MCSFD is also a convenient alternative to deal with inverse simulations.

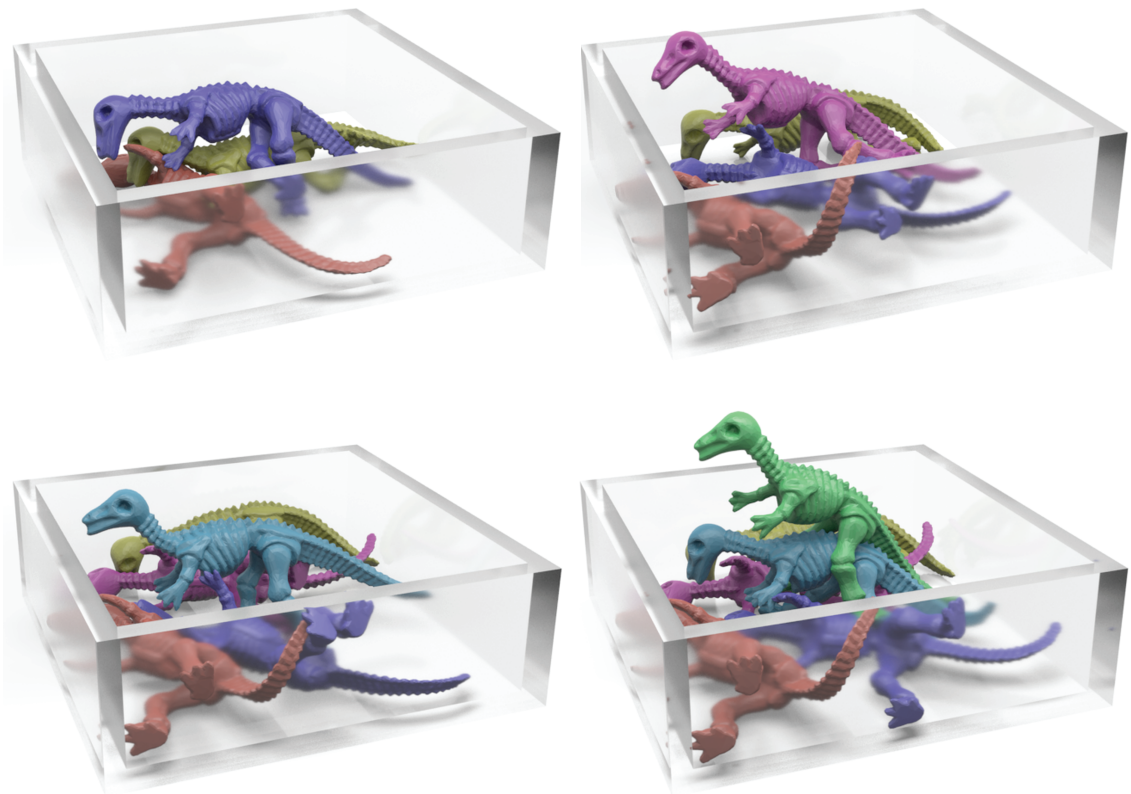


Figure 4.15: Real-time simulation of six falling dinosaur models using modal derivative (30 modes for each dinosaur). The first-order derivative modes are computed using CSFD, and we use Fung, Mooney-Rivlin, Ogden, Yeoh, Arruda–Boyce and Polynomial materials for each dinosaur.

In Fig. 4.1, we show an example where the user wants to adjust the linear vibration frequencies of a bridge for a given external wind field by changing three primary geometry parameters: length l, width w and the height of the arch top t. For an intuitive visualization of a frequency pattern, our system allows the user to apply this wind field to a standard rectangular beam (with two ends fixed) and to change its geometry/material to generate a preferred vibration pattern (see Fig. 4.17). The principle vibration of a linear structure under a given direction $\mathbf{u}$ is described by the *Rayleigh quotient* defined as $\omega^2 = \mathbf{u}^\top \mathbf{K}\mathbf{u}/\mathbf{u}^\top \mathbf{M}\mathbf{u}$. The wind is modeled as an

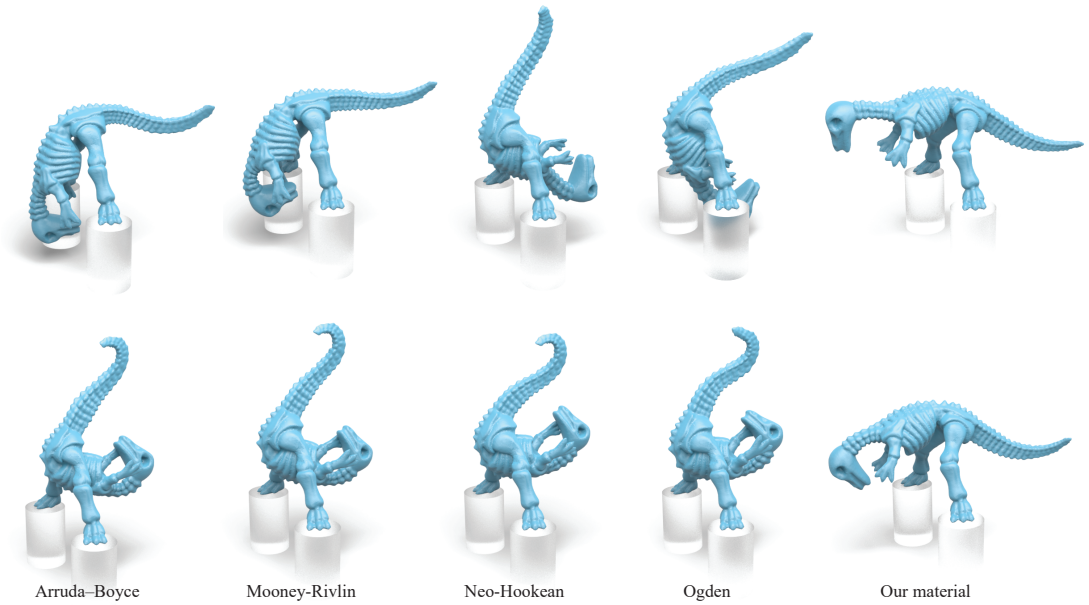


Figure 4.16: MCSFD allows us to compute higher-order modal derivatives that capture extreme bending effects of the dinosaur model (using 30 second-order derivative modes). Interestingly, the hyperelastic energy of Eq. (4.38), because of its strong resistance to volume change, cannot be bent as hard as other materials under the same external force.

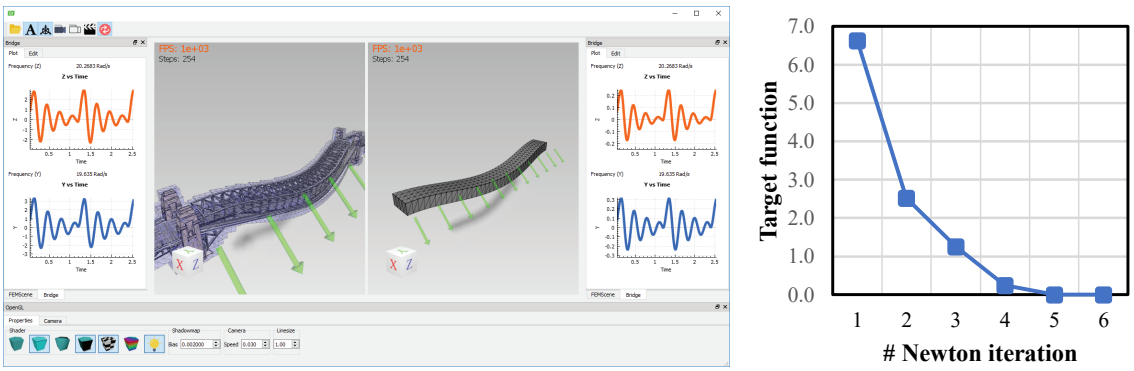


Figure 4.17: We develop a system with an intuitive interface for the linear frequency design (left). The error reduces quickly along Newton iterations, with the Hessian accurately computed from MCSFD. (right)

acceleration field $\mathbf{a}$ meaning $\mathbf{u} = \mathbf{K}^{-1}\mathbf{M}\mathbf{a}$. As a result, the frequency design procedure can be formulated as an optimization problem of:

$$\arg\min_{l,w,t} f, \quad f(l,w,t) = \left\| \omega^{*2} - \frac{\mathbf{a}^\top \mathbf{M}\mathbf{K}^{-1}\mathbf{M}\mathbf{a}}{\mathbf{a}^\top \mathbf{M}\mathbf{K}^{-1}\mathbf{M}\mathbf{K}^{-1}\mathbf{M}\mathbf{a}} \right\|^2, \tag{4.40}$$

where ω^{*2} is our target frequency. $\mathbf{M} = \mathbf{M}(l,w,t)$ and $\mathbf{K} = \mathbf{K}(l,w,t)$ are tensor functions of the unknown geometry parameters l, w, t to be optimized. In this example, we use the CR form (Eq. (4.31)) to promote $\mathbf{M}(l,w,t)$ and $\mathbf{K}(l,w,t)$, and Newton's method is used to solve Eq. (4.40). Thanks to the accurate Hessian obtained by MCSFD, our solver quickly finds the optimal geometry only with few iterations.

Chapter 5

Neural Network-based Nonlinear Deformation

5.1 Introduction

In this chapter, we present a framework combined neural network and warping method together to simplify and accelerate the traditional FEM based frameworks [170]. In the past ten years, the FEM based frameworks become more and more popular due to its versatility of encoding various material behaviors. With the prescribed external force $\mathbf{f}_{\texttt{ext}}$, the dynamic equilibrium is forwarded by solving a high-dimensional nonlinear system of $\mathbf{f}(\mathbf{u}) = \mathbf{f}_{\texttt{ext}}$[1] at each time step. Most nonlinear solvers start with an initial guess of the unknown displacement $\mathbf{u}$ and iteratively refine the result until the system converges in order to calculate the deformed model shape. While conceptually straightforward, the requirement of repetitive evaluations of the nonlinear internal force $\mathbf{f}_{\texttt{int}}$ or/and its gradient $\partial\mathbf{f}_{\texttt{int}}/\partial\mathbf{u}$ makes the simulation

[1]Here $\mathbf{f}(\mathbf{u})$ is the general internal force consisting of standard nonlinear internal force, damping force and inertial force, and it is a function of the unknown displacement $\mathbf{u}$ after time derivative terms are linearized based on the chosen time integration method.

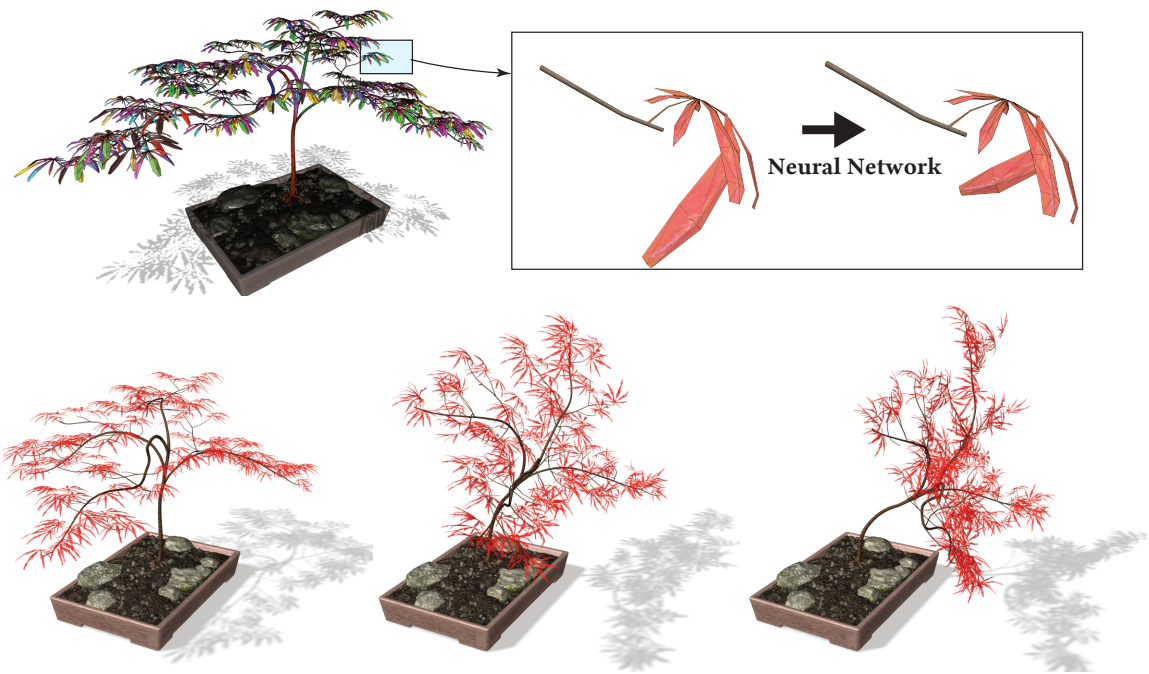


Figure 5.1: NNWarp is a data-driven neural network based nonlinear deformable simulator. By learning from full FEM simulation poses, it yields more accurate results than existing warping methods. We design three compact contextual features making the network training highly re-usable. In this example, the maple bonsai model consists of 255, 552 elements, and is decomposed into 1, 771 domains. A single net trained using a regular beam handles local dynamics for all the domains. High-quality animations with well-preserved local high-frequency deformations are produced at a near-interactive rate (5 FPS) without using model reduction.

rather computational expensive.

Recently, the rapid development of the computing hardware pushes forward the frontier of machine intelligence to an unprecedented extend, and we have witnessed tremendous successes of utilizing carefully constructed neural networks (NNs) [171] in many classic computing problems like language processing [172], speech recognition [173, 174], object tracking [175, 176] etc. With the support of sufficient ground truth data, an NN serves as a black box mapping its input to the output without the necessity of an explicit mathematical formulation. Since the FEM simulation is

able to provide us as many as needed noise-free data, can we also exploit NNs to deal with deformable simulation?

At the first sight of the question, the answer seems to be positive because deformable simulation is essentially the reconstruction of the *force-displacement relation* of an elastic body, and NNs are known good at expressing complex nonlinear relations [177,178]. However, this problem is challenging in practice because the nonlinear force-displacement relation varies significantly (and intrinsically) under different simulation configurations such as domain geometries, discretizations, boundary conditions, constitutive laws etc. If one chooses to build a network incorporating all the possible input permutations, the network would indubitably be an extremely huge one. Even we manage to generate sufficient training data and optimize the network parameters to a reasonable level. A single forward pass of the network itself could take a longer time than running a regular FEM simulator due to the complexity of the network.

In this chapter, we present a method, named *NNWarp*, to leverage neural networks to tackle intricate force-displacement relations of different nonlinear materials with a simple and light-weight network. As the name implies, our strategy is not to link the standard input ($\mathbf{f}_{\texttt{ext}}$) and output ($\mathbf{u}$) of deformable simulation via a neural network directly. Instead, we map or *warp* a simplified constitutive law $\mathcal{L}_0$ to a more complex and nonlinear one $\mathcal{L}_1$ using NNs. It is expected that, the calculated displacement under $\mathcal{L}_0$ well encapsulates simulation configurations like force magnitude, domain tessellations and boundary conditions so that the remaining warp is local, and can be well fit by a simple net. To this end, we choose to use the linear elasticity for $\mathcal{L}_0$. The linear elasticity has long been used to describe small-scale deformations (i.e. the infinitesimal strain theory). It is based on the Cauchy strain tensor, which is the first-order Taylor approximation of the full Green tensor. Besides, because linear elasticity has a constant stiffness matrix, setting it as $\mathcal{L}_0$ makes NNWarp *polynomial* faster than another other nonlinear constitutive models with the same number of

simulation DOFs during the run-time simulation .

NNWarp uses a single node-wise NN to correct the nodal linear deformation to the corresponding nonlinear one. In other words, it takes the linear nodal displacement as the input, and outputs a corrective displacement fix to warp the linear result to be a nonlinear one. From this perspective, our method is conceptually similar to stiffness warping [179] and modal warping [12], in which a linear solver is used after rotating the deformed shape back to its undeformed orientation. We augment the input of per-node linear displacement with three novel discriminative features, namely the *geodesic*, *potential* and *digression*. We find that with these three descriptors, NNWarp becomes fairly shape- and tessellation-independent, and the network trained with a simple model can be used to warp deformable bodies of distinctively different geometries making our network training highly re-usable. This important advantage is further enhanced when combined with the substructuring method [180], where we decompose the input model into multiple convex domains, and run NNWarp on each domain separately. For instance, all the experiments reported in the chapter (except Fig. 5.10) are based on the network trained using a simple rectangular beam. NNWarp is fast at both training stage and simulation stage. We utilize the rotation invariant property of local deformation and compress the training set by at least an order. During the simulation, as NNWarp only needs to perform a pre-factorized linear solve at each time step, it is able to handle large-scale models interactively.

5.2 Related Work

The concept of neural network based learning can be dated back to late 1980s [181] in the machine learning community. Empowered by recent hardware advance, neural networks of various architectures and deeper depths have been harnessed to solve many long-standing computer vision problems such as recognition [182, 183], classi-

fication [184–187] and segmentation [188–190]. Some existing methods are able to match or even beat human's vision perception system e.g. see the report from the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [191]. Given sufficient training data, Deep neural networks (DNNs) provide a general "template" for the user to learn the input-output correspondence, which could be otherwise difficult or even impossible to be analytically formulated.

Learning for animation Indeed, the idea of learning is not new to computer animation, and it is also widely-known as *data driven* methods [192]. For the **cloth animation**, low-resolution simulation can be enriched by using pre-computed high-resolution results with detailed wrinkles [193, 194]. Wang et al. [195] built a piecewise linear stretching and bending model based on measured data to better depict the nonlinear dynamics of different cloth materials. Miguel et al. [196] further enhanced this framework and recorded more deformation behaviors of the cloth simulation. Kim et al. [197] proposed a method to compress a large pre-simulation dataset so that these poses can be used at run time to improve the inertial cloth deformation. Following the similar idea, Xu et al. [198] blended pre-computed cloth shapes to directly synthesize the cloth deformation using the sensitivity analysis. Learning-based methods have also been popular for **motion and control** i.e. the reinforcement learning [199–202]. NNs provide a convenient approach for further improving the learning effects [203]. Following this direction, Liu et al. [204] employed the deep Q-network to reorder existing control fragments and created necessary responses to unseen disturbances. Peng et al. [205] used an NN to train a high-level controller and a low-level one, which achieved robust locomotion coordinately. Holden et al. [206] designed a phase-functioned neural network, whose weights are computed using a cyclic function. For **solid modeling**, learning is also a powerful tool, which allows the user to obtain actual physical parameters based on captured point cloud sequences [207]. Xu and Barbič [208] fine-tuned the damping model based on a few example deformations. Kim

et al. [209] combined the physics-based simulation and data-driven to produce realistic soft tissue animation. Jones et al. [210] used the similar idea to simulate plastic deformation with a skinning-alike method. An et al. [211] proposed a learning-based numerical procedure named Cubature to efficiently evaluate the internal force and the force gradient during reduced deformable simulation. Deep learning also benefits the **fluid animation**. For instance, Ladicky et al. [212] proposed a random forest based regression method to accelerate fluid simulation by predicting the kinematic configurations of particles based on a large training set. Chu and Thuerey [213] used the convolutional neural networks (CNN) to extract necessary features to augment a coarse simulation and add back high-frequency details.

Nonlinear deformable simulation Physics-based deformable simulation has been an active research topic in graphics and animation since the exemplar work by Terzopoulos et al. [116]. While particle-based methods [33, 83, 214] or mass-spring systems [118, 215] are also legit, FEM becomes more widely-used [122] for solid simulation. Wang et al. [216] proposed a strain limiting method to increase the numerical stability for stiff deformable bodies. Alternatively, Irving et al. [6] tweaked the principle stress to resolve degenerated elements from extreme deformations. Forming the deformable simulation as a nonlinear optimization procedure, Hecht et al. [217] used an incremental Cholesky factorization scheme to lower the frequency of matrix re-factorization during the simulation. Zhu et al. [218] adopted the multi-grid method to simulate high-resolution deformable volumes. Bouaziz et al. [219] introduced a robust local-global iterative solver named *projective dynamics*. This idea later was generalized as the ADMM solver [220] and synergized with Chebyshev [5, 221], L-BFGS [121] and GPU Gauss-Seidel [74]. Accelerating nonlinear simulation can also be achieved by pre-computed models, for instance using modal analysis [11, 58, 131] or recent fullspace simulations [222]. Also known as model reduction methods, it is assumed that the deformed shape be a linear combination of those pre-computed

poses or *modes* so that the simulation can be projected into the spanned subspace. In an asymptotic sense however, model reduction is not better than regular simulation as the time complexity remains cubic w.r.t. the number of simulation DOFs.

NNWarp and existing warping methods In this chapter, we re-investigate this classic animation problem of nonlinear deformable simulation from a data-driven point of view by shaping it as a nonlinear regression using the neural network. Unfortunately, the full spectrum of the force-displacement relation is complex and sensitive to the variance of simulation settings. For instance, modifying the boundary condition (the anchor nodes of an FE mesh) could completely alter the deformed shape even with other simulation parameters unchanged. Besides, the dynamic simulation is essentially 4D – the kinematic status of the deformable body does not only depend on its current external stimuli but also on its historic motion trajectory. To circumvent these two practical obstacles, we forge our regression based on the simulation result obtained using the linear elasticity. This idea is not new in graphics. An epic example would be the stiffness warping [179], which re-used the linear stiffness matrix by un-rotating the external force back to the model's rest shape orientation. Similarly, modal warping [12] and rotation-strain coordinate [223, 224] embedded a local coordinate frame at each node/element to relieve the artifacts of the linear elasticity under rotational deformation. This idea was also used for geometrically constructing nonlinear modes [60]. These geometric warping techniques have been proven effective for animation editing [129, 225], which requires performing high-dimensional space-time optimization.

Solving the linear elasticity encodes many simulation parameters such as boundary condition, tessellation resolution, external force etc. into the resulting linear displacement vector. On the top of this, we train a neural network to further correct the result to be a plausible and nonlinear one without worrying about accommodating all the simulation settings into the net. Our training is re-usable – an NN trained

using a regular model of few thousand elements can be used to handle a wide range of geometrically complex deformable bodies. During the simulation, because the system matrix for the linear elasticity is constant and pre-factorized, we obtain $\mathbf{O}(N^2)$ run-time complexity *in fullspace*, which is polynomially faster than existing nonlinear solvers.

5.3 Contextual Feature Vector

The underlying mathematical relations between external forces and displacements of elastic bodies could be intrinsically changed under different simulation settings, and it is impossible in practice to encode the entire simulation configuration into a feature vector and feed to a neural network. Therefore, the primary challenge we are facing is to figure out an *informative* and *compact* feature vector as the input. Informative refers to the discriminability of the feature so that an irrelevant training instance does not interfere. Compact means the feature should also be general so that the built network is small and light-weight. In this section, we start with a short review of the deformable model, pointing out that while the simulation is sophisticated, the linear-nonlinear deformation map of a small local volume is actually smooth. Bearing that in mind, we show that our *heuristic* feature vector augments the extracted kinematic information and produces plausible results.

5.3.1 Deformable model: a quick review

Given an arbitrary material point x on the deformable body, its deformation gradient $\mathbf{F} \in \mathbb{R}^{3\times3}$ is computed as $\mathbf{F} = \partial\mathbf{x}/\partial\bar{\mathbf{x}}$, where $\bar{\mathbf{x}}$ and $\mathbf{x}$ denote its rest shape position and the deformed position. Alternatively, we can also express $\mathbf{x}$ using its displacement $\mathbf{u}$ as $\mathbf{x} = \bar{\mathbf{x}} + \mathbf{u}$. Let $\mathbf{G} = \partial\mathbf{u}/\partial\bar{\mathbf{x}}$ and we name this 3 by 3 tensor as

displacement gradient tensor. It is easy to verify that $\mathbf{F} = \mathbf{G} + \mathbf{I}$. Under the linear elasticity, the deformation is described using the Cauchy strain: $\widetilde{\epsilon} = \frac{1}{2}(\mathbf{G} + \mathbf{G}^\top)$, and the strain energy density $\widetilde{\Psi}$ is:

$$\widetilde{\Psi} = \frac{k}{2(1+\nu)}\widetilde{\epsilon} : \widetilde{\epsilon} + \frac{k\nu}{2(1+\nu)(1-2\nu)}\mathtt{tr}^2(\widetilde{\epsilon}). \tag{5.1}$$

Here k and ν are the Young's modulus and Poisson's ratio. As $\widetilde{\Psi}$ is a quadratic function of $\mathbf{G}$, the corresponding Piola stress becomes a linear function of $\mathbf{G}$:

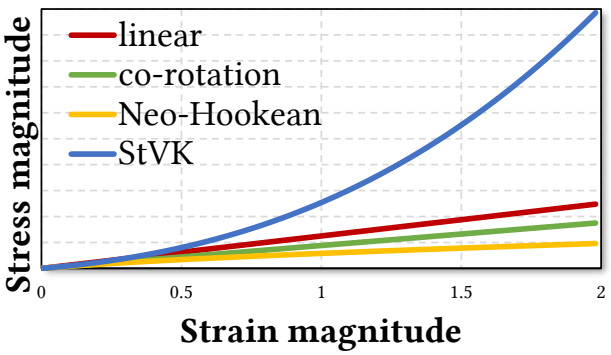
$$\widetilde{\mathbf{P}} = \frac{k}{2(1+\nu)}\left(\mathbf{G} + \mathbf{G}^\top\right) + \frac{k\nu \cdot \mathtt{tr}(\mathbf{G})}{(1+\nu)(1-2\nu)}\mathbf{I}. \tag{5.2}$$

For most other hyperelastic materials, the deformation is actually described with the Green strain: $\epsilon = \frac{1}{2}(\mathbf{F}\mathbf{F}^\top - \mathbf{I}) = \widetilde{\epsilon} + \frac{1}{2}\mathbf{G}\mathbf{G}^\top$. One can see that the Cauchy strain used in linear elasticity is simply the linear portion of the Green strain. Take the St. Venant-Kirchhoff (StVK) material an example, whose strain energy density is formulated by replacing $\widetilde{\epsilon}$ by ϵ:

$$\Psi_{\mathtt{StVK}} = \frac{k}{2(1+\nu)}\epsilon : \epsilon + \frac{k\nu}{2(1+\nu)(1-2\nu)}\mathtt{tr}^2(\epsilon), \tag{5.3}$$

which is a fourth-order polynomial of the displacement gradient $\mathbf{G}$, and its stress is cubically related to $\mathbf{G}$. With the help of FEM, the differential strain-stress relation is integrated and becomes the macroscopic force-displacement relation that we are interested in.

In reality, the *magnitude* of a deformation, which may be somewhat understood as $|\mathbf{G}|$, is typically small. For instance, doubling the length of an elastic rope by stretching is considered a very large deformation where $|\mathbf{G}| = 1$. In addition, strain-stress curves for various materials are all



aligned at the origin (a zero strain yields a zero stress) and within the same monotonically increasing interval (a larger strain yields a larger stress). This implies that the strain-stress curves of the linear elasticity and a nonlinear elasticity do not fundamentally differ from each other in regular deformable simulations. An example is given in the inset figure, where we plot the strain-stress curves of the linear, co-rotation, StVK and Neo-Hookean laws under a rotation-free linear stretch.

Geometric warp In fact, the dominant factor drives the linear elasticity away from a nonlinear counterpart is not the material nonlinearity, but the geometry nonlinearity. This is because a rigid, deformation-free rotation leads to a non-zero Cauchy strain, which produces unrealistic deformation effects. Under this consideration, the modal warping (MW) technique [12] embeds each node on the mesh a local frame. The curl of local displacement field around the i-th node is calculated: $\mathbf{w}_i = \nabla \times \mathbf{u}_i$. If it takes a unit time to displace node i from $\bar{\mathbf{x}}_i$ to $\bar{\mathbf{x}}_i + \mathbf{u}_i$, $\mathbf{u}_i$ also represents its velocity at $t = 1$. $\mathbf{w}_i$ can then be understood as its angular velocity at the same moment. Based on this assumption, MW linearly ramps the angular velocity from the rest shape to the current time instance t and calculates a warp transformation as:

$$\mathbf{W}_{\texttt{MW}} = \frac{1}{t}\int_0^t \texttt{exp}\left(\frac{\tau}{t}[\mathbf{w}_i]_\times\right) \mathrm{d}\tau, \tag{5.4}$$

where $[\mathbf{w}_i]_\times$ is the skew symmetric matrix of $\mathbf{w}_i$. Similarly, one can use rotation-strain coordinate by decomposing the $\mathbf{G}_i$ into a skew symmetric part: $[\mathbf{w}_i]_\times = (\mathbf{G}_i - \mathbf{G}_i^\top)/2$ and a symmetric part: $\mathbf{S}_i = (\mathbf{G}_i + \mathbf{G}_i^\top)/2$ [223, 224]. The rotation-strain warp (RSW) transformation can then be computed treating $\mathbf{w}_i$ as an Euler vector:

$$\mathbf{W}_{\texttt{RSW}} = \texttt{exp}\left([\mathbf{w}_i]_\times\right)(\mathbf{S}_i + \mathbf{I}) - \mathbf{I}. \tag{5.5}$$

While not physically accurate, these geometric warping methods produce visually pleasing shapes and have been used in many time-critical graphics applications [129, 225].

5.3.2 Linear-nonlinear correspondence

NNWarp is inspired by the encouraging results from the existing warping methods. However, NNWarp does not explicitly assume a fixed nonlinear regression formula as Eqs. (5.4) or (5.5). Instead, we train an NN to obtain a more accurate regression based on full simulations. The key question here is how to determine what is the "right" nonlinear deformation that corresponds to the one calculated using the linear elasticity.

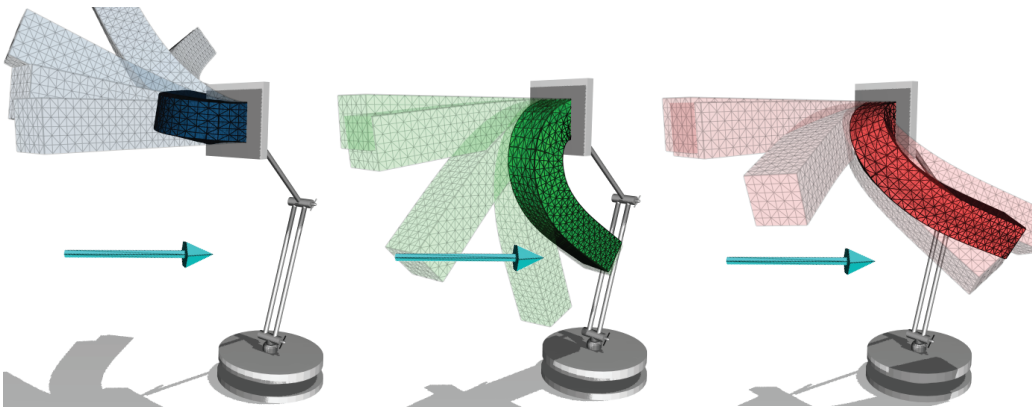


Figure 5.2: Different motion trajectories lead to different equilibrium shapes even under the same external force.

A naïve thought is to solve the quasi-static equilibrium of $\mathbf{f}_{\texttt{int}}(\mathbf{u}) = \mathbf{f}_{\texttt{ext}}$ for a deformable body under the same external force and boundary condition using the linear elasticity and a nonlinear constitutive model. Unfortunately, this method is only valid for small deformations. Under large external forces, the nonlinear system could reach multiple local minima of different shapes, and which one being reached is context-dependent i.e. up to the history of the deformation trajectory (Fig. 5.2). From a numerical point of view, the solution of $\mathbf{f}_{\texttt{int}}(\mathbf{u}) = \mathbf{f}_{\texttt{ext}}$ depends on the initial guess of $\mathbf{u}$ and the strategy of computing $\Delta\mathbf{u}$ during the iteration. The iteration may not converge to the global minimum if the starting guess is far away from it.

Our solution to this problem is to register a linear deformation *sequence* to a nonlinear one starting from the rest shape. Specifically, given an external force $\mathbf{f}_{\texttt{ext}}$, we compute a series of quasi-static linear deformation by solving the Euler-Lagrange equation with an increased mass damping so that the acceleration at each time step is negligible. Each time step yields a linear displacement vector $\widetilde{\mathbf{u}}$, and we estimate a local rotation for the i-th node as:

$$\mathbf{R}_i = \texttt{exp}\Big(\Big[\nabla \times \left(\mathbf{P}_i \mathbf{P}_i^\top\right)^{-1} \mathbf{P}_i^\top \mathbf{U}_i \Big]_\times \Big), \tag{5.6}$$

where columns in $\mathbf{P}_i$ and $\mathbf{U}_i$ are rest shape positions and displacements of neighbor nodes adjacent to i so that $\left(\mathbf{P}_i \mathbf{P}_i^\top\right)^{-1} \mathbf{P}_i^\top \mathbf{U}_i$ gives a least-square evaluation of $\mathbf{G}$ around the i-th node. The linear internal force at the current time step is $\widetilde{\mathbf{f}}_{\texttt{int}} = \mathbf{K}\widetilde{\mathbf{u}}$. Note that $\widetilde{\mathbf{f}}_{\texttt{int}} \neq \mathbf{f}_{\texttt{ext}}$ until the final equilibrium is reached due to the existence of the damping. Afterwards, the corresponding nonlinear deformation $\mathbf{u}$ is obtained by solving:

$$\min_{\mathbf{u}} |\mathbf{f}_{\texttt{int}}(\mathbf{u}) - \mathcal{R}\mathbf{K}\widetilde{\mathbf{u}}|, \tag{5.7}$$

where $\mathcal{R}$ is a block-diagonal matrix, and each of its 3 by 3 diagonal block is the estimated nodal rotation computed via Eq. (5.6). Eq. (5.7) is clearly an approximate because in practice when NNWarp is being used, we do not know what are the "ground truth" acceleration (which yields the inertia force) and the velocity (which yields the damping force) corresponding the a linear pose. Therefore, our best guess is the solve the nonlinear equilibrium of Eq. (5.7) according to its linear counterpart. We use Newton's method to solve Eq. (5.7) by setting the initial guess of $\mathbf{u}$ as the solution in the previous time step. In our implementation, we notice that Newton's method occasionally fails during the iteration. Therefore, we impose the Wolfe condition [226] to adjust the step length.

In our network training, we simplify the external force setting by only considering two types of $\mathbf{f}_{\texttt{ext}}$: directional force field and circular force field. The directional

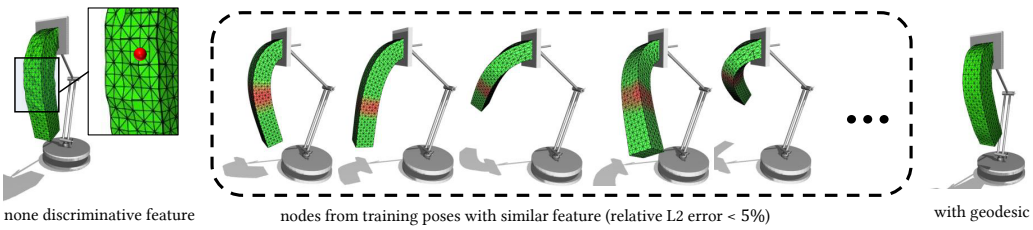


Figure 5.3: Only using kinematic feature as the input of the network yields noticeable jittery artifacts. A node, because of its kinematic feature is not discriminative, could be influenced by many irrelevant instances in the training date. Large discrepancies among these mismatched nodes induce high-frequency variations of the NNWarp. Incorporating geodesic feature effectively eliminates this artifact.

field uses a prescribed force direction, while the force direction in the circular field follows the tangent direction of a set of concentric circles. Such simplification frees us from generating an overwhelmingly large training set due to diverse external force conditions. Its limitation is also obvious: NNWarp loses some local deformation effects induced by high-frequency external forces.

5.3.3 Discriminative feature

With paired $\langle \widetilde{\mathbf{u}}, \mathbf{u} \rangle$, we can build a node-wise regression machine using a neural network that replaces Eq. (5.4) or Eq. (5.5). For the i-th node, in addition to its linear displacement $\widetilde{\mathbf{u}}_i$, the rotation information of its local displacement gradient $\mathbf{G}_i$ is directly pertinent to the warp transformation, and should be passed to the network as the input. To this end, we choose to use the skew symmetric part of $\mathbf{G}_i$ and represent it as a 3-vector as in [223]. However, only feeding these two pieces of information to the network is not enough, and the resulting deformation appears jittery and non-smooth as shown in Fig. 5.3. In this example, we use the Neo-Hookean elasticity, whose strain energy is:

$$\Psi_{\texttt{NH}} = \frac{k}{4(1+\nu)}[I_1 - \log(I_3) - 3] + \frac{k\nu}{8(1+\nu)(1-2\nu)}\log^2(I_3). \tag{5.8}$$

I_1 and I_2 are the invariants of the deformation gradient, defined based on $\mathbf{F}$'s singular values σ_1, σ_2 and σ_3 such that: $I_1 = \sigma_1^2 + \sigma_2^2 + \sigma_3^2$ and $I_3 = \sigma_1^2\sigma_2^2\sigma_3^2$.

This artifact was also noticed and discussed in previous data-driven simulation literature [212], which is because pure node-wise kinematic features do not contain sufficient contextual information, and thus are not discriminative to reach a conclusive per-node linear-nonlinear map. To further illustrate this artifact, we pick a jittery node (marked as a red sphere in the figure) and inversely query for nodes in our training set that have similar features ($< 5\%$ relative L2 error w.r.t. the feature vector from the picked node). We can see from the figure that there are a number of training poses having multiple nodes (on the red-shaded areas) with very similar feature vectors as the input. In other words, the final displacement of the picked node becomes a certain mixture of displacements of many distant and irrelevant nodes. Such ambiguity of pure kinematic feature is the primary reason behind this artifact.

One of our contribution is to design a compact contextual feature to resolve this mismath. While one could follow the method used in [212] to use the integral features of local dynamic parameters around a node, we found that our simple strategy yields satisfying result. We speculate that this is because DOFs in solid simulation are more tightly coupled than in fluid simulation [212]. An important advantage of such compactness is that the network training is also quite fast. Compared with state-of-the-art pre-computed deformable models i.e. [211], we can finish training in a few minutes, and the obtained network can be applied to a wide range of models.

Discriminative feature I: geodesic The geodesic of a node i, g_i is the normalized length of the shortest path from node i to its nearest anchor node within the deformable body. For a training model, we first uniformly scale it to fit a unit bounding sphere. Then, we compute the shortest path using the Dijkstra's algorithm for all the un-anchored nodes. Lastly, calculated path lengths are scaled by the maximum geodesic so that all the g values are within the normalized interval of $[0, 1]$. Our heuristic of

choosing the geodesic feature is based on the observation that if a node is closer to an anchor node, it trends to have less deformation than nodes that are away from it. By inducing the geodesic feature, a node far from anchor nodes does not miss-pair to a node close to anchor nodes only because the it undertakes a smaller external force. Consequently, the jittery artifact is effectively removed as shown in the rightmost snapshot in Fig. 5.3.

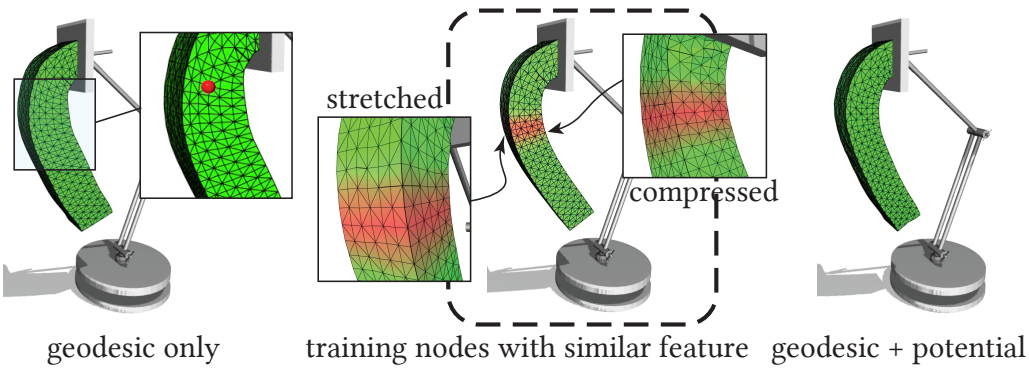
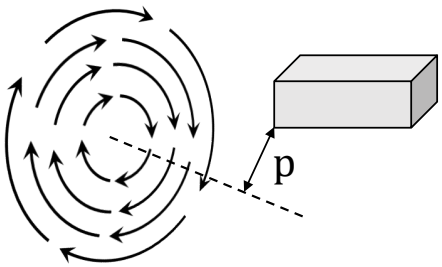


Figure 5.4: Volume expansion artifact remains even with the geodesic feature added. This is because the nodes with similar geodesic value may have different internal tractions. We use the potential feature to sort the training data to avoid this issue.

Discriminative feature II: potential Including the geodesic feature however, does not avoid the artifact of volume increase and shrinkage. As shown in Fig. 5.4, bending the beam also increases its volume noticeably, especially at curved areas. In order to dig out the missing contextual information behind this issue, we use the similar approach by picking a node within the problematic area and query the instances from our training set that have a similar feature of the selected one. We can see from the figure that, thanks to the incorporated geodesic feature, now this selected node only pairs with a training pose under a very similar deformation. However, it still matches multiple nodes on this pose. This



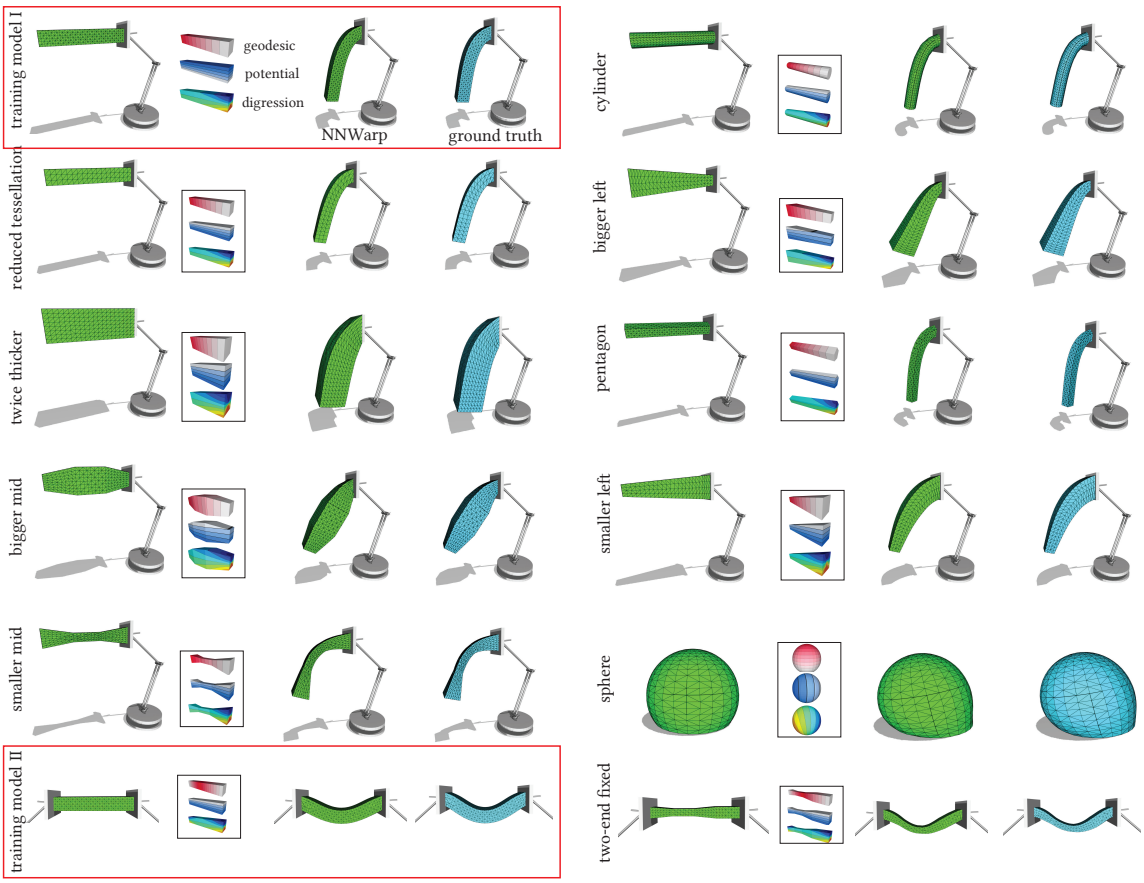


Figure 5.5: We test the generality of the designed features on a variety of shapes. The training data are generated using the standard rectangular beam (highlighted by a red box). The resulting DNN successfully handles many beam-like models but with distinctively different shapes. The distributions of three features are also plotted.

is because the beam is a symmetric shape, and a loop of nodes on its surface have similar geodesic values – among which, some are stretched and some are compressed. Without being able to distinguish these contexts, the volume of the warped model is likely to shrink or expand unnaturally.

We notice that whether nodes are being stretched or compressed typically depends on their relative positions in the applied force field. Therefore, we introduce another scaler feature named potential p to resolve this ambiguity. If a directional force field

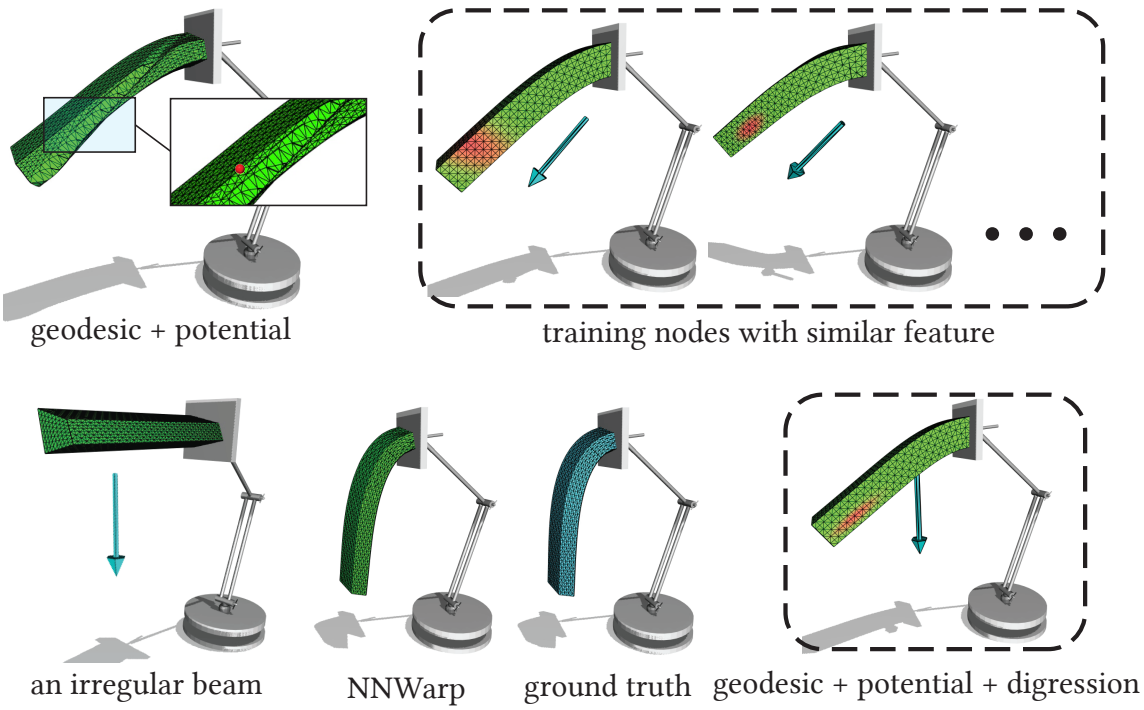


Figure 5.6: In order to make NNWarp re-usable for various deformable bodies, we use the digression as our third discriminative feature. With this feature included, the neural network is able to handle an irregular beam based on the training set generated using a standard rectangular beam model.

is applied, for each node on the mesh, we project its rest shape position onto the force direction and re-map the resulting projections to the interval of $[0, 1]$. On the other hand, if a circular force field is applied, the potential of a node is the distance between its rest shape position and the circular axis, as shown in the inset figure. This value is also scaled to $[0, 1]$. As we can see from Fig. 5.4, the deformation of the beam model is almost identical to the one obtained using the full simulation after we inject the potential feature into the network.

Discriminative feature III: digression So far, we generate a set of registered linear and nonlinear poses of the beam model. Node-wise linear to nonlinear deviation is learnt by a neural network, which is then used to warp a linear displacement of the

same model to obtain its nonlinear shape. While the results are visually plausible, real-world applications will require deformable animations of various 3D models. NNWarp becomes cumbersome and less practical if one needs to re-train a network for each different deformable body.

Unfortunately, if we alter the rest shape geometry of the beam model as shown in Fig. 5.6, unrealistic jittery deformations are observed again even after incorporating geodesic and potential features. By querying the training set, we see that because the updated shape is irregular and asymmetric, the most similar training poses become the ones under oblique force fields regardless an upright gravity force field is applied in the simulation. To further correct this mismatch, we use the digression feature d to describe the nodal position w.r.t. the direction of the external force. Specifically, the digression for node i is defined as:

$$d_i = \arccos\left(\frac{\bar{\mathbf{x}}_i - \bar{\mathbf{x}}_a}{|\bar{\mathbf{x}}_i - \bar{\mathbf{x}}_a|} \cdot \frac{\mathbf{f}_{\mathtt{ext}}}{|\mathbf{f}_{\mathtt{ext}}|}\right), \tag{5.9}$$

where $\bar{\mathbf{x}}_a$ is the rest shape position of the anchor node that is closest to node i. Indeed, digression sorts nodes based on their local orientational deviations from the external force direction. The digression feature ranges from 0 to π. If a circular force field is applied, the digression is simply set as -1. As shown in Fig. 5.6, with geodesic, potential and digression included, the training data generated using a rectangular beam model can also be used to warp the irregular beam, and NNWarp produces high-quality nonlinear deformation.

Discussion Our features allow the resulting network well handles models with various shapes, different tessellations and altered boundary conditions. More results can be found in Figs 5.5 and 5.7. From these examples, readers may probably notice that geodesic, potential and digression features actually provide a volumetric *parametrization* of deformable bodies so that models of different geometries and tessellations are somehow registered in a meaningful way, and node-wise neural network can then be applied. In fact, there are many elegant algorithms in graphics

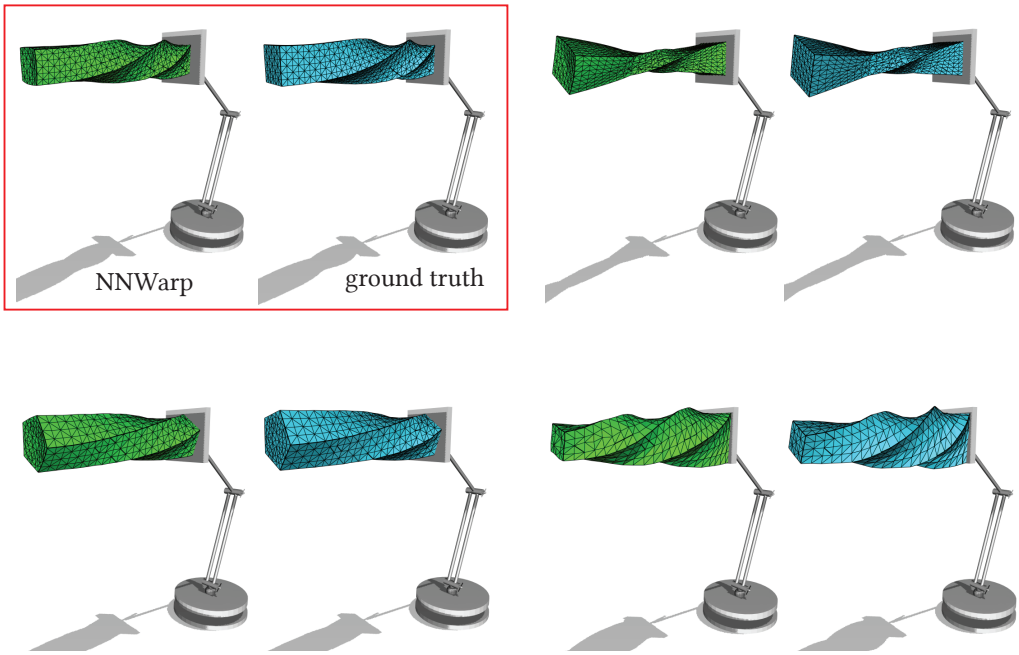


Figure 5.7: NNWarp works well under circular force field too. In this case, the digression feature is set as -1 for all the nodes on the mesh.

and computational geometry that generate the volumetric map between different shapes [227–229]. However in the context of NNWarp, this volumetric map depends on the configuration of external force and boundary conditions. While existing methods may also be modified to incorporate these additional conditions or constraints, we found that our simple strategy suffices in most cases. An exception is reported in Fig. 5.8, and we find that NNWarp using a convex training model often fails when the deformable body gets more concave. Next, we will show how to walk around this limitation without re-training the network for a different target shape.

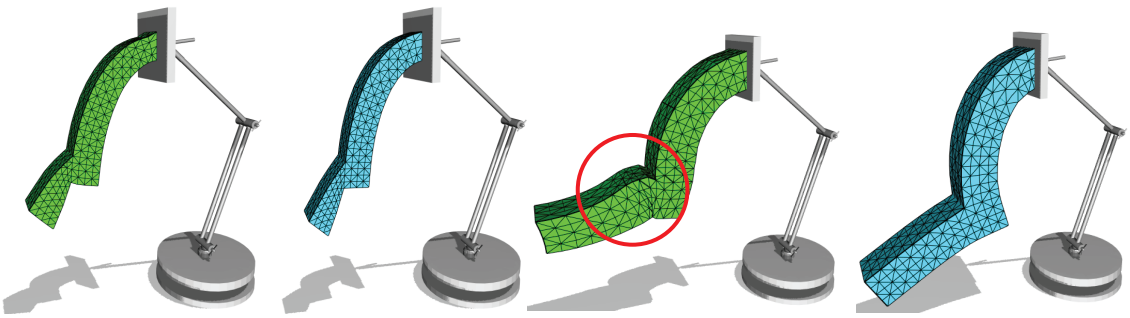


Figure 5.8: When the shape becomes more concave, the network trained using a rectangular beam produces artifacts.

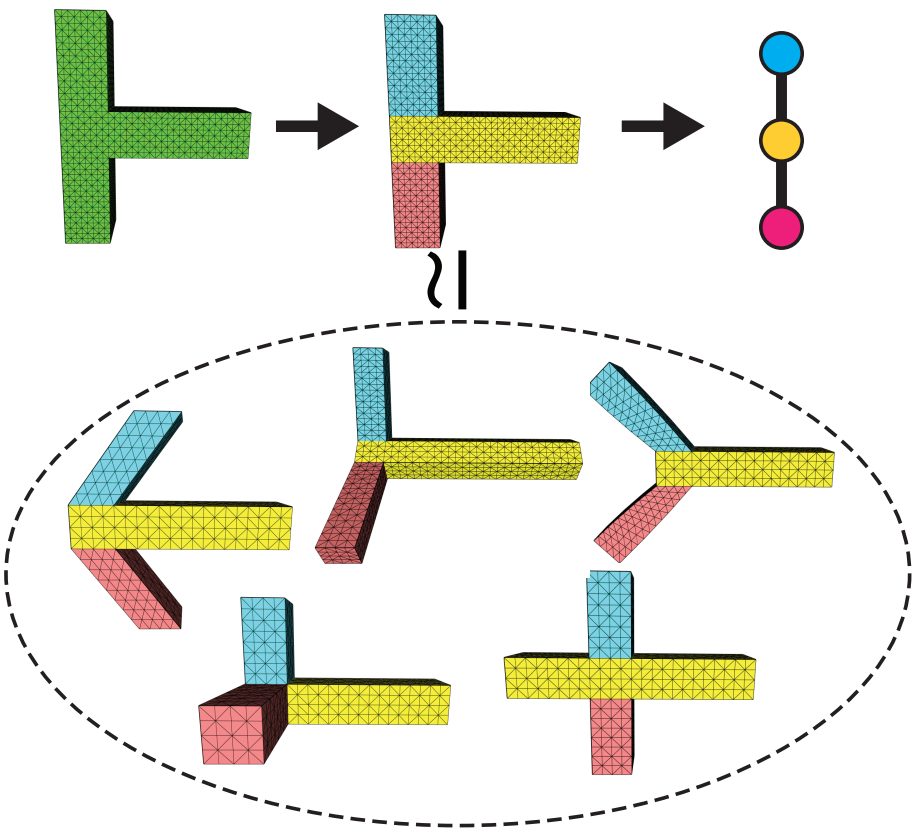


Figure 5.9: Building domain graph for the domain decomposed model is an easy and effective way to identify shapes with similar concavity.

5.4 Incorporate Complex Shapes

The exhaustiveness of 3D geometric diversity is endless. Obviously, training set generated with a single rectangular beam cannot cover all the different feature combinations. We find that the network trained using the beam model is able to deal with many convex 3D shapes (i.e. see Fig. 5.5). However, it often fails when the target deformable body becomes more concave (Fig. 5.8). A straightforward idea is to train a new network using a model with similar concavity of the target deformable body, but *how to describe the similarity of concavity among 3D shapes?*

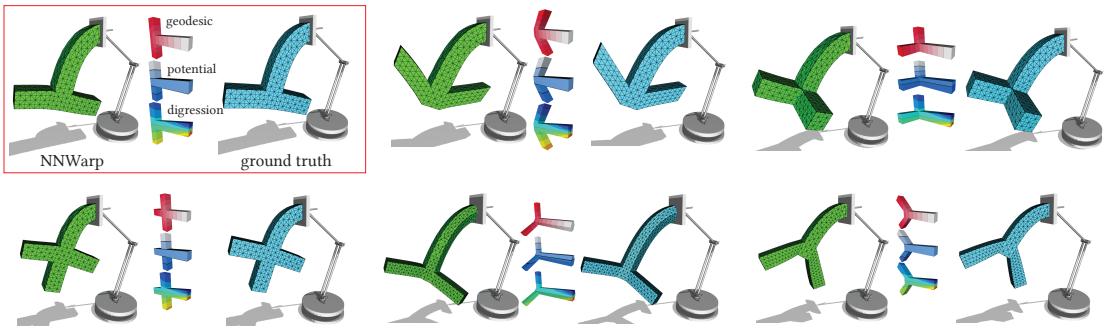


Figure 5.10: While a simple rectangular beam is not able to handle highly concave shapes, by referring to the domain graph we can train a network using a T-shape beam and the resulting network can be used to warp a wide range of concave beams whose domain graphs are isomorphic to the T-shape beam.

We borrow the idea from the graph theory, and subdivide a concave model into several convex components or domains. Afterwards, we create a *domain graph* $\mathcal{G}$ by using a graph vertex to represent each of the subdivided domains. An edge connects two vertices if and only if the corresponding two domains are face-connected on the original mesh. We find that if the domain graph $\mathcal{G}$ is isomorphic to the domain graph of the training model $\mathcal{G}_{\texttt{train}}$ or $\mathcal{G} \simeq \mathcal{G}_{\texttt{train}}$, NNWarp typically yields satisfying results. An example can be found in Fig. 5.9. The T-shape beam is decomposed into three domains, each of which is convex and rectangular. Its domain graph is isomorphic

to many similar concave shapes like the Y-shape beam, the arrow-shape beam, the crossing beam etc. If we use the T-shape beam as the training model, the resulting neural network is able to handle all of these variations as verified in Fig. 5.10.

Even utilizing the concept of graph isomorphism, one may still have to re-train the network (and re-generate training poses) for an arbitrary geometrically complex model, which is tedious and time consuming. A more general and powerful solution maximizing the re-usability of the network training is to use the substructuring method [180]. This method wisely leverages the hierarchical propagation of the deformation over a complicated structure and isolates the deformable simulation at each individual domain sequentially. While it loses some physics accuracy (mostly, the frequency of the trajectory due to the mass lumping, which can also be fixed as in [230]), the resulting deformation is natural and realistic. After the domain decomposition is complete so that each domain is a convex 3D shape, we can use one representative convex model to train the network. With the help of the proposed three discriminative features, the resulting network is able to correct local dynamics of all the domains.

In our NNWarp version of substructuring, the dynamics of the domain $\mathcal{D}_j$ is updated and corrected by NNWarp. After that, we calculate the best-fitting linear transformation $\mathbf{A}_{j,k}$ for the small patch of the mesh interfacing $\mathcal{D}_j$ and one of its children domain say $\mathcal{D}_k$ as $\mathbf{A}_{j,k} = (\mathbf{P}_{j,k}\mathbf{P}_{j,k}^\top)^{-1}\mathbf{P}_{j,k}^\top\mathbf{Q}_{j,k}$, where $\mathbf{P}_{j,k}$ and $\mathbf{Q}_{j,k}$ store the rest shape and deformed positions of all the nodes on the interface patch. We extract the relative rotation between $\mathcal{D}_j$ and $\mathcal{D}_k$ using the polar decomposition as $\mathbf{A}_{j,k} = \mathbf{R}_{j,k}\mathbf{S}_{j,k}$. Based on this, the angular velocity $\omega_{j,k}$ and the angular acceleration $\dot{\omega}_{j,k}$ can be calculated. Each domain is pinned to a local non-inertial reference frame. Therefore, in addition to the regular external forces, inertial forces originated from the accelerated linear and angular motion of the interface should also be computed as the *system force* and the *interface force*. We refer the reader to the related reference

from Barbič and Zhao [180] for the detailed formulation.

An example is given in Fig. 4.1, where NNWarp is still based on a rectangular beam model. However, because we decompose the maple tree into domains of branches and leaves, the neural network well handles nonlinear dynamics for each domain regardless how complex the original mesh is. Unlike other tree simulation results in the literature [131, 180, 230, 231], the example given in the figure is simulated in the *fullspace* without any reduction of the simulation DOFs. Therefore, local high-frequency details are well preserved. The simulation is close to interactive at 5 FPS – this is roughly 1,000 times faster than running fullspace nonlinear simulation using substructuring.

5.5 Network Structure and Training

The input of our neural network includes kinematic features of the linear displacement of the i-th node $\widetilde{\mathbf{u}}_i$ and its instantaneous angular velocity $\mathbf{w}_i = \nabla \times \left(\mathbf{P}_i\mathbf{P}_i^\top\right)^{-1}\mathbf{P}_i^\top\mathbf{U}_i$ as in Eq. (5.6). As to be discussed shortly, we further compress this pair of vectors into three scalars utilizing the rotation invariance property of the isotropic hyperelastic material. Doing so significantly relieves the effort of generating the training set. Besides, three discriminative features of geodesic, potential and digression are also included. We find that the Young's modulus k behaves more like a linear amplifier. Increasing Young's modulus yields a deformation similar to the one obtained by reducing the magnitude of the external force. Therefore, this material parameter is not explicitly fed to the network. However, Poisson's ratio ν controls the volume change, and its impact on the final deformation is much more nonlinear. This is also reflected in the strain energy formulation of Eqs. (5.1), (5.3) and (5.8). As a result, the Poisson's ratio is also an input feature. Other simulation configurations like the external force, tessellation, boundary conditions are not the input since we believe

this information is well encoded during the linear solve. The final input feature is a seven-dimension vector, and the network outputs a 3D vector of $\delta\mathbf{u}_i$ corresponding to a node-wise displacement fix so that $\widetilde{\mathbf{u}}_i + \delta\mathbf{u}_i$ is a well approximated nonlinear nodal displacement for a target material model. We use a different network for a different nonlinear material model instead of building a comprehensive one.

Training data alignment The complexity of a neural network highly depends on its input [178]. For instance, $\mathbf{w}_i$ can be extracted from the displacement gradient tensor $\mathbf{G}_i$. Nevertheless, if we simply put all the nine elements of $\mathbf{G}_i$ into the network, much higher training and testing errors are observed, which could only be improved by spanning the network depth and generating more training data.

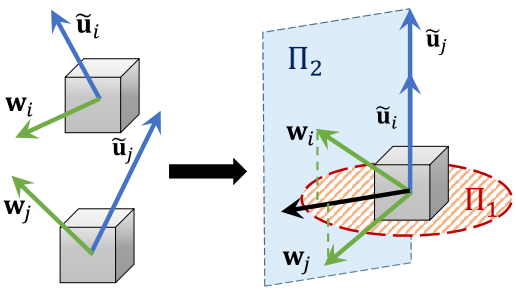


Figure 5.11: Rotation invariance allows us to further compress the input kinematic feature.

In order to make the network as compact as possible, we further align vectors $\widetilde{\mathbf{u}}$ and $\mathbf{w}$ based on the fact that *a nodal deformation measure can always be examined within a local coordinate frame which is invariant under rotations.*

This procedure is illustrated in Fig. 5.11. Suppose we have two nodes i and j. They are surrounded by two infinitesimal volumes, which are small enough to be considered as symmetric in all the orientations. We first rotate these two volumes so that the linear displacements $\widetilde{\mathbf{u}}_i$ and $\widetilde{\mathbf{u}}_j$ are both in the positive y direction. The follow-up rotation is around the y axis. One can pick an arbitrary direction (the black vector in the figure) within plane Π_1, which is perpendicular to the y axis. In our implementation, we set this direction as the negative x axis. After that, $\mathbf{w}_i$ and $\mathbf{w}_j$ are rotated so that they both reside in plane Π_2 i.e. the xy plane in our implementation. Because the second rotation is around the y axis, $\widetilde{\mathbf{u}}_i$ and $\widetilde{\mathbf{u}}_j$ remain aligned. By doing so, pairs of $\widetilde{\mathbf{u}}$ and $\mathbf{w}$ only differ at the magnitude or the norm of

the linear displacement, the magnitude of $\mathbf{w}$ and the angle between them. In other words, the real useful kinematic information hidden behind vectors $\widetilde{\mathbf{u}}$ and $\mathbf{w}$ are only three scalars. If one insists on putting the original $\widetilde{\mathbf{u}}$ and $\mathbf{w}$ into the network, the net must learn this double-rotation alignment out of the training data first and then fits the linear-nonlinear map. Unfortunately, the neural network is not good at processing such rotation invariant features. For instance, in existing works of using deep learning to perform 3D shape analysis [232, 233], in order to relieve the burden of the analysis of rotation invariant shape features, it is common to use *rotation augmentation* that duplicates a training data by rotating it from multiple angles. The final result is pooled out of all the rotated duplicates. Using data alignment, the size of the training set is reduced by over 10 times, and the training time is also significantly shortened. Fig. 5.12 plots the distribution of 1,000 randomly picked kinematic features.

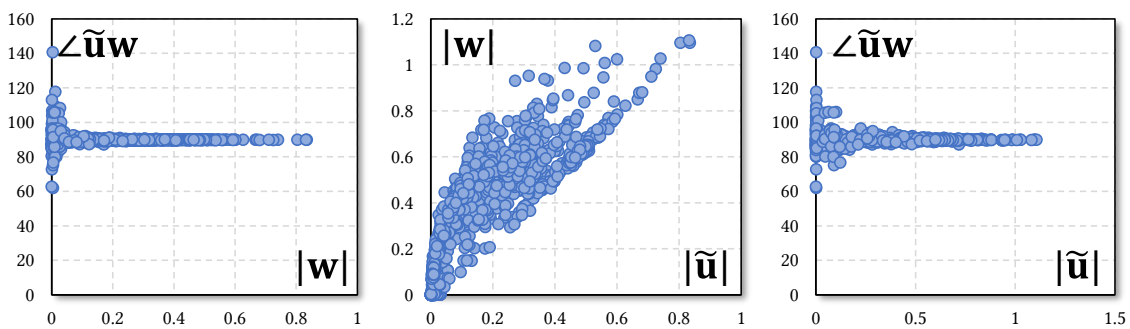


Figure 5.12: The distribution of three kinematic features of $1,000$ entries after the alignment. $\angle\widetilde{\mathbf{u}}\mathbf{w}$ denotes the angle between aligned vectors $\widetilde{\mathbf{u}}$ and $\mathbf{w}$.

Generating training set It is important to make sure that the training set covers the feature space of the simulation, because machine learning is known to have a relatively poor performance for extrapolation. For the direction of the external force field, we evenly scatter samples over a unit semi-hemisphere surface for the rectangular beam model. Specifically, we uniformly sample two variables α and β from the interval of $[0, \pi/2]$, which correspond to the latitudinal and longitudinal

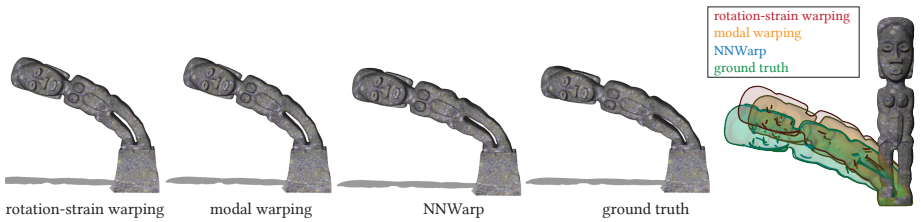


Figure 5.13: A side-by-side comparison shows a clear advantage of NNWarp over the existing warping techniques. Its data-driven nature makes the result almost identical to the ground truth while the simulation is as fast as the linear elasticity. The training still uses the rectangular beam model.

spans on the semi-hemisphere. The unit directional vector can be calculated as: $\mathbf{e} = [\sin\beta\cos\alpha, \cos\beta, \sin\beta\sin\alpha]^\top$. The magnitude of the external force determines the magnitude of the linear displacement, which could be an infinitely large vector in theory. However, as we have already normalized our training model into a unit sphere, an excessively large displacement vector is unlike to occur in a real simulation application. Therefore, we stop applying bigger external force if $|\mathbf{u}_i| \geq 2$ during the training data generation.

The discriminative features g, p and d are essentially for model registration. Therefore, how they are sampled depends on the training model's geometry and tessellation. In general, a moderately fine mesh should suffice for these features. However, if the training model is too coarse i.e. with few hundred elements, one may observe artifacts after warping.

Training specifications In our implementation, the beam model for the network training consists of $2,629$ elements, and we generate $20,829$ training poses including $16,730,893$ training nodal pairs and $167,309$ validation nodal pairs. The test data is $1/7$ of the training data with $2,064,812$ nodal pairs. Training and testing data are stored as binary files in `.npy` format with a total size of 1.42 GB. Unlike [213], we do not need to load these training poses during the simulation. Only the resulting

network parameters are needed. The network structure is rather simple – for co-rotation and Neo-Hookean materials there are only two hidden layers, and each of which has 16 neurons. For StVK material, the network has three hidden layers with 16 neurons at each layer.

The network is optimized using the Adam solver [234]. Druing the training, the neural network was built using `Google Tensowflow` [235] and optimized with `Google Cloud Platform` with 8 virtual CPUs. The training error over the first 20 epoches is plotted in Fig. 5.14. In practice however, we typically stop at 10 epoches. The total training time is less than 10 minutes on `Google Cloud`. It takes similar time if one performs the training on an `i7` PC with a high-end video card. The minibatch size is 1,024 and the learning rate is set as 0.001. Two hyper-parameters β_1 and β_2 control the exponential decay rates of moving averages, which are set as $\beta_1 = 0.9$ and $\beta_2 = 0.999$, and $\epsilon = 1e-8$. We use the `tanh` defined as $e^x - e^{-x}/e^x + e^{-x}$ as the nonlinear activation function. We found that `tanh` outperforms the widely-used `ReLU` in our experiment. We guess this is because the input-output relation of the net is clearly a smooth nonlinear function in our case, and `ReLU` may excel when the input-output relation contains discontinuity and/or singularity as in many computer vision problems like image recognition. Also, because all the training data generated using FEM simulation are clear and noise-free, and the data coverage is carefully controlled to avoid over- and under-sampling of the input feature space, we do not apply dropout during our training.

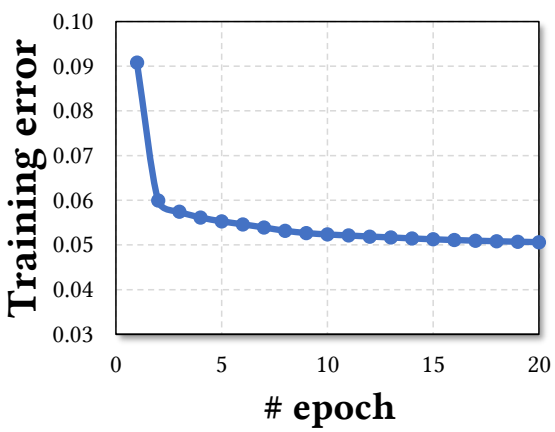


Figure 5.14: Training error vs epoch of the Adam solver.

5.6 Other Experimental Results

The simulator module was implemented using `MS Visual C++ 2013` on an `Alienware` desktop PC with an `Intel i7 5960` CPU (at 3.0 GHz) and 32 GB memory. It also equips with an `nVidia GTX 970` GPU. We used `Eigen C++` template for most numerical computations. Some of our implementations also used the published `Vega library` [236]. NNWarp utilizes a standard linear simulation running at background. The external force applied at each node needs to be rotated back to its rest-shape orientation as did in stiffness warping [179]. This local rotation is computed by converting $\mathbf{w}_i$ into a rotation matrix, which only induces minor extra computing efforts since $\mathbf{w}_i$ itself is also the network's input. The timing statistics of examples shown in the chapter are reported in Table 5.1. The source code (for both neural network and simulator) and executables can be found in the supplementary file. The training data (for the Neo-Hookean material) is also available from an anonymous `dropbox` link provided in the supplementary file.

Comparison with existing geometry warping methods First of all, we compare our method with existing geometry warping methods including modal warping (MW) [12] and rotation-strain warping (RSW) [223]. We stick with using the rectangular beam as our training model, and simulate the bending deformation of a Neo-Hookean toy statue using MW, RSW, NNWarp and fullspace FEM simulation. While all the methods demonstrate plausible nonlinear bending effects, when putting together, one can see that MW and RSW are actually quite different from the ground truth result. On the other hand, NNWarp yields a result that is hardly distinguishable from the ground truth. Because MW and RSM use a fixed linear-nonlinear map template (i.e. Eqs (5.4) and (5.5)), they show no difference with different hyperelastic materials. However, NNWarp is able to produce high-quality results for various material models due to its data-driven nature.

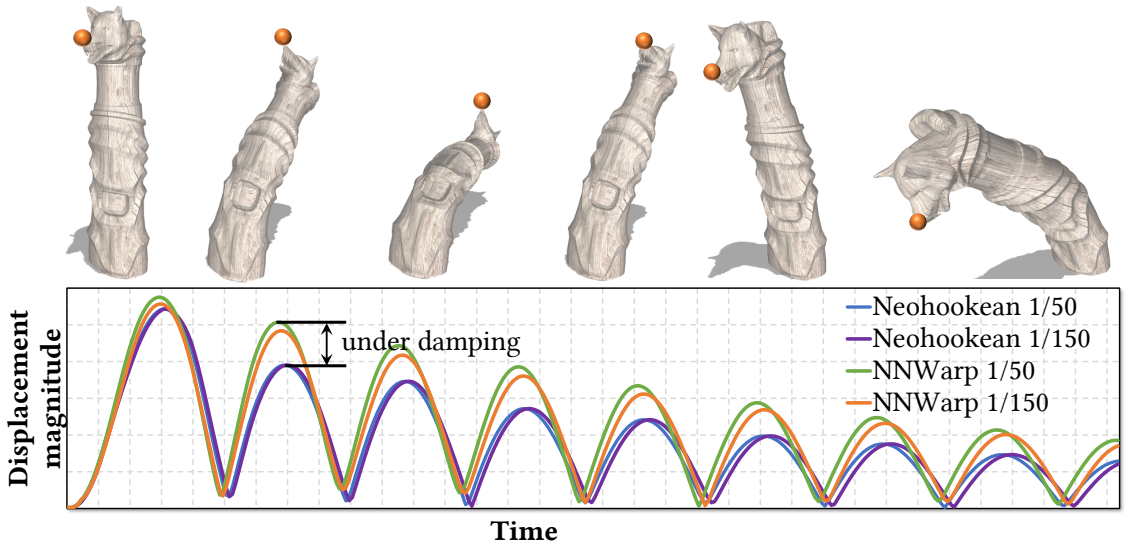


Figure 5.15: The deformable motion trajectory (at the nose tip of the wolf head) generated using NNWarp well matches the ground truth under different time step sizes. The vibration frequency resembles the ground truth as well. We use the Newmark integrator in this example.

Trajectory comparison Another aspect we would like to investigate is the motion trajectory, and see how far NNWarp deviates from the ground truth along the simulation time. To this end, we apply NNWarp to a wolf totem model and plot the displacement of the node at the nose tip of the wolf head w.r.t. time. Our reference is the fullspace FEM simulation using the Newmark integration and the material model is Neo-Hookean. We compare the resulting trajectories with time step size set as 1/50 *sec* and 1/150 *sec* respectively. Surprisingly, the trajectory generated using NNWarp is very close to the ground truth in both time step size settings. The vibration frequency is almost identical to the ground truth. This is probably because NNWarp is essentially a fullspace simulator, where the mass inertial is lossless unlike in reduced simulations. On the other hand, we do observe an artificial under-damping issue as we can see from the plotted trajectories. It seems that the linear Rayleigh damping dissipates less energy ($\sim$ 10% in this example) than the nonlinear one. However, this issue should be fixed by dynamically adjusting the Rayleigh damping

Model	**# Tetrahedra**	**# Domains**	**Factorization**	**Solve**	**NNWarp (CPU)**	**NNWarp (Shader)**	**FPS** (CPU/GPU)
Beam	$2,629$	1	$6.9ms$	$< 1ms$	$1.5ms$	$< 1ms$	333/666
Dragon	$51,850$	14	$307ms$	$15ms$	$15ms$	$< 1ms$	16/22
Armadillo	$52,278$	15	$403ms$	$15ms$	$18ms$	$< 1ms$	15/21
Dinosaur	$54,796$	14	$334ms$	$18ms$	$15ms$	$< 1ms$	18/24
Bunny	$24,956$	4	$273ms$	$10ms$	$7ms$	$< 1ms$	33/43
Maple bonsai	$255,552$	$1,771$	$1,556ms$	$83ms$	$109ms$	$< 1ms$	5/10

Table 5.1: Time performance of the examples reported in the chapter. **Factorization** is the time needed to pre-factorize the system matrix of the linear elasticity. We use `SimplicialLLT` solver shipped with `Eigen`. During the simulation, we only need to solve the system once. **FPS** reports both CPU and GPU performance.

coefficients as did in [231].

More examples & GPU implementation With the help of substructuring method [180], training a single model can be utilized to handle geometrically complex deformable bodies. In addition to the example shown in Fig. 4.1, Fig. 5.16 shows more results using NNWarp. The Armadillo, dinosaur and dragon models are of StVK, co-rotation and Neo-Hookean materials respectively. The networks used are still based on a single rectangular beam model.

NNWarp is node-wise. Its local correction of each nodal displacement is independent and can be parallelized trivially on GPU. We also implemented a shader version of NNWarp. NNWarp relies an underlying linear solver during the simulation run time. It is known that the asymptotic time complexity of solving a pre-factorized matrix is $\mathbf{O}(N^2)$ while NNWarp correction is just $\mathbf{O}(N)$. In other words, the benefit of the GPU implementation is limited in general. It is easy to see that NNWarp also synergizes well with model reduction. One can use the linear modal analysis to construct a r-dimensional linear subspace. Because the model reduction is applied to the linear solver, other more expensive pre-computations like Cubature training [211] are not needed. The network training for NNWarp is much faster than the Cubature training. More importantly, Cubature training is model-dependant, while NNWarp training is

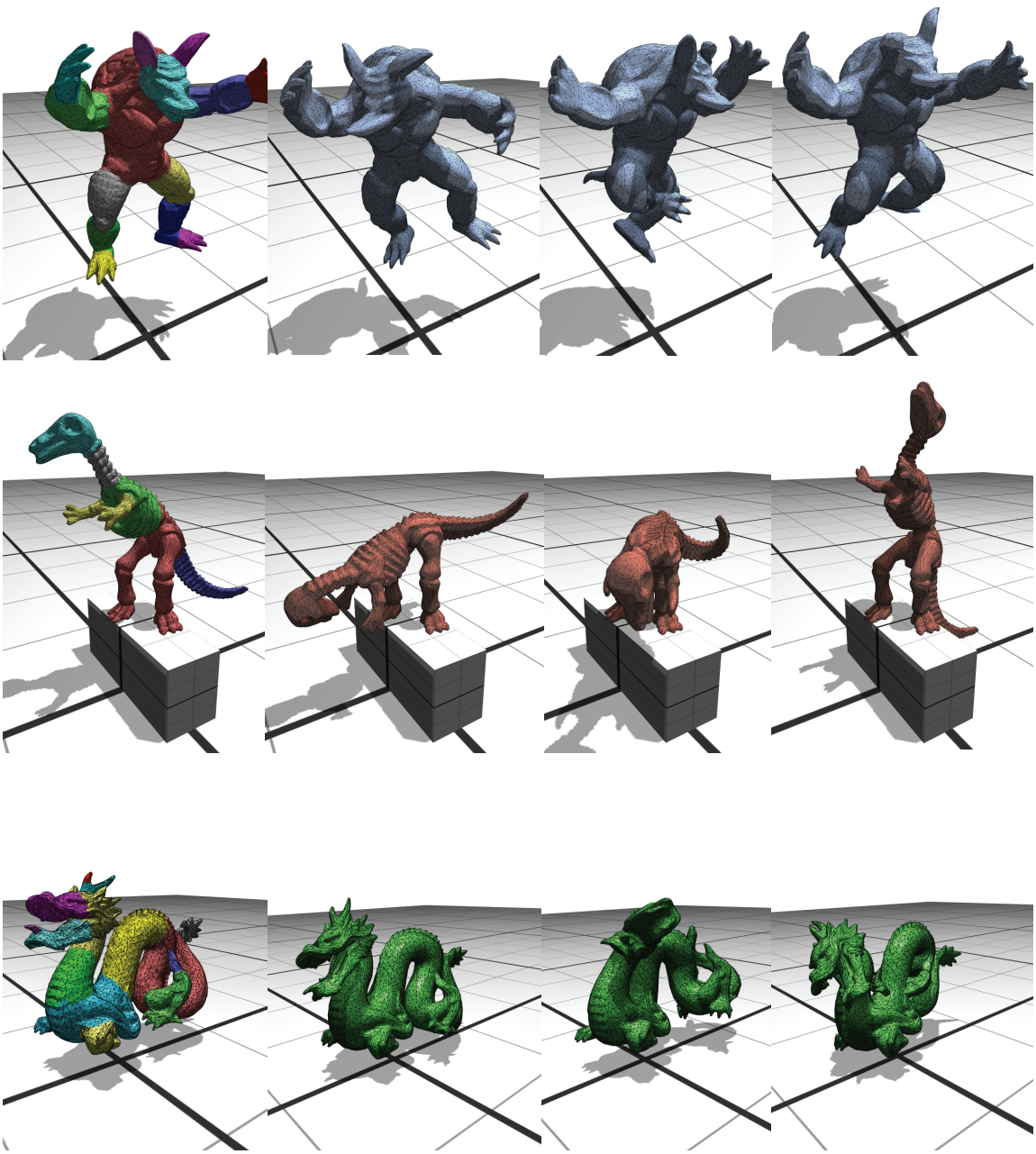


Figure 5.16: Substructuring allows us to re-use the training data of a regular shape to handle complex deformable bodies. The Armadillo, dinosaur and dragon models use the StVK, co-rotation and Neo-Hookean materials respectively. They use the networks trained with the rectangular beam.

more general. With the linear modal reduction, the cost for the diagonalized linear solver is reduced to $\mathbf{O}(r)$, and one should expect more noticeable accelerations by using the GPU. We do not report extra results using model reduced NNWarp since this is a natural extension and not the primary contribution of this work, nevertheless the simulation performance of the maple bonsai model shown in Fig. 4.1 can easily exceed 100 FPS with modal reduction.

When using the GPU-based NNWarp, some extra cares are needed for the deformation substructuring. This is because all the information regarding the final nonlinear displacement is in the GPU memory, which prevents us to evaluate the system and interface forces for per-domain dynamics at the CPU side. For the interface force, since it is assumed that the number of nodes on the domain's interface is small, we compute a CPU-based NNWarp for all the interface nodes to obtain their corrected displacement. For the system force, we treat an entire domain as a single mass point and estimate a domain-level rotation to warp it to the local non-inertial frame. Doing so compromises the physics accuracy, but avoids expensive data exchange from GPU and CPU.

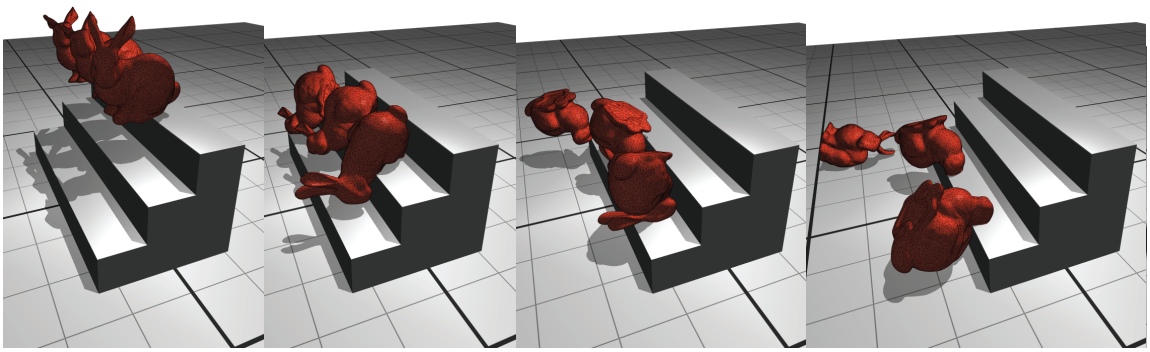


Figure 5.17: We can simulate free-floating deformable bodies by creating an artificial boundary condition to constrain the element near the mass center.

Free-floating deformable bodies Free-floating objects do not have boundary

conditions, and our discriminative features are ill-defined under this situation. As an easy walk-round, we pick a tetrahedron that is closest to the mass center of the deformable body, constrain all of its four nodes and training the network based on it. During the simulation, we couple a rigid body simulator with the deformable simulation as in [237], where NNWarp is applied within the reference frame attached to the rigid body simulator (Fig. 5.17).

Chapter 6

Application: Real-time Speech-driven Tongue System

6.1 Introduction

In this chapter, we provide a novel application of real-time reduced nonlinear solid simulation. The human tongue is a muscular organ that plays an essential role during speech production. A high-quality visual representation of the human tongue for specific speech sounds is of importance in the domain of speech research and has numerous potential applications. For example, in the rehabilitation of speech disorders [238], a realistic visualization of 3D tongue motion could provide a visible paradigm that helps an individual achieve the correct articulation of the tongue during the production of various speech sounds.

Unfortunately, the detailed mechanism that drives the deformable motion of the human tongue remains largely unknown to the research community – there exist many challenges, both practical and theoretical, that are still underexplored. Firstly, the tongue is an interior organ inside of the oral cavity. As such, ordinary optical

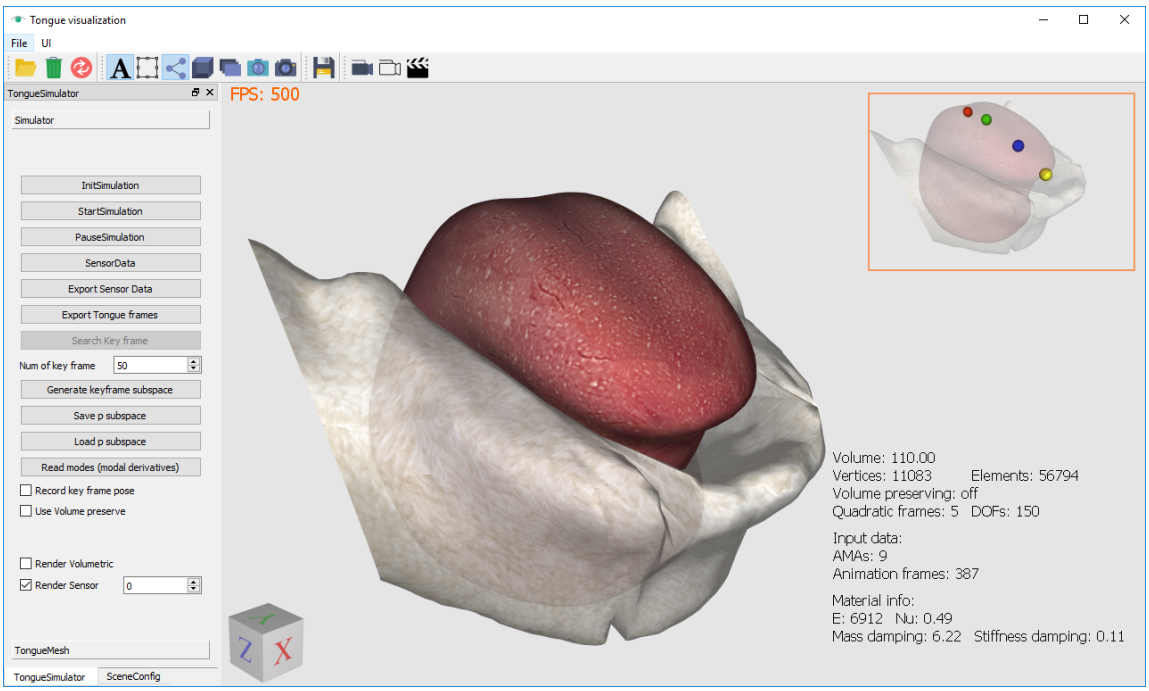


Figure 6.1: A snapshot of the interface of the proposed system.

sensors like video cameras are not suited to retrieve the motion data. Secondly, the tongue's movement during the production of speech is swift. For instance, a complete production of a single vowel-consonant-vowel (VCV) syllable takes only tenths of a second. Therefore, most 3D imaging modalities like computed tomography (CT) or magnetic resonance imaging (MRI) are not able to follow such quick movements. The ultrasound imaging (US), while widely used in many tongue related research, produces only 2D information and contains noise which requires post-processing. More importantly, the contour information at the tongue tip is frequently missed. Restoring or constructing the per-frame correspondence for a given US sequence is challenging and relies heavily on manual labelling, which is subjective and tedious. Lastly, the anatomical structure of the tongue is quite complex [239, 240], requiring several intrinsic and extrinsic muscles to be tightly coordinated during the speech sound production. An accurate mathematical description for the speech motor control is still beyond the knowledge for speech scientists [241, 242]. The inverse dynamics

method turns out to be a promising solution to this problem [243–247]. This method does not require an active muscle-activation-driven motor to control the tongue's motion. Instead, it unitizes the pre-defined bio-mechanical parameters of the vocal tract to reconstruct the tongue motion by enforcing certain constraints during the simulation.

The proposed framework further advances the existing inverse dynamics models and advances the frontier of creating a realistic virtual reality representation of the invisible vocal tract. While our framework also employs the idea of inverse dynamics, many novel techniques have been developed which complement and are orthogonal to the state-of-the-arts. First of all, the tedious sensor setup is hidden in our system to the end user. To this end, we use deep learning to train a mapping mechanism that directly converts input acoustic signals to feature vectors (position information) of articulators. The training uses a database consisting of complete and meaningful sentences (in Chinese) instead of simple CV syllables (i.e. as in [247]). Secondly, our framework equips a dedicated nonlinear finite element method (FEM) simulator using a technique called *spatial reduction* and domain decomposition. Nonlinear deformations of the tongue can be simulated accurately in *real time*. This method allocates low-dimensional simulation degrees of freedom (DOFs) more effectively than standard modal reduction techniques [11, 76]. The nonlinear deformation pattern is well captured by quadratic DOFs associated with each domain and is smoothly blended across the entire tongue model. The nonlinear volume preserving constraint is fully addressed in our framework. At each simulation time frame, we compute a displacement and velocity correction that effectively suppresses the volume change. A data-driven method is used to create a pressure subspace to facilitate an efficient volume conservation at the simulation runtime. The simulated tongue shapes are compared to real-world MRI-CT data. The results show that our framework delivers a high-quality animated tongue dynamics which could be of great potential for a wide range of medical scenarios and clinic applications. In summary, some noteworthy

technical features of our framework are:

- **Plug and play:** We leverage the deep learning method to train a speech inversion mechanism from acoustic signals to articulators' positions (§ 6.5). Hence, from an end user's point of view, tedious experiment setup is skipped, and the speech production can be performed in a comfortable and sensor-free environment, which drives the visual animated dynamic model in real time.
- **Nonlinearity in real time:** Our inverse dynamic simulator uses a series of novel numerical techniques that accurately capture local nonlinear deformations of the tongue while retaining the entire simulation algorithm within a low-dimensional configuration (§ 6.6 & § 6.7). Volume preservation is achieved by using displacement/velocity correction within the pressure subspace (§ 6.8).
- **High-quality:** Our framework is empowered by real-world and subject-specific data from various imaging modalities (§ 6.4). The model's quality at each step of the framework is systematically evaluated in an objective way (§ 6.9).

6.2 Related Work

The human tongue is a critical articulator for speech sound production. Investigating its behavior and contribution to speech production has been of interest to researchers in linguistics, phonetics and physiology. Due to the interdisciplinary nature of this work, we only briefly cover a few of the most relevant existing studies in acoustic signal processing, speech inversion and FEM tongue modeling in this section.

Acquisition of the tongue's geometry The human tongue is an interior organ and its motion is inaccessible to regular optical sensors like video cameras. The rapidly developing MRI systems have been used as an important data source [248, 249] for gathering 3D tongue shapes. To capture the tongue motion, three sagittal directions

of MRI images [250] were used to record the 2D contours of tongue in three sagittal planes. However, the MRI acquisition frequency is too low for capturing the rapid tongue motion during real language production. Recent advances of high-speed MRI [251] have shown significant potentials of real-time shape acquisition [252–254]. However, they are still not yet able to capture intact 3D geometric information of the tongue. X-ray CT imaging systems have higher temporal resolutions [255]. However, they expose the speaker to radiation. Thus, they are not applicable for massive data collection. Ultrasound or US systems [256, 257] have also been widely used for modeling tongue movements at a very high frequency (100 Hz for instance). However, they often miss the tracking of the tongue tip [258–260] because of the surrounding air gaps, and the resulting images are always noisy. Hence, restoring the frame-to-frame correspondence over the tongue 2D contour for an US sequence is a challenging problem which requires significant overhead [261–263].

Speech inversion Our framework is also related to speech inversion, a technique that estimates vocal tract shapes or articulators' positions based on input speech signals. Speech inversion has been performed by the codebook searching method by synthesizing sounds from the entire space of control parameters of an articulatory model [264, 265]. The problem with this approach is that the synthesis includes articulations that never occur in real human speech production. Clearly, the relationship between the acoustic and articulatory features is highly nonlinear and may not be bijective. Furthermore, the articulator's movements are not solely determined by the phoneme being pronounced, but also by the succeeding or preceding phonemes (the so called co-articulation phenomenon). In past decades with the increasing popularity of machine learning techniques, a number of methods have been proposed to tackle this problem using statistical learning such as the Hidden Markov model (HMM) [266, 267], the Gaussian mixture model (GMM) [268], the artificial neural network (ANN) [269], and the deep neural network (DNN) [270]. Wu and colleagues [271]

tested the performance of the aforementioned techniques on the acoustic-articulatory English speech corpus MNGU0 (http://www.mngu0.org/). The results indicated that DNN-based acoustic-articulatory mapping tended to yield the best performance, and our framework also uses the DNN model to achieve a high-quality speech inversion.

FEM-based tongue dynamics Biomechanical models of the tongue using FEM methods are widely used [272–274]. An active biomechanical model takes the muscle activations as the input to simulate the speech motors. Interesting results have been reported using active models. For instance, Stavness and colleagues developed an algorithm that is able to automatically estimate the internal activation of a muscle group [275]. The limitations of this approach is that the state-of-the-art active model is only able to simulate general tongue movements like upward or lateral bending. Subtle and localized deformations on the tongue are still difficult to be directly generated. Conversely, the inverse dynamics technique or the passive model that builds the unknown motion based on pre-known constraints [243, 245, 247] have some notable advantages over the active methods. They allow us to restore the 3D motion of the tongue with partial information that is more accessible than full-scale activation control.

Real-time deformable model Simulating the human tongue is a problem well suited for FEM deformable models. Since the FEM simulation of nonlinear deformation is known to be time-consuming, a technique referred to as *model reduction* is widely used in the computer graphics and animation community [70], which is able to improve performance by orders-of-magnitude. The idea is to build a displacement subspace consisting of representative deformed shapes and restrict the nonlinear integration within the constructed subspace. Standard model reduction uses the modal analysis that decomposes the dynamics into a set of linear vibrations [276] which is only valid for small-scale linear elasticity. Yang and colleagues [247] used an

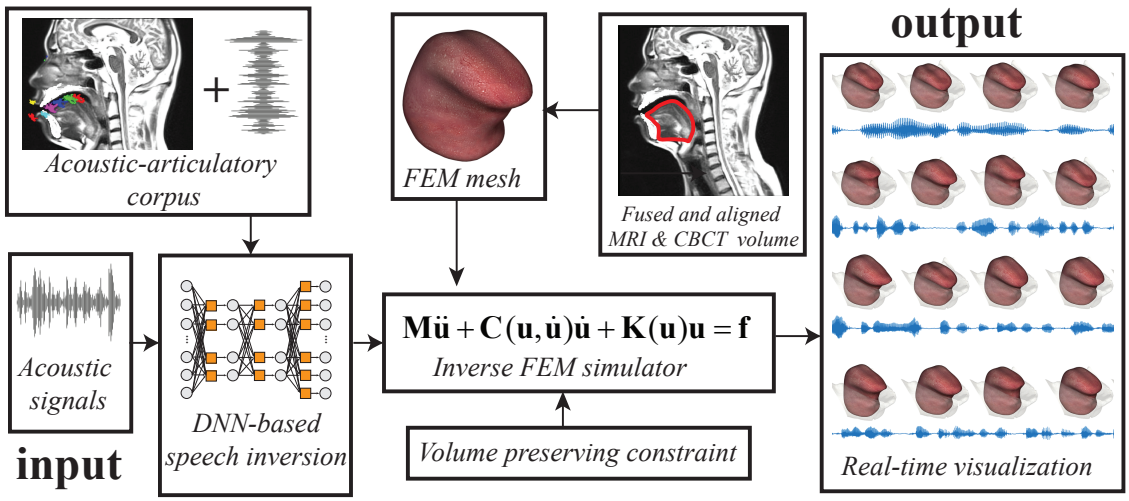


Figure 6.2: An overview of our acoustic-VR system.

extended modal analysis method called modal warping [76] to simulate the tongue's dynamics. Because this method is still based on linear elasticity, it is not able to produce plausible tongue dynamics for full words and sentences. Nonlinear elasticity can also be dealt with using modal analysis [11] however, subspace bases are global eigenvectors and less expressive for localized deformation.

6.3 System Overview

As sketched in Fig. 6.2, our framework takes real human speech signals (in Chinese) as input, and outputs realistic tongue animation sequences in real time corresponding to the speech being produced. The input speech signals are first mapped to articulator's positions through a DNN-based speech inversion. The DNN is trained using an acoustic-articulatory corpus consisting of $1,108$ complete sentences (§ 6.5). The output of the speech inversion is the estimated position information corresponding to four electromagnetic articulography (EMA) sensors at the tongue's tip, blade, dorsum and rear on the mid-sagittal plane. This information serves a set

of constraint equations for the FEM tongue simulator. Our simulator is geometrically nonlinear. The finite element mesh of the tongue is obtained by fusing the MRI and CBCT volumes of the same subject, whose contour is manually outlined by domain experts and triangularized afterwards (§ 6.4). We developed a novel reduced deformable simulator using blended quadratic domains. While this simulator is low-dimensional and model reduced, the nonlinear DOFs are assigned according to the location of EMA sensors so that local deformation can be well captured (§ 6.6). The global and general tongue's motion is calculated by smoothly blending the local domain-wise deformations in a material-aware manner (§ 6.7). Our simulator also addresses the nonlinear volume preserving constraint efficiently in a pressure subspace (§ 6.8). Collision detection and resolving is also handled in our system using the penalty method. As most FEM-related computation is done within a low-dimensional subspace, our system is real-time and able to produce plausible animations directly from human speech. The following sections describe each major technical component of our system in detail.

6.4 Data Acquisition

Our system is built upon real-world, subject-specific data including the tongue's geometry, feature positions from EMA coils, and the associated acoustic signals. This section elaborates on the data acquisition procedure.

Construction of the 3D tongue model Our 3D tongue model is built using both MRI (for soft tissues like the tongue, soft palate, and the pharyngeal wall) and cone beam CT (CBCT) (for bony structures) of an individual subject. The MRI data is recorded using the `SIEMENS MAGNETOM Trio`, a Tim system with 3 tesla magnetic field strength, 64 ms echo time, 340 ms repetition time, 31 sagittal slice

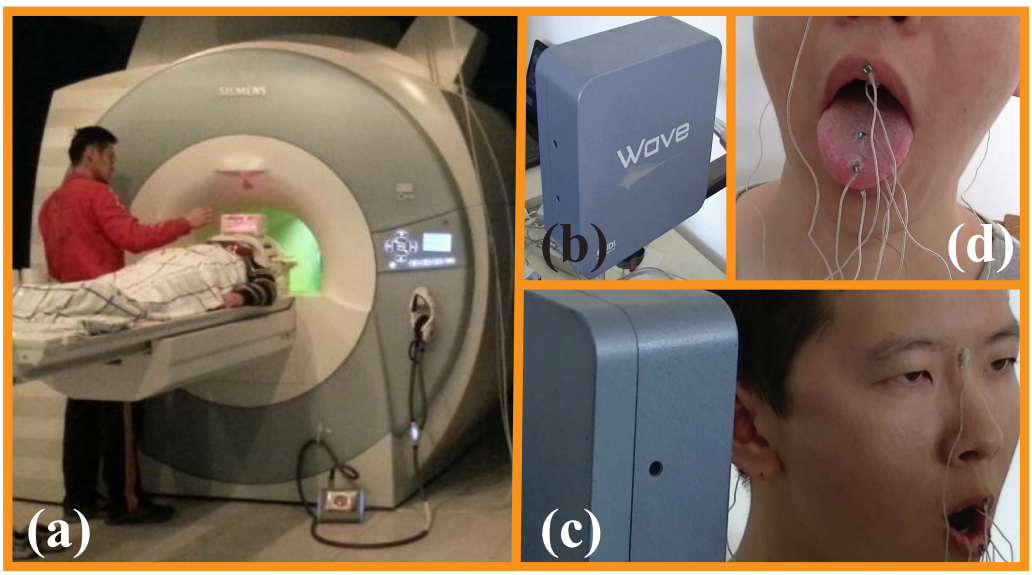


Figure 6.3: Experiment setting for data acquisition. (a) the SIEMENS MRI system. (b) NDI Wave EMA system. (c) Gathering the articulary movement data using EMA. (d) Placing EMA coils on the tongue.

planes, 3 *mm* slice thickness, 3.6 *mm* slice interval, 256 by 256 *mm* field of view, and 192 by 192 pixel resulting image size (Fig. 6.3 (a)). The rightmost and leftmost planes are located at 54 *mm* from the mid-sagittal plane. The "rest shape"of the tongue is defined as the averaged tongue shape for 36 Chinese vowels (9 vowels with 4 different tones) and 73 consonants in symmetric VCV syllables including fricative, stop, affricate, nasal, as well as lateral[1]. Detailed phonetic information is reported in Fig. 6.4.

During MRI data acquisition, the subject took the supine position and was asked to perform the required VCV syllables after a short period of warm-up practice. Each VCV sequence was produced with a consonant, surrounded by vowels, e.g. [a]–[t]–[a]. All articulations were artificially sustained during the ten-second acquisition time.

[1]We note that there does not exist a well defined rest configuration of the tongue, and the most comfortable position of the tongue varies significantly by individuals. Therefore, the median shape of the tongue while performing a series of standard pronunciations is a more meaningful starting point.

Vowel	Fricative	Stop	Affricate	Nasal	Lateral
[a], [i], [ɳ], [ɭ], [u], [ɛ], [ɤ], [o], [y]	[s] + [a], [i], [u]; [ʂ] + [a], [i], [u]; [ɕ] + [i], [y]; [f] + [a], [ɛ], [u], [o]; [x] + [a], [ɛ], [u];	[t] + [a], [i], [u], [ɛ]; [k] + [a], [i], [u], [ɛ]; [p] + [a], [i], [u], [o]; [p^h] + [a], [i], [u], [o]; [t^h] + [a], [i], [u], [ɛ]; [k^h] + [a], [i], [u], [ɛ];	[ʦ] + [a], [i], [u], [ɛ]; [ʦʂ] + [a], [i], [u], [ɛ]; [ʦɕ] + [i], [y]; [$ʦ^h$] + [a], [i], [u], [ɛ]; [$ʦʂ^h$] + [a], [i], [u], [ɛ]; [$ʦɕ^h$] + [i], [y];	[m] + [a], [i], [u], [o], [ɛ]; [n] + [a], [i], [u];	[l] + [a], [i], [u], [y], [ɤ]; **Approximant** [r] + [i];

Figure 6.4: VCV syllables used for MRI-based tongue shape retrieval.

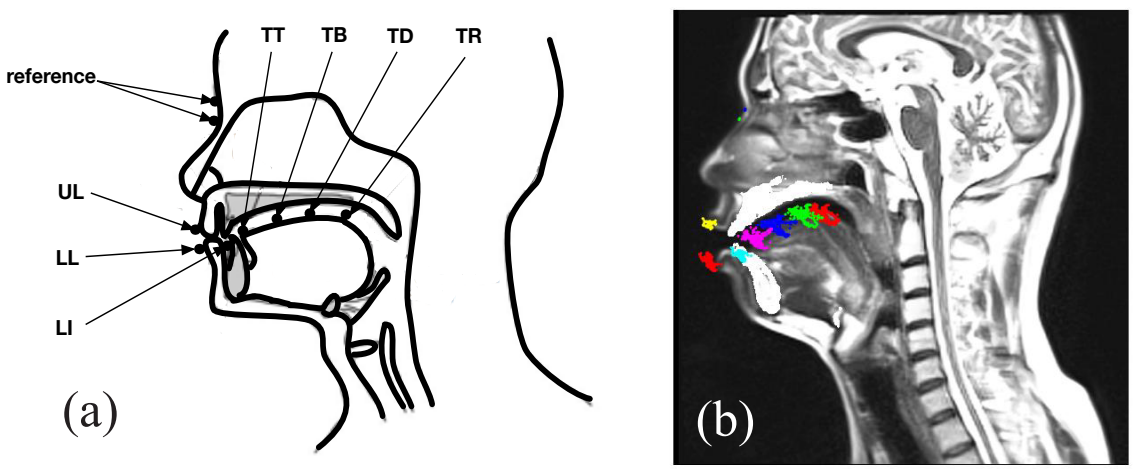


Figure 6.5: (a) EMA sensors' placement and (b) the aligned MRI-CBCT-EMA volume.

For consonants, the subject made the initial VC transition before the acquisition, then held the articulation while breathing out very slowly (for fricatives) or holding his breath (for stops) and finally made the rest CV transition after the MRI scan. Other bony structures attached to the tongue such as teeth and jaw are acquired by a `LargeV HiRes 3D` dental CBCT. This device is primarily used for dental surgery and delivers much less radiation to the subject than regular CT systems. Afterwards, a rigid body registration is applied to align the MRI and CBCT volumes (Fig. 6.5 (b)). Finally, the volumetric tetrahedral mesh is built using `Tetgen` [277] based on the extracted surface information.

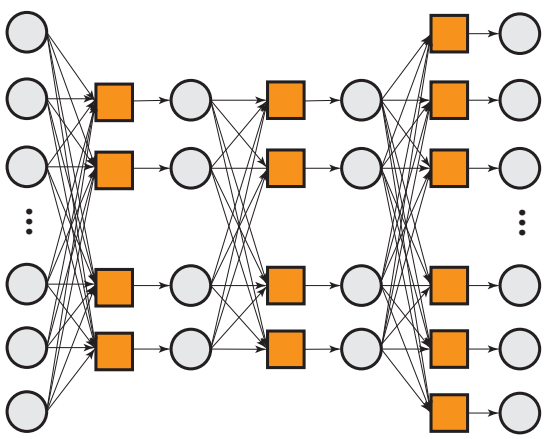
Acoustic-articulatory corpus The `NDI Wave` system (Fig. 6.3 (b)) is employed to record acoustic and articulator position recordings simultaneously. The articulators use electromagnetic transducer coils glued to the vocal-tract articulators to record precise measurements of their positions. There are $1,108$ phonetically balanced Chinese sentences in total selected to serve as the recording prompts. In the EMA experiment, coils or sensors are attached to the Tongue Rear (TR), Tongue Dorsum (TD), Tongue Blade (TB), Tongue Tip (TT), Lower Incisor (LI), Lower Lip (LL) and Upper Lip (UL) in the mid-sagittal plane. Another two coils attached to the ridge of nose serve as a reference (as shown in Fig. 6.5 (a)). As a result, we can easily extract the global rigid body motion associated with head's movement as in [247]. The same subject participates in the EMA experiment. The acoustic signals and articulatory data are recorded simultaneously. The sampling frequencies are $16,000\ Hz$ for acoustic signals and $100\ Hz$ for the articulatory signal, respectively. A third-order *Savitzky-Golay* filter [278] with the frame size of 21 is applied to smooth the trajectory of coils attached to articulators to suppress their jittery motions.

6.5 DNN-based Speech Inversion

Speech inversion is the first step in our system. It refers to the procedure that estimates the vocal tract shape or articulators' positions based on speech signals. We develop a DNN-based mapping mechanism bridging the acoustic speech and the articulatory movement, taking the features representing the acoustic speech as the initial input and outputting the articulatory features at EMA sensors.

Traditional DNN is constructed by stacking a series of trained *Restricted Boltzmann Machines* (RBMs) [279], where a hidden layer of the preceding RBM serves as the visible layer of the following RBM. At the top, a regression layer with linear units is added to the RBM stacks. This method has been proven to be simple and effective in many ap-

plications. The activation of each unit can be formulated as: $I_{(n),i} = \sum_j w_{(n),ij} o_{(n-1),j} + b_{(n),j}$, where $I_{(n),i}$ is the input of the ith unit in the nth layer. $o_{(n-1),j}$ is the output of the jth unit from the $(n-1)^{\text{th}}$ layer. $b_{(n),i}$ is the bias of the jth unit in the nth layer. The distribution of $I_{(n),i}$ varies during the training as the parameters of the previous layers change. This issue downgrades the learning rates, requires a more dedicated parameter initialization, and makes it hard to train models with saturating nonlinearities. To deal with this problem, we employ a batch normalization strategy as proposed in [280]. This method performs the normalization over a part of the model's architecture (orange blocks in the inset) as:



$$\widetilde{x}_{(n),i} = \frac{x_{(n),i} - \mu_{(n),i}}{\sqrt{\sigma^2_{(n),i} + \epsilon}}, \quad x_{(n),i} = \sum_j w_{(n),ij} o_{(n-1),j} + b_{(n),j}, \tag{6.1}$$

and the final output becomes:

$$x_{(n),i} = \gamma_{(n),i} \widetilde{x}_{(n),i} + \beta_{(n),i}. \tag{6.2}$$

Here, $\mu_{(n),i}$ and $\sigma^2_{(n),i}$ are the mean and variance of $x_{(n),i}$. $\gamma_{(n),i}$ and $\beta_{(n),i}$ are scaling and shifting parameters applied to the normalized $\widetilde{x}_{(n),i}$. All the parameters are

evolved using the iterative momentum gradient method as:

$$
\begin{aligned}
&\mathbf{W}^{i+1} = \mathbf{W}^{i} + \Delta\mathbf{W}^{i+1} && \Delta\mathbf{W}^{i+1} = d \cdot \Delta\mathbf{W}^{i} - \eta \cdot \frac{\partial L}{\partial \mathbf{W}} \\
&\mathbf{b}^{i+1} = \mathbf{b}^{i} + \Delta\mathbf{b}^{i+1} && \Delta\mathbf{b}^{i+1} = d \cdot \Delta\mathbf{b}^{i} - \eta \cdot \frac{\partial L}{\partial \mathbf{b}} \\
&\boldsymbol{\gamma}^{i+1} = \boldsymbol{\gamma}^{i} + \Delta\boldsymbol{\gamma}^{i+1} && \Delta\boldsymbol{\gamma}^{i+1} = d \cdot \Delta\boldsymbol{\gamma}^{i} - \eta \cdot \frac{\partial L}{\partial \boldsymbol{\gamma}} \\
&\boldsymbol{\beta}^{i+1} = \boldsymbol{\beta}^{i} + \Delta\boldsymbol{\beta}^{i+1} && \Delta\boldsymbol{\beta}^{i+1} = d \cdot \Delta\boldsymbol{\beta}^{i} - \eta \cdot \frac{\partial L}{\partial \boldsymbol{\beta}} \\
&\boldsymbol{\mu}^{i+1} = \boldsymbol{\mu}^{i} + \Delta\boldsymbol{\mu}^{i+1} && \Delta\boldsymbol{\mu}^{i+1} = d \cdot \Delta\boldsymbol{\mu}^{i} - \eta \cdot \frac{\partial L}{\partial \boldsymbol{\beta}} \\
&\boldsymbol{\sigma}^{2^{i+1}} = \boldsymbol{\sigma}^{2^{i}} + \Delta\boldsymbol{\sigma}^{2^{i+1}} && \Delta\boldsymbol{\sigma}^{2^{i+1}} = d \cdot \Delta\boldsymbol{\beta}^{i} - \eta \cdot \frac{\partial L}{\partial \boldsymbol{\beta}},
\end{aligned}
\tag{6.3}
$$

where $\mathbf{W}$, $\mathbf{b}$, $\boldsymbol{\gamma}$, $\boldsymbol{\beta}$, $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}^2$ are the aggregated training parameters w, b, γ, β, μ and σ^2 for a certain layer in the matrix/vector form. L is the loss over the training set. The superscript $[\cdot]^i$ indicates the iteration index. The partial derivatives of $\partial L/\partial \mathbf{W}$, $\partial L/\partial \mathbf{b}$, $\partial L/\partial \boldsymbol{\gamma}$, $\partial L/\boldsymbol{\beta}$, $\partial L/\boldsymbol{\mu}$ and $\partial L/\boldsymbol{\sigma}^2$ can be calculated using the backpropagation algorithm as:

$$
\begin{aligned}
&\frac{\partial L}{\partial \mathbf{W}_{(n)}} = \frac{1}{m}\sum_{l=1}^{m} \mathbf{o}_{(n-1)}^{(l)} \left(\frac{\partial L^{(l)}}{\partial \mathbf{x}_{(n)}^{(l)}}\right)^{\top}, \qquad \frac{\partial L}{\partial \mathbf{b}_{(n)}} = \frac{1}{m}\sum_{l=1}^{m} \left(\frac{\partial L^{(l)}}{\partial \mathbf{x}_{(n)}^{(l)}}\right)^{\top}, \\
&\frac{\partial L}{\partial \gamma_{(n),i}} = \sum_{l=1}^{m} \frac{\partial L^{(l)}}{\partial I_{(n),i}^{(l)}} \widetilde{x}_{(n),i}^{(l)}, \qquad \frac{\partial L}{\partial \beta_{(n),i}} = \sum_{l=1}^{m} \frac{\partial L^{(l)}}{\partial I_{(n),i}^{(l)}}, \\
&\frac{\partial L}{\partial \mu_{(n),i}} = -\frac{1}{m}\sum_{l=1}^{m} \frac{\partial L^{(l)}}{\partial \widetilde{x}_{(n),i}^{(l)}} \frac{1}{\sqrt{\sigma_{(n),i}^2 + \epsilon}} - \frac{2}{m}\sum_{l=1}^{m} \frac{\partial L^{(l)}}{\partial \sigma_{(n),i}^2} \left(x_{(n),i}^{(l)} - \mu_{(n),i}\right), \\
&\frac{\partial L}{\partial \sigma_{(n),i}^2} = \frac{1}{m}\sum_{l=1}^{m} \frac{\partial L^{(l)}}{\partial \sigma_{(n),i}^2},
\end{aligned}
\tag{6.4}
$$

where $\mathbf{o}_{(n)}^{(l)}$ is the output of the nth layer. The summation index l iterates all the m

samples in a mini-batch. Other intermediate partial derivatives can be computed as:

$$\frac{\partial L^{(l)}}{\partial \mathbf{x}_{(n)}^{(l)}} = \frac{\partial \mathbf{I}_{(n)}^{(l)}}{\partial x_{(n)}^{(l)}} \frac{\partial L^{(l)}}{\partial \mathbf{I}_{(n)}^{(l)}}, \qquad \frac{\partial I_{(n),i}^{(l)}}{\partial x_{(n),i}^{(l)}} = \frac{\gamma_{(n),i}}{\sqrt{\sigma_{(n),i}^2 + \epsilon}},$$
$$\frac{\partial L^{(l)}}{\partial \mathbf{I}_{(n-1)}^{(l)}} = \frac{\partial \mathbf{o}_{(n-1)}^{(l)}}{\partial \mathbf{I}_{(n-1)}} \mathbf{W}_{(n)} \frac{\partial \mathbf{I}_{(n)}^{(l)}}{\partial \mathbf{x}_{(n)}^{(l)}} \frac{\partial L^{(l)}}{\partial \mathbf{I}_{(n)}^{(l)}}, \qquad \frac{\partial L^{(l)}}{\partial \widetilde{x}_{(n),i}^{(l)}} = \frac{\partial L^{(l)}}{\partial I_{(n),i}^{(l)}} \gamma_{(n),i},$$
$$\frac{\partial L^{(l)}}{\partial \sigma_{(n),i}^2} = -\frac{1}{2} \frac{\partial L^{(l)}}{\partial \widetilde{x}_{(n),o}^{(l)}} \left(x_{(n),i}^{(l)} - \mu_{(n),i} \right) \left(\sigma_{(n),i}^2 + \epsilon \right)^{-\frac{3}{2}}.$$

The recorded speech signals are segmented into frames with a hanning window. Each frame contains a speech segment of 25 ms, and is encoded by the log-energy and 12-order Mel-frequency cepstral coefficients (MFCC) augmented with their delta and delta-deltas. The frame shift between consecutive frames is 10 ms to match the sampling rate of EMA sensors. The dataset is partitioned in three sets: a validation and a testing set of 110 utterances each, and a training set consisting of the other 880 utterances. Both EMA and MFCC feature vectors are normalized by subtracting their global mean and dividing by the standard deviation of each dimension, respectively.

6.6 Real-time Deformable Simulation of Tongue

The tetrahedral finite element mesh used for the 3D tongue model consists of $11,083$ nodal points and $56,794$ tetrahedral elements. Simulating such high-dimensional mesh with over 30K DOFs at the rate in sync with the acoustic input is challenging. Yang and colleagues [247] adopted a simplified dynamic model extending the linear elasticity using the modal warping technique [76] to alleviate this problem. Unfortunately, we found that the simulator was only able to generate plausible results for repeated CV trainings (e.g. [ta]–[ta]–[ta]). It often produced unnatural motion patterns for real speech production of complete words and sentences. The reasons are twofold. First, the modal warping method is still based on linear elasticity and its nonlinear

deformation comes from a geometric warping correction, which is not physics-based. Second, modal analysis constructs global subspace basis vectors while during language production, the tongue's deformation could be highly nonlinear and localized. As reported in the previous study [281], the human tongue undergoes a compression up to $\sim 200\%$ and an elongation up to $\sim 160\%$ when producing certain speech sounds. In other words, we need a new numerical framework that is able to perform the deformation integration in real time and effectively capture the nonlinear local deformations. In this section, we detail a novel *spatial reduction* method that allocates nonlinear simulation DOFs via *quadratic domains*. Each domain houses 30 DOFs grouped into 3 translation DOFs, 9 affine DOFs, 9 quadratic homogenous DOFs, as well as 9 quadratic heterogenous DOFs. We assign each EMA sensor a domain and an additional one for the tongue's interior in order to capture local deformation nearby the sensor while keeping the overall simulation in a low-dimensional subspace. Our model also fully addresses the volume preserving constraint rather than relying on tweaking Poisson's ratio as in [247].

Kinematics For a given material point $\mathcal{P}$ on the tongue model, we denote $\mathbf{x} = [x_1, x_2, x_3]^\top$ and $\mathbf{u} = [u_1, u_2, u_3]^\top$ as its rest shape position and displacement. A nearby *domain* imposes a quadratic influence to its displacement components such that $u_i = \mathbf{x}^\top \mathbf{Q}_i \mathbf{x} + \mathbf{a}_i^\top \mathbf{x} + t_i$ for $i = 1, 2, 3$. $\mathbf{Q}_i \in \mathbb{R}^{3\times 3}$ is a symmetric tensor encoding the iso-quadratic DOFs. We put its three diagonal DOFs into a vector such that $\mathbf{q}_{o_i} = [Q_{11}, Q_{22}, Q_{33}]^\top$ and refer to it as *homogenous* DOFs. Similarly, the vector $\mathbf{q}_{e_i} = [2Q_{12}, 2Q_{23}, 2Q_{13}]^\top$ containing off-diagonal elements of $\mathbf{Q}_i$ is referred to as *heterogenous* DOFs. The *affine* DOFs $\mathbf{a} \in \mathbb{R}^3$ describes how u_i is linearly related to its rest position, and t_i is a *translation* DOF. Each type of deformable DOFs from different domains are convexly combined, and the displacement of $\mathcal{P}$ can be written

as:

$$u_i = \sum_j w_t^j(\mathbf{x}) t_i^j + w_a^j(\mathbf{x}) \mathbf{a}_i^{j^\top} \mathbf{x} + w_o^j(\mathbf{x}) \mathbf{q}_{o_i}^{j^\top} \widetilde{\mathbf{x}} + w_e^j(\mathbf{x}) \mathbf{q}_{e_i}^{j^\top} \widehat{\mathbf{x}}, \tag{6.5}$$

where w_t^j, w_a^j, w_o^j and w_e^j are location-dependent weight coefficients indicating how much domain j affects different types of deformable DOFs. $\widetilde{\mathbf{x}} = [x_1^2, x_2^2, x_3^2]^\top$ and $\widehat{\mathbf{x}} = [x_1x_2, x_2x_3, x_1x_3]^\top$ are second-order homogenous and heterogenous vectors of $\mathcal{P}$. By stacking all the deformable DOFs from the jth domain into a single vector $\mathbf{q}^j \in \mathbb{R}^{30}$ such that $\mathbf{q}^j = [\mathbf{t}^{j^\top}, \mathbf{a}_1^{j^\top}, \mathbf{a}_2^{j^\top}, \mathbf{a}_3^{j^\top}, \mathbf{q}_{o_1}^{j^\top}, \mathbf{q}_{o_2}^{j^\top}, \mathbf{q}_{o_3}^{j^\top}, \mathbf{q}_{e_1}^{j^\top}, \mathbf{q}_{e_2}^{j^\top}, \mathbf{q}_{e_3}^{j^\top}]^\top$, the displacement of $\mathcal{P}$ can be concisely expressed as a matrix-vector product:

$$\mathbf{u} = \mathbf{G}^j \mathbf{q}^j = \left[\mathbf{G}_t^j | \mathbf{G}_a^j | \mathbf{G}_o^j | \mathbf{G}_e^j\right] \mathbf{q}^j, \tag{6.6}$$

where

$$\begin{aligned} \mathbf{G}_t^j &= w_t^j \mathbf{I} & \mathbf{G}_a^j &= w_a^j \mathbf{I} \otimes \mathbf{x}^\top \\ \mathbf{G}_o^j &= w_o^j \mathbf{I} \otimes \widetilde{\mathbf{x}}^\top & \mathbf{G}_e^j &= w_e^j \mathbf{I} \otimes \widehat{\mathbf{x}}^\top. \end{aligned}$$

We call matrix $\mathbf{G}^j$ the *geometric displacement matrix* as it depends soley on the rest shape of the tongue mesh. The generalized coordinate $\mathbf{q}^j$ uniquely determines the kinematic of $\mathcal{P}$:

$$\dot{\mathbf{u}} = \sum_j \mathbf{G}^j \dot{\mathbf{q}}^j, \qquad \ddot{\mathbf{u}} = \sum_j \mathbf{G}^j \ddot{\mathbf{q}}^j. \tag{6.7}$$

Reduced dynamics Let $\mathbf{e}_i$ denote the canonical basis vectors of $\mathbb{R}^3$, and we drop the domain superscript $[\cdot]^j$ in this paragraph for the sake of a succinct formulation. Based on Eq. (6.5), each row of the deformation gradient tensor $\mathbf{F} = [\mathbf{F}_1, \mathbf{F}_2, \mathbf{F}_3]^\top \in \mathbb{R}^{3\times3}$ can be written as $\mathbf{F}_i = \mathbf{F}_{t_i} + \mathbf{F}_{a_i} + \mathbf{F}_{o_i} + \mathbf{F}_{e_i} + \mathbf{e}_i$, where

$$\begin{aligned} \mathbf{F}_{t_i} &= \sum \nabla w_t t_i & \mathbf{F}_{a_i} &= \sum \mathbf{a}_i^\top \mathbf{x} \nabla w_a + w_a \mathbf{a}_i^\top \\ \mathbf{F}_{o_i} &= \sum \mathbf{q}_{o_i}^\top \widetilde{\mathbf{x}} \nabla w_o + w_o \mathbf{q}_{o_i}^\top \widetilde{\mathbf{X}} & \mathbf{F}_{e_i} &= \sum \mathbf{q}_{e_i}^\top \widehat{\mathbf{x}} \nabla w_e + w_e \mathbf{q}_{e_i}^\top \widehat{\mathbf{X}}, \end{aligned}$$

and

$$\widetilde{\mathbf{X}} = \begin{bmatrix} x_2 & x_1 & 0 \\ 0 & x_3 & x_2 \\ x_3 & 0 & x_1 \end{bmatrix}, \qquad \widehat{\mathbf{X}} = \begin{bmatrix} 2x_1 & 0 & 0 \\ 0 & 2x_2 & 0 \\ 0 & 0 & 2x_3 \end{bmatrix}.$$

Once we have computed $\mathbf{F}$, we can evaluate the nonlinear Green strain, $\mathbf{E} = \frac{1}{2}(\mathbf{F}^\top\mathbf{F} - \mathbf{I})$, and proceed to express the strain energy density Ψ as well as the first Piola-Kirchhoff stress tensor (PK1) based on the chosen material model. Previous research [273, 281, 282], indicates that an isotropic and homogenous material model for the tongue is applicable as the variation of Young's modulus at different parts of the tongue is very small. Accordingly we choose to use the St. Venant-Kirchhoff (StVK) model since it is capable of producing most desired nonlinear deformation effects of the tongue. The Young's modulus is set as $6,912$ and the Poisson's ratio is set as 0.49. Extending our method to accomodate other materials like Neo-Hookean is straightforward under our framework.

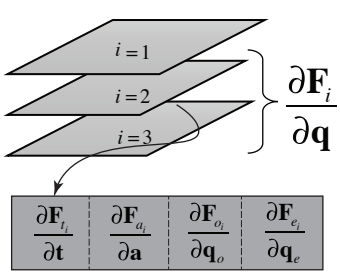
With the StVK material, the energy density and PK1 are formulated as: $\Psi = \mu\mathbf{E} : \mathbf{E} + \frac{\lambda}{2}\mathrm{tr}^2(\mathbf{E})$ and $\mathbf{P} = \mathbf{F}[2\mu\mathbf{E} + \lambda\mathrm{tr}(\mathbf{E})\mathbf{I}]$, respectively, where λ and μ are the Lamé parameters. The per-domain reduced internal force $\widetilde{\mathbf{f}}_{int}$ and its gradient $\partial\widetilde{\mathbf{f}}_{int}/\partial\mathbf{q}$ are computed as:

$$\widetilde{\mathbf{f}}_{int} = -\int \left(\mathbf{P}\frac{\partial\mathbf{F}}{\partial\mathbf{q}}\right) \mathrm{d}V, \tag{6.8}$$

and

$$\frac{\partial\widetilde{\mathbf{f}}_{int}}{\partial\mathbf{q}} = -\int \left[\left(\frac{\partial\mathbf{P}}{\partial\mathbf{F}}\frac{\partial\mathbf{F}}{\partial\mathbf{q}}\right)^\top \frac{\partial\mathbf{F}}{\partial\mathbf{q}}\right] \mathrm{d}V. \tag{6.9}$$

Here, $\partial\mathbf{F}/\partial\mathbf{q} \in \mathbb{R}^{3\times3\times30}$ is a third-order block-sparse tensor, which can be understood as the superposition of three layers as shown at right. The ith layer represents the matrix $\partial\mathbf{F}_i/\partial\mathbf{q}$ and it hosts four sub-matrices, namely $\partial\mathbf{F}_{t_i}/\partial\mathbf{t}$, $\partial\mathbf{F}_{a_i}/\partial\mathbf{a}$,



$\partial \mathbf{F}_{o_i}/\partial \mathbf{q}_o$, and $\partial \mathbf{F}_{e_i}/\partial \mathbf{a}_e$. All of these sub-matrices are block-sparse as the partial derivative yields a nonzero block only when the subscript of the generalized coordinates agree. Each nonzero block can be easily calculated as:

$$\begin{aligned} \frac{\partial \mathbf{F}_{t_i}}{\partial t_i} &= \nabla w_t & \frac{\partial \mathbf{F}_{a_i}}{\partial \mathbf{a}_i} &= \nabla w_a \otimes \mathbf{x} + w_a \mathbf{I} \\ \frac{\partial \mathbf{F}_{o_i}}{\partial \mathbf{q}_{o_i}} &= \nabla w_o \otimes \widetilde{\mathbf{x}} + w_o \widetilde{\mathbf{X}}^\top & \frac{\partial \mathbf{F}_{e_i}}{\partial \mathbf{q}_{e_i}} &= \nabla w_e \otimes \widehat{\mathbf{x}} + w_e \widehat{\mathbf{X}}^\top. \end{aligned} \tag{6.10}$$

Applying temporal discretization using the implicit Euler integration leads to the final nonlinear system to be solved at each time step:

$$\left(\widetilde{\mathbf{M}} - h\widetilde{\mathbf{C}} - h^2 \frac{\partial \widetilde{\mathbf{f}}_{int}}{\partial \mathbf{q}} \right) \Delta \dot{\mathbf{q}} = h\widetilde{\mathbf{f}}_{ext} + h^2 \frac{\partial \widetilde{\mathbf{f}}_{int}}{\partial \mathbf{q}} \dot{\mathbf{q}}, \tag{6.11}$$

where $\widetilde{\mathbf{M}}$ is the reduced mass matrix, which can be evaluated block-wisely: $\widetilde{\mathbf{M}}^{ij} = \int \rho \mathbf{G}^{i\top} \mathbf{G}^j \mathrm{d}V$ ($\rho = 1$ as the tongue consists of mostly water); $\widetilde{\mathbf{f}}_{ext}$ is the generalized external force; h is the time step size; and $\widetilde{\mathbf{C}}$ is the reduced damping matrix.

6.7 Domains' Weight Coefficients

Analogous to shape functions used in the standard FEM that blend nodal quantities volumetrically within an element, weighting functions superimpose quadratic transformations from domains and yield global deformations of the tongue. The

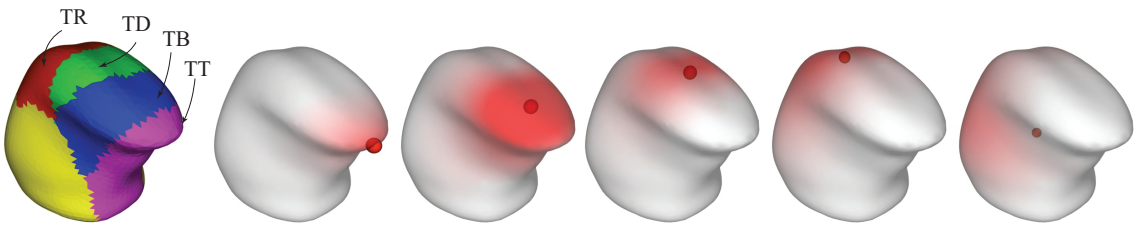


Figure 6.6: The domain partition of the input tongue mesh as well as the weight distribution for each domain.

domain subspace should be material-customized. To this end, we present an efficient algorithm to calculate the weight distribution for each domain to accurately reflect the material properties of the tongue and augment the geometric displacement matrix (Eq. (6.6)). Our method is *fully* material-and-geometric-aware, and possesses important traits such as locality, smoothness and interpolation.

Intuitively, the weighting function $w(\mathbf{x})$ ought to align with the visual impression of how the deformation fades away from the seed of the domain, where the maximum local displacement occurs. Apparently, a straightforward way to obtain such deformation dissipation is to solve a static equilibrium by imposing an external force $\mathbf{f}_s \in \mathbb{R}^3$ at the domain seed and anchoring all other seeds. Unfortunately, this problem is ill-defined as we have infinite choices of $\mathbf{f}_s$ – obviously they give different weighting distributions when used.

We resolve this ambiguity by restricting $\mathbf{f}_s$ along the *principle direction* $\mathbf{p}$ of a domain, which can be understood as the "most deformable direction" such that the domain undergoes the largest displacements when $\mathbf{f}_s$ aligns with it (i.e. $\mathbf{f}_s = \mathbf{p}$). Mathematically, it can be formulated as a quadratically constrained quadratic program (QCQP) problem as:

$$\begin{aligned} &\underset{\mathbf{p}}{\text{maximize}} && |\mathbf{u}|^2 \\ &\text{subject to} && \begin{bmatrix} \mathbf{K} & \mathbf{B}_a^\top \\ \mathbf{B}_a & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \lambda \end{bmatrix} = \begin{bmatrix} \mathbf{B}_s^\top \mathbf{p} \\ \mathbf{0} \end{bmatrix}, \\ &\text{and} && |\mathbf{p}|^2 = 1, \end{aligned} \tag{6.12}$$

where $\mathbf{B}_s$ and $\mathbf{B}_a$ are two binary matrices picking the domain's seed on which $\mathbf{f}_s$ is applied, and anchor seeds to incorporate boundary conditions. λ is the unknown multipliers. In general, QCQP is NP-hard and a polynomial-time solution may not be available. Fortunately as Eq. (6.12) only activates equality constraints, it can be

directly solved. To do so, we explicitly write down its inversion:

$$\begin{bmatrix} \mathbf{u}_p \\ \mathbf{u}_w \\ \lambda \end{bmatrix} = \begin{bmatrix} \mathbf{H}_{pp} & \mathbf{H}_{pw} & \mathbf{H}_{p\lambda} \\ \mathbf{H}_{pw}^\top & \mathbf{H}_{ww} & \mathbf{H}_{w\lambda} \\ \mathbf{H}_{p\lambda}^\top & \mathbf{H}_{w\lambda}^\top & \mathbf{H}_{\lambda\lambda} \end{bmatrix} \begin{bmatrix} \mathbf{p} \\ \mathbf{0}_w \\ \mathbf{0}_\lambda \end{bmatrix}. \tag{6.13}$$

Here, the matrix $\mathbf{H}$ is the inverse of the constrained stiffness matrix in Eq. (6.12), which is often referred to as the *flexibility matrix*. We arrange all DOFs into groups such that subscript p is for 3 DOFs associated with the seed where the unit force $\mathbf{f}_s = \mathbf{p}$ is applied; subscript w is associated with all other nodal DOFs for which the weighting is to be calculated; and subscript λ is for multipliers' DOFs. Spanning the first rows in Eq. (6.13), the target function can be simplified:

$$\begin{aligned} |\mathbf{u}|^2 &= \mathbf{p}^\top (\mathbf{H}_{pp}\mathbf{H}_{pp} + \mathbf{H}_{pw}\mathbf{H}_{pw}^\top)\mathbf{p} \\ &\triangleq \mathbf{p}^\top \mathbf{A}\mathbf{p}. \end{aligned} \tag{6.14}$$

Note that $\mathbf{A}$ is SPD, and we diagonalize it using the eigen decomposition: $\mathbf{D} = \mathrm{diag}(d_1, d_2, d_3) = \mathbf{\Phi}^\top \mathbf{A}\mathbf{\Phi}$, $d_1 \leq d2 \leq d3$ leading to:

$$\begin{aligned} |\mathbf{u}|^2 &= (\mathbf{\Phi}\mathbf{p})^\top \mathrm{diag}(d_1, d_2, d_3)\,(\mathbf{\Phi}\mathbf{p})\,. \\ &= d_1 p_1^2 + d_2 p_2^2 + d_3 p_3^2 \\ &\leq d_3 \qquad\qquad \text{(by the fact that } |\mathbf{p}|^2 = p_1^2 + p_2^2 + p_3^2 = 1\text{).} \end{aligned}$$

The maximum value of $|\mathbf{u}|^2$ will be obtained when $\mathbf{p} = \boldsymbol{\phi}_3$, the eigenvector corresponding to the largest eigenvalue of $\mathbf{A}$.

After $\mathbf{p}$ is ready, the weighting function over the domain can be numerically computed by prescribing $\mathbf{p}$ as the constrained displacement. We notice that completely anchoring all the neighbor seeds as well as the domain boundary (i.e. fixing all of their x, y and z freedoms) produces an over-damped weighting. Accordingly, we lift up the boundary condition and only restrict their displacements along the principle direction, while tangential movements towards $\mathbf{p}$ are still allowed. In other words,

each 3 by 3 sub-block of an identity matrix in $\mathbf{B}_a$ corresponding to a anchor node on the mesh is changed to $\mathbf{p}^\top$. Fig. 6.7 left shows the comparative results of a standard bending simulation of a load-end cantilever beam using different weighting functions. It can be seen that our method yields a natural and smooth nonlinear bending.

We subdivide the tongue mesh into five domains as shown in Fig. 6.6. Four of them are seeded at the corresponding EMA sensors. The fifth one is at the mass center of the tongue mesh. The domain's partition is obtained by performing flooding from the seed. The weight distribution of the domains is visualized using the red-white color map.

6.8 Preserving Volume within the Subspace

As a muscular organ, the human tongue consists of 99% of water, which preserves its volume during speech production. To achieve this effect, we introduce an auxiliary pressure variable which provides a volume adjustment of both displacement and velocity vectors to the simulated tongue mesh at each frame. This method has been explored by the computer graphics community [283, 284]. However, a fullspace displacment/velocity amendment is needed to solve the pressure terms for each element, which is $\mathbf{O}(n^2)$ at runtime and downgrades the performance of the real time simulation. We leverage the fact that the nonlinear tongue deformation driven by EMA sensors is of a low rank and efficiently handle the volume preserving constraint in a reduced space.

Let V_0 denote the volume of the original tongue mesh Ω. It can be computed as $V_0 = \int_\Omega \mathrm{d}\mathbf{x}$. Its deformed volume V_t at time t can be calculated similarly as: $V_t = \int_{\Omega_t} \mathrm{d}\mathbf{y} = \int_\Omega |\mathbf{F}(\mathbf{x})| \mathrm{d}\mathbf{x}$, where $\mathbf{y} = \mathbf{x} + \mathbf{u}$ is the deformed nodal position. Noting that $|\mathbf{F}| = |\mathbf{I} + \nabla\mathbf{u}| \approx 1 + \mathtt{div}\,\mathbf{u}$, the volume change between V_0 and V_t can be

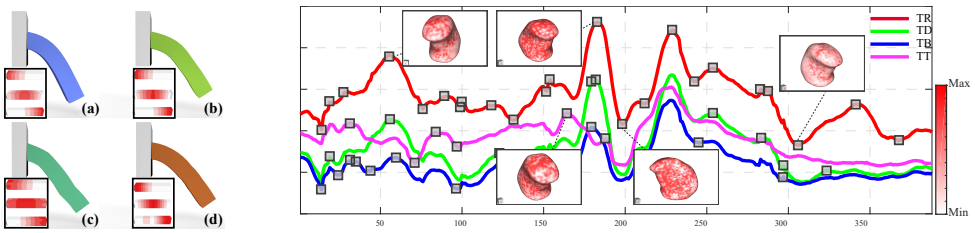


Figure 6.7: **Left:** In this illustrative example, we compare a simple bending simulation of a standard load-end cantilever beam (with three domains) using (a) our method, (b) weighting computed with a completely fixed boundary condition, (c) weighting computed along the direction perpendicular to the principle direction, and (d) using harmonic coordinates. **Right:** We pick key frames (grey blocks) by checking the finite difference acceleration magnitude of each EMA sensor and compute the corresponding pressure field. Selected pressure vectors and the corresponding tongue shapes are visualized as well using the red-white color map.

first-order approximated as:

$$\Delta V = V_t - V_0 = \int_\Omega (|\mathbf{F}(\mathbf{x})| - 1)\mathrm{d}\mathbf{x} \approx \int_\Omega \mathtt{div}\,\mathbf{u}\mathrm{d}\mathbf{x}. \tag{6.15}$$

We introduce a virtual pressure term $\mathbf{p} \in \mathbb{R}^n$ to cancel ΔV which results in a "pressure force" of $-\nabla\mathbf{p}$. Inserting $-\nabla\mathbf{p}$ into the time integration yields:

$$\Delta V \approx \mathtt{div}\,\left(\mathbf{M}^{-1}\nabla\mathbf{p} - \mathbf{v}\right) \cdot h, \tag{6.16}$$

where h is the time step size, which is set as 0.01 in our system. Discretizing Eq. (6.16) at each tetrahedral element on the mesh allows us to solve $\mathbf{p}$ by inverting an n by n matrix. This matrix is constant and can be pre-factorized. Thus, the runtime evaluation of $\mathbf{p}$ requires a complete $\mathbf{O}(n^2)$ forward-backward substitution at each time step. The resulting $\mathbf{p}$ is used to obtain a displacement correction $\Delta\mathbf{u} = -h \cdot \mathbf{M}^{-1}\nabla\mathbf{p}$. We follow the same idea as Irving and colleagues [283] and apply another velocity correction $\Delta\mathbf{v}$ to make the velocity field as divergence-free as possible. Doing so effectively stabilizes potential oscillations under nonlinear constraints.

Data-driven pressure subspace Since the deformable motion of the tongue driven by the EMA coils is obviously of low rank, we further construct a reduced

pressure subspace and solve Eq. (6.16) within the subspace. Our method is data-driven, based on the recorded articulatory corpus consisting of $1,108$ complete Chinese sentences. Each EMA frame i includes a vector $\mathbf{s}_i$ of 3D positions of TT, TB, TD and TR sensors. We evaluate the finite difference acceleration as: $(\mathbf{s}_{i+1}+\mathbf{s}_{i-1}-2\mathbf{s}_i)/2\Delta t^2$ to obtain the inflection points of each sensor's trajectory. Frames with small acceleration magnitude are chosen as key frames (Fig. 6.7 right). While there are hundreds of thousands of EMA frames, we found that using 100 key frames is sufficient to construct a high-quality subspace for the volume correction.

For each selected key frame k_i, we solve a nonlinear static equilibrium by imposing constraint forces at nodal points corresponding to EMA sensors under the volume preserving constraint, and record the associated correcting pressure vector $\mathbf{p}_{k_i}$ such that:

$$\begin{bmatrix} \mathbf{K}(\mathbf{u}) & \mathbf{C}^\top(\mathbf{u}) \\ \mathbf{C}(\mathbf{u}) & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \lambda \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ \mathbf{0} \end{bmatrix}, \tag{6.17}$$

where $\mathbf{C}(\mathbf{u})$ encodes the required nonlinear constraints for both position constraints at EMA sensors and volume preservation. Newton's method is used to solve this nonlinear problem, which requires the evaluation of the tangent stillness matrix $\mathbf{K}(\mathbf{u}^i)$ at each intermediate $\mathbf{u}^i$ of the ith iteration as well as the Jacobi of the constraint matrix, which can be calculated as $\nabla\mathbf{C} = [\mathbf{B}_s^\top\, \mathbf{D}]^\top$. Here $\mathbf{B}_s$ is a constant binary matrix picking the nodal DOFs corresponding to the EMA sensors. $\mathbf{D} \in \mathbb{R}^{n\times 3n}$ is the discretized matrix representation of the `div` operator. In the dynamic integration, the first-order approximation of the volume change (e.g. Eq. (6.15)) is applied to an incremental displacement update occurring within a single time step. Together with the velocity correction, the volume preserving constraint can always be well satisfied. However, in our subspace construction a given key frame often corresponds to a deformed configuration of the tongue deviating significantly from the rest shape. Indeed, solving Eq. (6.16) is numerically equivalent to performing one Newton iteration to solve the nonlinear system. Typically, we need three to five iterations to fully

suppress ΔV. Finally, a modified Gram-Schmidt process (MGS) is applied to all the computed $\mathbf{p}_{k_i}$, which serves as the basis vectors for the pressure subspace. In this manner the displacement and velocity corrections can be efficiently calculated within milliseconds and impose a nominal computation penalty to the simulator.

Other implementation details We employ Rayleigh damping and the mass and stiffness damping coefficients are set as 6.22 and 0.11 as reported in [281, 282]. In order to efficiently evaluate the reduced internal force and its gradient, we use the Cubature scheme proposed by An and colleagues [285]. The idea is to avoid evaluating $\widetilde{\mathbf{f}}_{int}$ and $\partial\widetilde{\mathbf{f}}_{int}/\partial\mathbf{q}$ at each element, which is a $\mathbf{O}(n)$ runtime procedure. Instead, the Cubature scheme selects a set of few key elements and approximates them as the weighted summation of per-element internal force and force gradient. We refer the reader to the related documents [285, 286] for a detailed exposition of the Cubature method. In our implementation, the training data for the Cubature is selected in a similar way as for constructing the pressure subspace, yet consists of 500 training poses. The DNN-based speech inversion feeds Eq. (6.11) the positional information of the TT, TB, TD and TR sensors based on the acoustic signals. We use the Lagrange multiplier method to deform the tongue mesh so that positional constraints can be precisely satisfied.

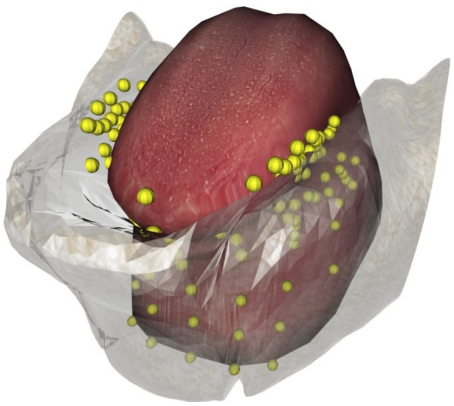


Figure 6.8: Collision detection is limited at few selected collision points.

We monitor the collisions between the tongue and the jaw. As the collision patterns between them are highly coherent, the collision detector simply tracks only selected collision points as shown in Fig. 6.8. If a collision is detected, a damped spring is applied to resolve it.

6.9 Experimental Result

In addition to the data acquisition equipment described in § 6.4, the numerical part of our framework was implemented using `Microsoft Visual C++ 2013` on a desktop PC with an `Intel i7-5960` CPU and 32GB of DDR4 RAM. The GUI was implemented using `QT`. All the numerical algorithms were implemented using the `Eigen C++` template (the Cholesky `LDLT` routine is used for solving Eq. (6.11)). Our simulation runs at 60+ FPS including collision and volume preservation.

Evaluation of the speech inversion During the speech inversion, in order to determine the number of hidden units of each layer, we first conduct an experiment on a neural network with one hidden layer. The number of hidden units varies from 50 to $1,600$. The results indicate that the neural network with 400 hidden units should achieve a good performance. Therefore, we construct a deep neural network with 6 hidden layers. The momentum d (e.g. in Eq. (6.3)) is set to be 0.8. The initial learning rate is set to be 0.0004, and decays with the proportion of 0.9. Each mini-batch contains $1,024$ examples. The maximum number of training epoch is set to be 50. The evaluation of the DNN-based speech inversion is performed over the 110 utterances out of the collected corpus that do not participate in the DNN training. We compute the root mean-squared error (RMSE) defined as: $\epsilon_{RMSE} = \sqrt{\frac{1}{m}\sum_{i=1}^{m} |\widetilde{\mathbf{x}} - \mathbf{x}|^2}$, for each EMA sensor to see how much deviation we have between the sensor's position and DNN trained model. Here, $m = 110$. $\widetilde{\mathbf{x}}$ and $\mathbf{x}$ are the sensors' positions as the output from the DNN and their observed coordinates. ϵ_{RMSE} for all the sensors is less than 3 mm. Specifically, the deviations are 2.93 mm for the TR sensor, 2.56 mm for the TD sensor, 1.2 mm for the TB sensor, and 0.87 mm for the TT sensor. We also compute the cross correlation coefficient to evaluate similarity of the motion trajectories. The correlation coefficient between the trained and real EMA sensors' trajectory is 0.81. In addition, we further applied our DNN to the MOCHA database [287]. Our DNN

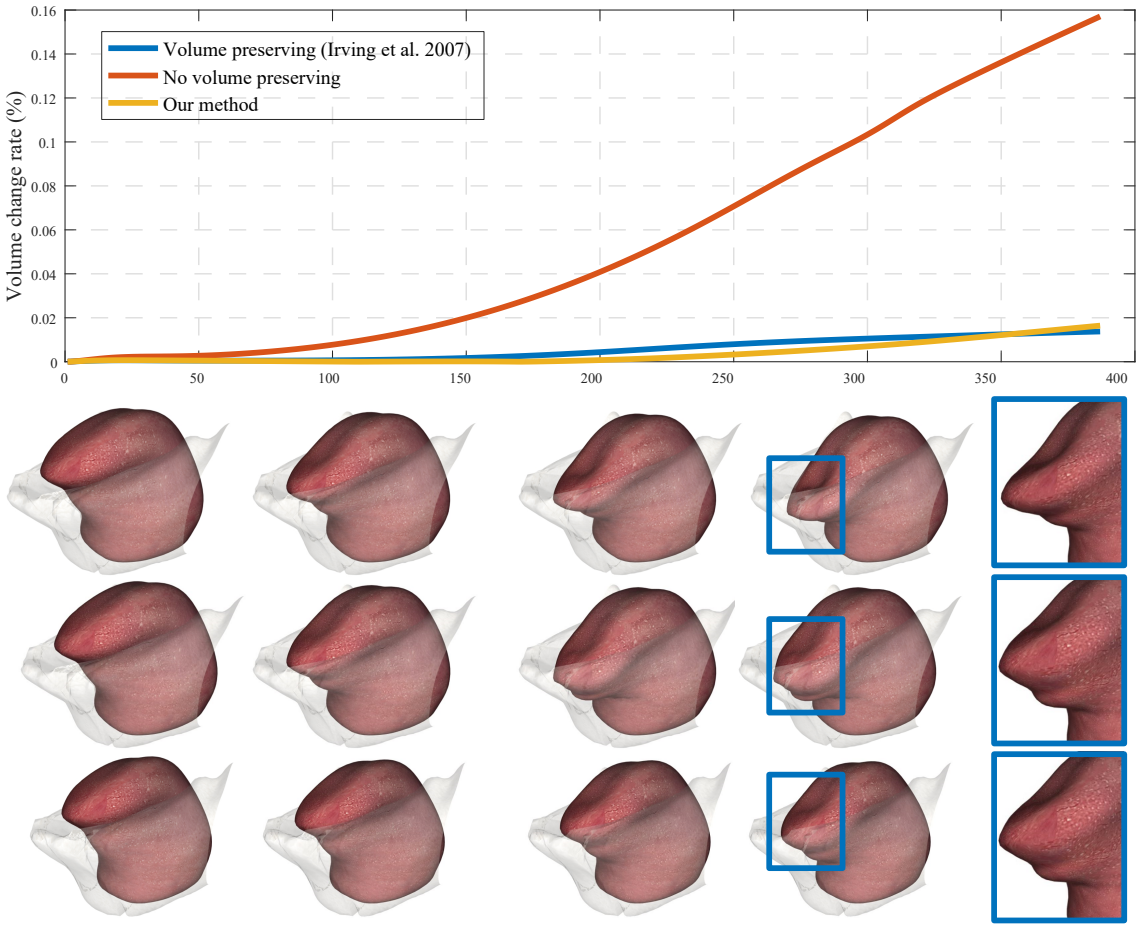


Figure 6.9: Applying the volume preserving constraint yields more natural tongue shapes, and our subspace volume preserving is able to effectively suppress volume change during tongue's deformation. The first row of snapshots is the shapes without volume preserving. The second and the third rows are the results using fullspace method and our method. It can be seen that our method is able to produce almost identical results compared to the fullspace volume correction. With our method, the volume change during the tongue simulation is always less than 2%.

model produces comparable results (average $\epsilon_{RMSE} = 1.09\ mm$ and correlation is 0.89) as other paradigms for the inverse speech mapping (e.g. in [288]).

Evaluation of subspace volume preserving Next, we quantitatively evaluate the performance of the proposed subspace volume preserving method. Fig. 6.9 shows

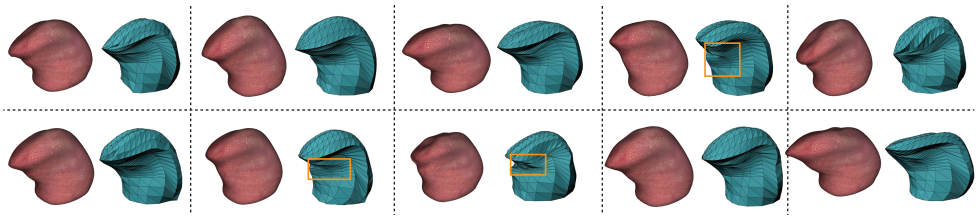


Figure 6.10: Side-by-side comparisons between simulated tongue shapes (textured) and real-world shapes extracted from MRI-CBCT fused images (in cyan)

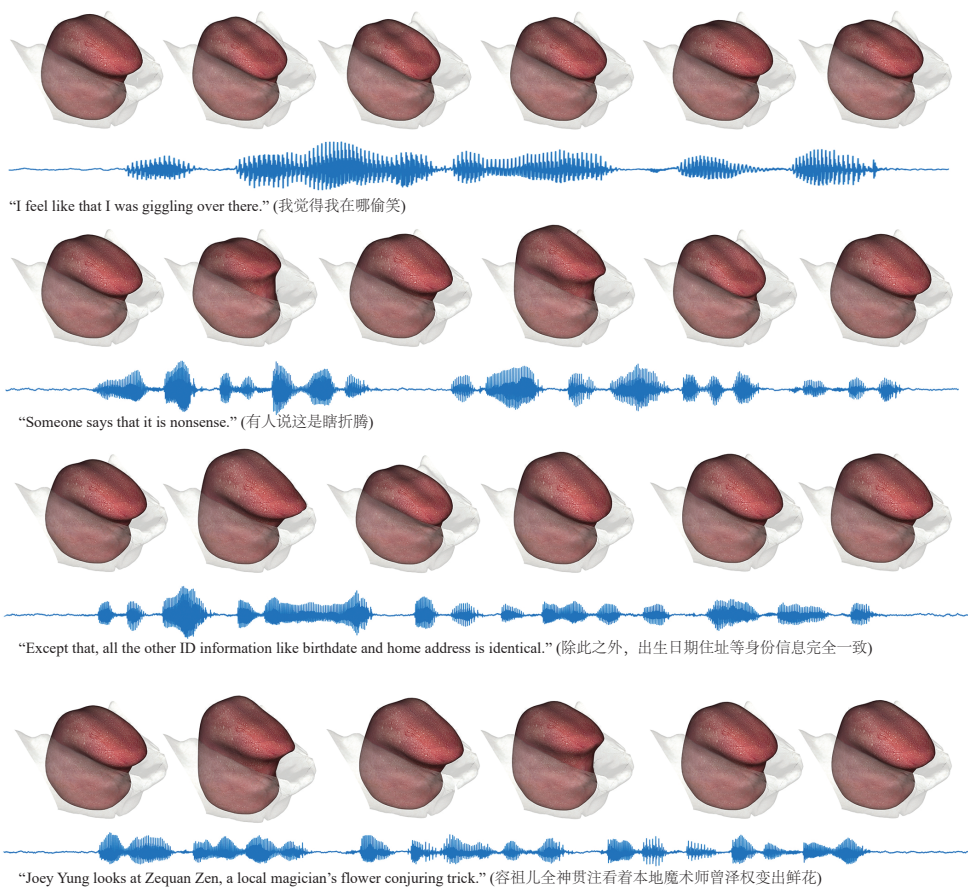


Figure 6.11: More snapshots of the tongue during speech production. The input acoustic signal waves are also provided.

a representative example of a deformable tongue motion when the entire tongue mesh is moving downwards. Without enforcing the volume conservation constraint, the

volume change of the entire mesh can be as high as 15%. Fullspace volume preservation as in [283] is able to correct this issue but takes $500-600\ ms$ to solve Eq. (6.16) in the fullspace. This correction must be calculated twice for both corrective displacement and velocity. Our subspace volume preserving algorithm can be completed less than 20 ms and the visual difference between the fullspace volume preservation and our method is indistinguishable.

Evaluation of the FEM simulator Evaluating the resulting deformed tongue shapes is crucial for us to understand the quality of the proposed simulator. However, there does not exist a "gold standard" that could serve as the ground truth to conclusively tell if a given tongue's shape is valid or not. Indeed, the current knowledge of tongue placement is quite limited even for certificated Speech-Language Pathologists (SLP) [289]. Previous work on the tongue modeling borrowed experiences from domain experts and conducted qualitative visual evaluations [247]. While such approach is able to more or less estimate the quality of the visual tongue model, it is highly subjective. Occasionally, even certificated SLPs are not able to tell if a motion looks "right" or not. In this work, thanks to the various medical imaging systems adopted, we are able to quantitatively evaluate the quality of the simulated tongue shapes. Our ground truth is obtained in a similar manner as for constructing a subject-specific tongue mesh (§ 6.4) by fusing and aligning both MRI and CBCT volumes. We compared the simulated shapes and the ones extracted from the 3D images for 10 representative poses. To the best of our knowledge, it is the first quantitative evaluation for FEM based tongue models that leverages full 3D real-world data. Fig. 6.10 reports the side-by-side comparison. We also computed the Hausdorff distance [290] between the simulated shape and captured ones. We first eliminate the shape differences induced by uniform scaling and rigid body transformation. To do so, a shape matching [291] is performed to find the optimal rotation between a pair of meshes (i.e. the simulated and captured ones). After that, a scaling factor

s can be computed, and the Hausdorff distance is evaluated finally. The average shape difference is less than 5% of for all the 10 examples listed in Fig. 6.10. Visually, our simulator replicates real-world tongue shapes plausibly. However, it can also be seen from this comparison that localized denting deformation under the tongue tip is not well captured, which occurs due to the contraction of the underlaying muscle group. As our DOF assignment and domain decomposition are based on the placement of EMA sensors, and such localized sharp deformation is probably beyond our subspace expressivity. After all, we only use 150 DOFs to simulate the complex nonlinear motion of the tongue (over 30K fullspace DOFs). More results are reported in Fig. 6.11. The sound waves along with the original Chinese language and the corresponding English translations are also provided. We refer readers to the accompanying video for more animated results.

Chapter 7

Conclusion and Limitation

The dissertation presents a set of novel tools and algorithms to generate real-time realistic animations. The dissertation presents the idea of how to design a good subspace for tubular structures in chapter 2. We analyze the geometry and general utility of the tubular support structure and find that in most cases the internal DOFs can be decided by boundary DOFs. The founding tells us the simulation does not need the full DOFs and the number of DOFs in the system can be condensed, thus the simulation can be accelerated. Unlike the case of chapter 2, chapter 3 provides a more general model reduction method in solid dynamics simulation. The method reduces the number of DOFs by dropping carefully selected high-frequency shapes. The overlapping quadratic domains with elastic weight distribution are used in our simulator to achieve a robust result under large deformation. We also introduce two directions to simplify the framework of nonlinear solid simulation. One is using the accelerated complex-step finite difference to simplify the programming of derivatives of the energy function. Another one is utilizing neural networks to correct linear elastic deformations into corresponding nonlinear elastic deformations. At last, we leverage our model reduction method to a tongue visualization system to provide real-time response tongue animation along with the input speech voice signal.

The potential of these methods in computer graphics is still not fully explored and gives us further directions to explore in the future. For the tubular structure editing, we could provide more complex tubes of T-shape or Y-shape which are not able to be intuitively edited currently. As future work, we will study how to use fundamental solutions to accelerate the geometric design of multi-interface tube components. Also, our system should have the ability to solve the self-intersection automatically. Feature modeling is not supported in our system. In the simulation part, we ignore the deflection induced by the component's self-weight, which could also induce simulation inaccuracy when the tubular component is fabricated using material of high specific modulus. We plan to incorporate the inertia-relief modes [8] to fully accommodate the gravity effect on the system in the future. Another interest future direction is to make simulation active meaning the simulator will provide the user potential solutions to fix a "faulty" geometric edit as in many recent design-simulation systems [38].

For our reduced simulator, the method also has several limitations. At first, while quadratic transformations provide plenty of nonlinear freedoms, they could also inject excessive DOFs for modest deformations. As a result, placing a lot of quadratic domains (i.e. over hundreds of domains as in [1]) will quickly drop simulation FPS. A possible solution is to explore the geometric symmetry/degeneration hidden in the deformable body to further condense the domain's DOFs (i.e. downgrade entries in the geometry matrix to linear DOFs that are perpendicular to the neutral axis of a beam, where we have limited nonlinear deformations). Another possible treatment is to use mixed domains, like affine [3] or rigid [4] domains. Adding new domains during the simulation runtime alters the subspace matrix and popping artifacts are possible if the time step size is aggressive. For instance in [79], the time step is set conservatively at the order of $1e^{-4}$ to $1e^{-6}$ to alleviate the issue. Another limitation lies in the fact that our weight function is still computed based on the linear elasticity and the rest shape stiffness matrix. Under large deformations, the weight distribution is likely to change too. We will look into the possibility of calculating the spatial weight

derivative similar to the modal derivative [59] to better incorporate such nonlinearity. Augmenting modal deformations with elastic weighting is also an interesting future work for us. In order to do so, we need to carefully design local boundary conditions to construct modal bases and couple them with local rigid body transformations. Of course, doing so will induce more freedoms to the simulator putting us back to the original question for the reduced simulation: how to find the best balance between simulation speed and quality?

The limitaion of our our CSFD and MCSFD is it requires a dedicated implementation in order to achieve a good performance. When CR form is used, the computation quickly becomes prohibitive if one wants to evaluate higher-order derivatives for a tensor-valued function. However, if the efficiency is not the primary concern, one can implement CSFD and MCSFD quickly based on any existing complex arithmetic library. In the future, we would like to fully leverage this new method to attack other challenging computational problems. For instance, to perform the imaginary perturbation along the time domain to get a better time integration. It is also of great interests to us to apply this method to machine learning and other similar problems where optimizing complicated functions is required.

For NNwarp, the first limitation, as a common drawback of learning-based methods, is the performance drops rapidly if an extrapolation is needed. In other words, if the training set does not cover the feature vectors that appear in the simulation, NNWarp may produce unrealistic deformations. In our current setting, we only consider isotropic hyperelastic materials. While we believe NNWarp should be able to handle more complicated anisotropic materials, doing so may require a re-design of contextual features and more training efforts since we cannot align training pairs within a local frame. We use directional and rotational force fields as the external forces in our current training data generation, both of which are low-frequency forces. As a result, NNWarp is less accurate when a high-frequency external force is applied

i.e. during the collision and contact. One may observe popping artifact when the bunny hits the floor in Fig. 5.17. A potential solution may be to use the idea of condensation [292] by splitting the deformable body according to its contact regions and rolling NNWarp back to a regular nonlinear solver to accurately simulate detailed denting effects, or to exhaustively sample the high-frequency external forces during the network training.

For the tongue visual system, we still don't have the optimal positions and number of EMA sensors for the inverse simulation. While putting quadratic domains according to the sensors' locations gives a satisfactory result in general, some local deformation is missed (highlighted in Fig. 6.10). Therefore, performing the domain partition in a way that better reflects the tongue's anatomy [293] may be a potential improvement. We will work closely with our collaborators and domain experts to find the answer to this fundamental question. We will also apply our system to other languages. Since our DNN-based method works well for the MOCHA database, it is expected that our system should perform well on the English language. Using an active model instead of a passive model to synthesize the articulation of the tongue is also an ambitious future work for us. Combing machine learning, physics-based modeling, and multi-modality data fusion seems to be a worthy idea.

Appendices

A Proving Eq. 2.28 **4**

B Elementary Complex Promotion **5**

Appendix A

Proving Eq. 2.28

Proof. We show that the subspace component deflection computed using Eq. 2.28 is identical to the solution of the full-space equilibrium (e.g. $\mathbf{Ku} = \mathbf{f}$). Noticing that for a loading component, external forces are only applied to the exciting DOFs and the corresponding component equilibrium becomes:

$$\begin{bmatrix} \mathbf{K}_{ee} & \mathbf{K}_{ep} & \mathbf{K}_{eb} \\ \mathbf{K}_{ep}{}^{\top} & \mathbf{K}_{pp} & \mathbf{K}_{pb} \\ \mathbf{K}_{eb}{}^{\top} & \mathbf{K}_{pb}{}^{\top} & \mathbf{K}_{bb} \end{bmatrix} \begin{bmatrix} \mathbf{u}_e \\ \mathbf{u}_p \\ \mathbf{u}_b \end{bmatrix} = \begin{bmatrix} \mathbf{f}_e \\ \mathbf{0}_p \\ \mathbf{f}_b \end{bmatrix}. \tag{A.1}$$

Expanding the second line of Eq. A.1 yields:

$$\mathbf{u}_p = -\mathbf{K}_{pp}^{-1}\mathbf{K}_{ep}^{\top}\mathbf{u}_e - \mathbf{K}_{pp}^{-1}\mathbf{K}_{pb}\mathbf{u}_b. \tag{A.2}$$

Similarly, we also expand the equilibrium of the passive DOFs through the definition of constraint mode and residual mode (e.g. Eqs. 2.14 and 2.25), which gives:

$$\mathbf{K}_{ep}^{\top}\Phi_e + \mathbf{K}_{pp}\Phi_p + \mathbf{K}_{pb}\mathbf{I}_b = \mathbf{0}, \tag{A.3}$$

and

$$\mathbf{K}_{pp}\Psi_p + \mathbf{K}_{ep}^{\top} = \mathbf{0}. \tag{A.4}$$

By substituting Eqs. A.3 and A.4 into Eq. A.2, we obtain:

$$\mathbf{u}_p = \Psi_p \mathbf{u}_e + (\Phi_p - \Psi_p \Phi_e)\, \mathbf{u}_b,$$

which leads to:

$$\begin{bmatrix} \mathbf{u}_e \\ \mathbf{u}_p \\ \mathbf{u}_b \end{bmatrix} = \begin{bmatrix} \Phi_e & \Psi_e \\ \Phi_p & \Psi_p \\ \mathbf{I}_b & \Psi_b \end{bmatrix} \begin{bmatrix} \mathbf{u}_b \\ \mathbf{u}_e - \Phi_e \mathbf{u}_b \end{bmatrix} = [\Phi | \Psi] \begin{bmatrix} \mathbf{q} \\ \mathbf{p} \end{bmatrix}.$$

□

Appendix B

Elementary Complex Promotion

The addition/subtraction and multiplication are trivial:

$$\begin{aligned} f(x_0) = x_0 \pm a &\rightarrow f^*(x_0 + hi) = x_0 \pm a + hi, \\ f(x_0) = s \cdot x_0 &\rightarrow f^*(x_0 + hi) = sx_0 + shi. \end{aligned} \tag{B.1}$$

The division is treated as the multiplication of the conjugate:

$$f(x_0) = \frac{a}{x} \rightarrow f^*(x_0 + hi) = \frac{a}{r^2}(x_0 - hi),\ r = \sqrt{x_0^2 + h^2}. \tag{B.2}$$

If the exponent of the power function (x^a) is an integer i.e. $a = n \in \mathbb{Z}$, we can use the De Moivre's formula:

$$f(x_0) = x^n \rightarrow f^*(x_0 + hi) = r^n(\cos n\phi + \sin n\phi i), \tag{B.3}$$

where $r\cos\phi = x_0$ and $r\sin\phi = h$ is the polar form of $x_0 + hi$. On the other hand, $a = 1/m$ ($m \in \mathbb{Z}$) makes $f(x_0)$ an m-root function, and the promotion is:

$$f(x_0) = x_0^{\frac{1}{m}} \rightarrow f^*(x_0 + hi) = r^{\frac{1}{m}} \left(\cos\frac{\phi + 2\pi k}{m} + \sin\frac{\phi + 2\pi k}{m} i \right). \tag{B.4}$$

Here, k is an integer between 0 and $m - 1$. In more general cases, when $a \in \mathbb{Q}$ is a rational number such that $a = n/m$, the power function of x^a is split as $f(x) = y^n$ and $y = a^{1/m}$.

The exponential function is promoted based on Euler's formula:

$$f(x_0) = e^{x_0} \rightarrow f^*(x_0 + hi) = e^{x_0}(\cos h + \sin hi). \tag{B.5}$$

The logarithmic promotion is the inverse of the exponential map, which can be obtained as:

$$f(x_0) = \ln x_0 \rightarrow f^*(x_0 + hi) = \ln r + (\phi + 2\pi k)i,\ k \in \mathbb{Z}. \tag{B.6}$$

Trigonometric functions can also be defined with complex numbers. According to Euler's formula, we have $\sin\alpha = (e^{\alpha i} - e^{-\alpha i})/2i$. Substituting α with $x_0 + hi$ leads to the promotion of $\sin x$:

$$f(x_0) = \sin x_0 \rightarrow f^*(x_0 + hi) = \frac{e^h + e^{-h}}{2}\sin x_0 + \frac{e^h - e^{-h}}{2}\cos x_0 i. \tag{B.7}$$

Similarly, $\cos\alpha = (e^{\alpha i} + e^{-\alpha i})/2$ is promoted as:

$$f(x_0) = \cos x_0 \rightarrow f^*(x_0 + hi) = \frac{e^h + e^{-h}}{2}\cos x_0 - \frac{e^h - e^{-h}}{2}\sin x_0 i. \tag{B.8}$$

Note that the promotion of exponential and logarithmic functions of Eqs. (B.4) and (B.6) is not unique due to the periodicity. We can restrict the argument angle to $[0, 2\pi]$, that makes $k = 0$.

References

[1] Jernej Barbič and Yili Zhao. Real-time large-deformation substructuring. SIGGRAPH '11, pages 91:1–91:8, 2011.

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

[3] François Faure, Benjamin Gilles, Guillaume Bousquet, and Dinesh K. Pai. Sparse meshless models of complex deformable solids. SIGGRAPH '11, pages 73:1–73:10, 2011.

[4] Benjamin Gilles, Guillaume Bousquet, Francois Faure, and Dinesh K. Pai. Frame-based elastic models. *ACM Trans. Graph.*, 30(2):15:1–15:12, April 2011.

[5] Huamin Wang and Yin Yang. Descent methods for elastic body simulation on the gpu. *ACM Transactions on Graphics (TOG)*, 35(6):212, 2016.

[6] Geoffrey Irving, Joseph Teran, and Ronald Fedkiw. Invertible finite elements for robust simulation of large deformation. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 131–140. Eurographics Association, 2004.

[7] Breannan Smith, Fernando De Goes, and Theodore Kim. Stable neo-hookean flesh simulation. *ACM Transactions on Graphics (TOG)*, 37(2):12, 2018.

[8] Roy R Craig Jr. A review of time-domain and frequency-domain component mode synthesis method. 1985.

[9] Walter C Hurty. Dynamic analysis of structural systems using component modes. *AIAA journal*, 3(4):678–685, 1965.

[10] Alexander Pentland and John Williams. Good vibrations: Modal dynamics for graphics and animation. 1989.

[11] Jernej Barbič and Doug L James. Real-time subspace integration for st. venant-kirchhoff deformable models. *ACM transactions on graphics (TOG)*, 24(3):982–990, 2005.

[12] Min Gyu Choi and Hyeong-Seok Ko. Modal warping: Real-time simulation of large rotational deformation and manipulation. *IEEE Transactions on Visualization and Computer Graphics*, 11(1):91–101, 2005.

[13] LH You, XS Yang, Mariusz Pachulski, and Jian J Zhang. Boundary constrained swept surfaces for modelling and animation. In *Computer Graphics Forum*, volume 26, pages 313–322. Wiley Online Library, 2007.

[14] Alec Jacobson, Elif Tosun, Olga Sorkine, and Denis Zorin. Mixed finite elements for variational surface modeling. In *Computer graphics forum*, volume 29, pages 1565–1574. Wiley Online Library, 2010.

[15] Klaus-Jürgen Bathe. *Finite element procedures*. Klaus-Jurgen Bathe, 2006.

[16] Uri Shani and Dana H Ballard. Splines as embeddings for generalized cylinders. *Computer Vision, Graphics, and Image Processing*, 27(2):129–156, 1984.

[17] Fopke Klok. Two moving coordinate frames for sweeping along a 3d trajectory. *Computer Aided Geometric Design*, 3(3):217–229, 1986.

[18] Wenping Wang, Bert Jüttler, Dayue Zheng, and Yang Liu. Computation of rotation minimizing frames. *ACM Transactions on Graphics (TOG)*, 27(1):2, 2008.

[19] Tae-Ick Chang, Joo-Haeng Lee, Myung-Soo Kim, and Sung Je Hong. Direct manipulation of generalized cylinders based on b-spline motion. *The Visual Computer*, 14(5-6):228–239, 1998.

[20] John K Johnstone and James P Williams. A rational model of the surface swept by a curve. In *Computer Graphics Forum*, volume 14, pages 77–88. Wiley Online Library, 1995.

[21] Sabine Coquillart. A control-point-based sweeping technique. *IEEE Computer Graphics and Applications*, 7(11):36–45, 1987.

[22] Guo-ping Wang and Jia-guang Sun. Shape control of swept surface with profiles. *Computer-Aided Design*, 33(12):893–902, 2001.

[23] LH You, Jian Chang, XS Yang, and Jian J Zhang. Solid modelling based on sixth order partial differential equations. *Computer-Aided Design*, 43(6):720–729, 2011.

[24] Mario Botsch and Leif Kobbelt. An intuitive framework for real-time freeform modeling. *ACM Transactions on Graphics (TOG)*, 23(3):630–634, 2004.

[25] Hong Qin and Demetri Terzopoulos. D-nurbs: a physics-based framework for geometric design. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):85–96, 1996.

[26] Demetri Terzopoulos and Hong Qin. Dynamic nurbs with geometric constraints for interactive sculpting. *ACM Transactions on Graphics (TOG)*, 13(2):103–136, 1994.

[27] Kexiang Wang, Ying He, Xiaohu Guo, and Hong Qin. Spline thin-shell simulation of manifold surfaces. In *Computer Graphics International Conference*, pages 570–577. Springer, 2006.

[28] David Baraff and Andrew Witkin. Large steps in cloth simulation. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 43–54. ACM, 1998.

[29] Akash Garg, Eitan Grinspun, Max Wardetzky, and Denis Zorin. Cubic shells. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 91–98. Eurographics Association, 2007.

[30] Eitan Grinspun, Anil N Hirani, Mathieu Desbrun, and Peter Schröder. Discrete shells. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 62–67. Eurographics Association, 2003.

[31] Max Wardetzky, Miklós Bergou, David Harmon, Denis Zorin, and Eitan Grinspun. Discrete quadratic curvature energies. *Computer Aided Geometric Design*, 24(8-9):499–518, 2007.

[32] Xiaohu Guo, Xin Li, Yunfan Bao, Xianfeng Gu, and Hong Qin. Meshless thin-shell simulation based on global conformal parameterization. *IEEE transactions on visualization and computer graphics*, 12(3):375–385, 2006.

[33] Sebastian Martin, Peter Kaufmann, Mario Botsch, Eitan Grinspun, and Markus Gross. Unified simulation of elastic rods, shells, and solids. In *ACM Transactions on Graphics (TOG)*, volume 29, page 39. ACM, 2010.

[34] Martin Wicke, Denis Steinemann, and Markus Gross. Efficient animation of point-sampled thin shells. In *Computer Graphics Forum*, volume 24, pages 667–676. Citeseer, 2005.

[35] W Michael Lai, David H Rubin, Erhard Krempl, and David Rubin. *Introduction to continuum mechanics*. Butterworth-Heinemann, 2009.

[36] Nan Zhang, Huamin Qu, and Robert Sweet. Orientation-preserving rod elements for real-time thin-shell simulation. *IEEE transactions on visualization and computer graphics*, 17(6):822–835, 2010.

[37] Ondrej Stava, Juraj Vanek, Bedrich Benes, Nathan Carr, and Radomír Měch. Stress relief: improving structural strength of 3d printable objects. *ACM Transactions on Graphics (TOG)*, 31(4):48, 2012.

[38] Nobuyuki Umetani, Takeo Igarashi, and Niloy J Mitra. Guided exploration of physically valid shapes for furniture design. *ACM Trans. Graph.*, 31(4):86–1, 2012.

[39] Qingnan Zhou, Julian Panetta, and Denis Zorin. Worst-case structural analysis. *ACM Trans. Graph.*, 32(4):137–1, 2013.

[40] Stelian Coros, Bernhard Thomaszewski, Gioacchino Noris, Shinjiro Sueda, Moira Forberg, Robert W Sumner, Wojciech Matusik, and Bernd Bickel. Computational design of mechanical characters. *ACM Transactions on Graphics (TOG)*, 32(4):83, 2013.

[41] Lifeng Zhu, Weiwei Xu, John Snyder, Yang Liu, Guoping Wang, and Baining Guo. Motion-guided mechanical toy modeling. *ACM Trans. Graph.*, 31(6):127–1, 2012.

[42] Moritz Bächer, Bernd Bickel, Doug L James, and Hanspeter Pfister. Fabricating articulated characters from skinned meshes. *ACM Trans. Graph.*, 31(4):47–1, 2012.

[43] Jacques Calì, Dan A Calian, Cristina Amati, Rebecca Kleinberger, Anthony Steed, Jan Kautz, and Tim Weyrich. 3d-printing of non-assembly, articulated models. *ACM Transactions on Graphics (TOG)*, 31(6):130, 2012.

[44] Xiang Chen, Changxi Zheng, Weiwei Xu, and Kun Zhou. An asymptotic numerical method for inverse elastic shape design. *ACM Transactions on Graphics (TOG)*, 33(4):95, 2014.

[45] Nobuyuki Umetani, Danny M Kaufman, Takeo Igarashi, and Eitan Grinspun. Sensitive couture for interactive garment modeling and editing. *ACM Trans. Graph.*, 30(4):90, 2011.

[46] Fehmi Cirak, Michael J Scott, Erik K Antonsson, Michael Ortiz, and Peter Schröder. Integrated modeling, finite-element analysis, and engineering design for thin-shell structures using subdivision. *Computer-Aided Design*, 34(2):137–148, 2002.

[47] Seth Green, George Turkiyyah, and Duane Storti. Subdivision-based multilevel methods for large scale engineering simulation of thin shells. In *Proceedings of the seventh ACM symposium on Solid modeling and applications*, pages 265–272. ACM, 2002.

[48] Steven S. An, Theodore Kim, and Doug L. James. Optimizing cubature for efficient integration of subspace deformations. *ACM transactions on graphics*, 27 5:165, 2008.

[49] Jernej Barbic and Doug L. James. Real-time subspace integration for st. venant-kirchhoff deformable models. *ACM Trans. Graph.*, 24:982–990, 2005.

[50] Min Gyu Choi and Hyeong-Seok Ko. Modal warping: real-time simulation of large rotational deformation and manipulation. *IEEE Transactions on Visualization and Computer Graphics*, 11:91–101, 2004.

[51] Min Gyu Choi, Seung Yong Woo, and Hyeong-Seok Ko. Real-time simulation of thin shells. *Comput. Graph. Forum*, 26:349–354, 2007.

[52] Bernd Weissmuller. Domain decomposition methods in science and engineering. 2016.

[53] C. Farhat and François-Xavier Roux. A method of finite element tearing and interconnecting and its parallel solution algorithm. 1991.

[54] Olek C Zienkiewicz and Robert L Taylor. *The finite element method for solid and structural mechanics*. Elsevier, 2005.

[55] Yin Yang, Weiwei Xu, Xiaohu Guo, Kun Zhou, and Baining Guo. Boundary-aware multidomain subspace deformation. *IEEE transactions on visualization and computer graphics*, 19(10):1633–1645, 2013.

[56] Lloyd N Trefethen and David Bau III. *Numerical linear algebra*, volume 50. Siam, 1997.

[57] Gaël Guennebaud, Benoıt Jacob, Philip Avery, Abraham Bachrach, Sebastien Barthelemy, et al. Eigen v3, 2010.

[58] A. Pentland and J. Williams. Good vibrations: Modal dynamics for graphics and animation. SIGGRAPH '89, pages 215–222, 1989.

[59] Jernej Barbič and Doug L. James. Real-time subspace integration for st. venant-kirchhoff deformable models. SIGGRAPH '05, pages 982–990, 2005.

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

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

[62] Theodore Kim and Doug L. James. Physics-based character skinning using multi-domain subspace deformations. SCA '11, pages 63–72, 2011.

[63] Francisco González García, Teresa Paradinas, Narcís Coll, and Gustavo Patow. Cages:: A multilevel, multi-cage-based system for mesh deformation. *ACM Trans. Graph.*, 32(3):24:1–24:13, July 2013.

[64] Thomas W. Sederberg and Scott R. Parry. Free-form deformation of solid geometric models. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '86, pages 151–160, 1986.

[65] Sebastian Martin, Peter Kaufmann, Mario Botsch, Eitan Grinspun, and Markus Gross. Unified simulation of elastic rods, shells, and solids. *ACM Trans. Graph.*, 29(4):39:1–39:10, July 2010.

[66] Steve Capell, Seth Green, Brian Curless, Tom Duchamp, and Zoran Popović. A multiresolution framework for dynamic deformations. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '02, pages 41–47. ACM, 2002.

[67] Eitan Grinspun, Petr Krysl, and Peter Schröder. Charms: A simple framework for adaptive simulation. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, pages 281–290. ACM, 2002.

[68] Xiaohuan Corina Wang and Cary Phillips. Multi-weight enveloping: Least-squares approximation techniques for skin animation. SCA '02, pages 129–138, 2002.

[69] Andrew Nealen, Matthias Müller, Richard Keiser, Eddy Boxerman, and Mark Carlson. Physically based deformable models in computer graphics. In *Computer Graphics Forum*, volume 25, pages 809–836, 2006.

[70] Eftychios Sifakis and Jernej Barbic. Fem simulation of 3d deformable solids: A practitioner's guide to theory, discretization and model reduction. In *ACM SIGGRAPH 2012 Courses*, SIGGRAPH '12, pages 20:1–20:50, 2012.

[71] 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.

[72] Rasmus Tamstorf, Toby Jones, and Stephen F. McCormick. Smoothed aggregation multigrid for cloth simulation. *ACM Trans. Graph.*, 34(6):245:1–245:13, October 2015.

[73] Florian Hecht, Yeon Jin Lee, Jonathan R. Shewchuk, and James F. O'Brien. Updated sparse cholesky factors for corotational elastodynamics. *ACM Transactions on Graphics*, 31(5):123:1–13, October 2012.

[74] Marco Fratarcangeli, Valentina Tibaldo, and Fabio Pellacini. Vivace: a practical gauss-seidel method for stable soft body dynamics. *ACM Transactions on Graphics (TOG)*, 35(6):214, 2016.

[75] Kris K. Hauser, Chen Shen, and James F. O'Brien. Interactive deformation using modal analysis with constraints. In *Graphics Interface*, pages 247–256, June 2003.

[76] Min Gyu Choi and Hyeong-Seok Ko. Modal warping: Real-time simulation of large rotational deformation and manipulation. *IEEE Transactions on Visualization and Computer Graphics*, 11(1):91–101, January 2005.

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

[78] Hongyi Xu and Jernej Barbič. Pose-space subspace dynamics. *ACM Transactions on Graphics (TOG)*, 35(4):35, 2016.

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

[80] Yun Teng, Mark Meyer, Tony DeRose, and Theodore Kim. Subspace condensation: Full space adaptivity for subspace deformations. *ACM Trans. Graph.*, 34(4):76:1–76:9, July 2015.

[81] Johannes Mezger, Bernhard Thomaszewski, Simon Pabst, and Wolfgang Straber. Interactive physically-based shape editing. *Computer Aided Geometric Design*, 26(6):680 – 694, 2009.

[82] Adam W. Bargteil and Elaine Cohen. Animation of deformable bodies with quadratic bÉzier finite elements. *ACM Trans. Graph.*, 33(3):27:1–27:10, June 2014.

[83] Matthias Müller, Bruno Heidelberger, Matthias Teschner, and Markus Gross. Meshless deformations based on shape matching. *ACM transactions on graphics (TOG)*, 24(3):471–478, 2005.

[84] Alec Jacobson, Zhigang Deng, Ladislav Kavan, and J. P. Lewis. Skinning: Real-time shape deformation (full text not available). In *ACM SIGGRAPH 2014 Courses*, SIGGRAPH '14, pages 24:1–24:1, 2014.

[85] Pushkar Joshi, Mark Meyer, Tony DeRose, Brian Green, and Tom Sanocki. Harmonic coordinates for character articulation. *ACM Trans. Graph.*, 26(3), July 2007.

[86] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans. Reconstruction and representation of 3d objects with radial basis functions. SIGGRAPH '01, pages 67–76, 2001.

[87] Michael S. Floater. Mean value coordinates. *Comput. Aided Geom. Des.*, 20(1):19–27, March 2003.

[88] Tao Ju, Scott Schaefer, and Joe Warren. Mean value coordinates for closed triangular meshes. *ACM Trans. Graph.*, 24(3):561–566, July 2005.

[89] Matthieu Nesme, Paul G. Kry, Lenka Jeřábková, and François Faure. Preserving topology and elasticity for embedded deformable models. SIGGRAPH '09, pages 52:1–52:9, 2009.

[90] Yu Wang, Alec Jacobson, Jernej Barbič, and Ladislav Kavan. Linear subspace design for real-time shape deformation. *ACM Transactions on Graphics (TOG)*, 34(4):57, 2015.

[91] G. Irving, J. Teran, and R. Fedkiw. Invertible finite elements for robust simulation of large deformation. SCA '04, pages 131–140, 2004.

[92] Ariel Shamir. A survey on mesh segmentation techniques. In *Computer graphics forum*, volume 27, pages 1539–1556. Wiley Online Library, 2008.

[93] S. Lloyd. Least squares quantization in pcm. *IEEE Trans. Inf. Theor.*, 28(2):129–137, September 1982.

[94] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[95] Steven S. An, Theodore Kim, and Doug L. James. Optimizing cubature for efficient integration of subspace deformations. SIGGRAPH Asia '08, pages 165:1–165:10, 2008.

[96] Maxime Tournier, Matthieu Nesme, François Faure, and Benjamin Gilles. Velocity-based adaptivity of deformable models. *Computers & Graphics*, 45:75–85, 2014.

[97] Alexey Stomakhin, Russell Howes, Craig Schroeder, and Joseph M. Teran. Energetically consistent invertible elasticity. SCA '12, pages 25–32, 2012.

[98] Funshing Sin, Yufeng Zhu, Yongqiang Li, and Daniel Schroeder. Invertible isotropic hyperelasticity using svd gradients. In *SCA'11 (Posters*, 2011.

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

[100] Sebastian Martin, Bernhard Thomaszewski, Eitan Grinspun, and Markus Gross. Example-based elastic materials. *ACM Trans. Graph.*, 30(4):72:1–72:8, July 2011.

[101] Alec Jacobson, Ilya Baran, Jovan Popovic, and Olga Sorkine. Bounded biharmonic weights for real-time deformation. *ACM Trans. Graph.*, 30(4):78–1, 2011.

[102] William Squire and George Trapp. Using complex variables to estimate derivatives of real functions. *SIAM review*, 40(1):110–112, 1998.

[103] Joaquim RRA Martins, Peter Sturdza, and Juan J Alonso. The complex-step derivative approximation. *ACM Transactions on Mathematical Software (TOMS)*, 29(3):245–262, 2003.

[104] Rafael Abreu, Zeming Su, Jochen Kamm, and Jinghuai Gao. On the accuracy of the complex-step-finite-difference method. *Journal of Computational and Applied Mathematics*, 340:390–403, 2018.

[105] JN Lyness. Differentiation formulas for analytic functions. *Mathematics of Computation*, pages 352–362, 1968.

[106] Griffith Baley Price. *An introduction to multicomplex spaces and functions*. M. Dekker, 1991.

[107] N Fleury, M Rausch Detraubenberg, and RM Yamaleev. Commutative extended complex numbers and connected trigonometry. *Journal of mathematical analysis and applications*, 180(2):431–457, 1993.

[108] Lars V Ahlfors. Complex analysis. 1979, 1973.

[109] Andrew Witkin. Physically based modeling: Principles and practice particle system dynamics. *SIGGRAPH Course notes*, 1997.

[110] David Baraff. Analytical methods for dynamic simulation of non-penetrating rigid bodies. In *ACM SIGGRAPH Computer Graphics*, volume 23, pages 223–232. ACM, 1989.

[111] David Baraff. Coping with friction for non-penetrating rigid body simulation. *ACM SIGGRAPH computer graphics*, 25(4):31–41, 1991.

[112] Robert Bridson. *Fluid simulation for computer graphics*. AK Peters/CRC Press, 2015.

[113] Rony Goldenthal, David Harmon, Raanan Fattal, Michel Bercovier, and Eitan Grinspun. Efficient simulation of inextensible cloth. In *ACM Transactions on Graphics (TOG)*, volume 26, page 49. ACM, 2007.

[114] Guowei Yan, Wei Li, Ruigang Yang, and Huamin Wang. Inexact descent methods for elastic parameter optimization. In *SIGGRAPH Asia 2018 Technical Papers*, page 253. ACM, 2018.

[115] Adriana Schulz, Jie Xu, Bo Zhu, Changxi Zheng, Eitan Grinspun, and Wojciech Matusik. Interactive design space exploration and optimization for cad models. *ACM Transactions on Graphics (TOG)*, 36(4):157, 2017.

[116] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically deformable models. *ACM Siggraph Computer Graphics*, 21(4):205–214, 1987.

[117] Javier Bonet and Richard D Wood. *Nonlinear continuum mechanics for finite element analysis*. Cambridge university press, 1997.

[118] Tiantian Liu, Adam W Bargteil, James F O'Brien, and Ladislav Kavan. Fast simulation of mass-spring systems. *ACM Transactions on Graphics (TOG)*, 32(6):214, 2013.

[119] Ari Stern and Mathieu Desbrun. Discrete geometric mechanics for variational time integrators. In *ACM SIGGRAPH 2006 Courses*, pages 75–80. ACM, 2006.

[120] Steve Capell, Seth Green, Brian Curless, Tom Duchamp, and Zoran Popović. Interactive skeleton-driven dynamic deformations. In *ACM transactions on graphics (TOG)*, volume 21, pages 586–593. ACM, 2002.

[121] Tiantian Liu, Sofien Bouaziz, and Ladislav Kavan. Quasi-newton methods for real-time simulation of hyperelastic materials. *ACM Transactions on Graphics (TOG)*, 36(4):116a, 2017.

[122] Eftychios Sifakis and Jernej Barbic. Fem simulation of 3d deformable solids: a practitioner's guide to theory, discretization and model reduction. In *ACM SIGGRAPH 2012 Courses*, page 20. ACM, 2012.

[123] Sebastian Martin, Bernhard Thomaszewski, Eitan Grinspun, and Markus Gross. Example-based elastic materials. In *ACM Transactions on Graphics (TOG)*, volume 30, page 72. ACM, 2011.

[124] Yuki Koyama, Kenshi Takayama, Nobuyuki Umetani, and Takeo Igarashi. Real-time example-based elastic deformation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 19–24. Eurographics Association, 2012.

[125] Hongyi Xu, Funshing Sin, Yufeng Zhu, and Jernej Barbič. Nonlinear material design using principal stretches. *ACM Transactions on Graphics (TOG)*, 34(4):75, 2015.

[126] Stephen Wolfram et al. *Mathematica*. Cambridge university press Cambridge, 1996.

[127] V Maple. Waterloo maple software. *University of Waterloo, Version*, 5, 1994.

[128] Fabian Hahn, Sebastian Martin, Bernhard Thomaszewski, Robert Sumner, Stelian Coros, and Markus Gross. Rig-space physics. *ACM transactions on graphics (TOG)*, 31(4):72, 2012.

[129] Jernej Barbič, Funshing Sin, and Eitan Grinspun. Interactive editing of deformable simulations. *ACM Transactions on Graphics (TOG)*, 31(4):70, 2012.

[130] Fabian Hahn, Bernhard Thomaszewski, Stelian Coros, Robert W Sumner, and Markus Gross. Efficient simulation of secondary motion in rig-space. In *Proceedings of the 12th ACM SIGGRAPH/eurographics symposium on computer animation*, pages 165–171. ACM, 2013.

[131] Yin Yang, Dingzeyu Li, Weiwei Xu, Yuan Tian, and Changxi Zheng. Expediting precomputation for reduced deformable simulation. *ACM Transactions on graphics (TOG)*, 34(6), 2015.

[132] Michael Renardy and Robert C Rogers. *An introduction to partial differential equations*, volume 13. Springer Science & Business Media, 2006.

[133] Patrick Brezillon, Jean-François Staub, Anne-Marie Perault-Staub, and Gérard Milhaud. Numerical estimation of the first order derivative: approximate evaluation of an optimal step. *Computers & Mathematics with Applications*, 7(4):333–347, 1981.

[134] James N Lyness. Numerical algorithms based on the theory of complex variable. In *Proceedings of the 1967 22nd national conference*, pages 125–133. ACM, 1967.

[135] W Kyle Anderson, James C Newman, David L Whitfield, and Eric J Nielsen. Sensitivity analysis for navier-stokes equations on unstructured meshes using complex variables. *AIAA journal*, 39(1):56–63, 2001.

[136] N Butuk and J-P Pemba. computing chemkin sensitivities using complex variables. *Journal of engineering for gas turbines and power*, 125(3):854–858, 2003.

[137] Andrew Voorhees, Harry Millwater, and Ronald Bagley. Complex variable methods for shape sensitivity of finite element models. *Finite elements in analysis and design*, 47(10):1146–1156, 2011.

[138] Arturo Montoya, Randal Fielder, Armando Gomez-Farias, and Harry Millwater. Finite-element sensitivity for plasticity using complex variable methods. *Journal of Engineering Mechanics*, 141(2):04014118, 2014.

[139] Agustí Pérez-Foguet, Antonio Rodríguez-Ferran, and Antonio Huerta. Numerical differentiation for local and global tangent operators in computational plasticity. *Computer Methods in Applied Mechanics and Engineering*, 189(1):277–296, 2000.

[140] Sonia Lebofsky. *Numerically Generated Tangent Stiffness Matrices for Geometrically Non-Linear Structures*. PhD thesis, 2013.

[141] Sanghaun Kim, Junghyun Ryu, and Maenghyo Cho. Numerically generated tangent stiffness matrices using the complex variable derivative method for nonlinear structural analysis. *Computer Methods in Applied Mechanics and Engineering*, 200(1-4):403–413, 2011.

[142] K-L Lai and JL Crassidis. Extensions of the first and second complex-step derivative approximations. *Journal of Computational and Applied Mathematics*, 219(1):276–293, 2008.

[143] RL Bagley. On fourier differentiation – a numerical tool for implicit functions. *International Journal of Applied Mathematics*, 19(3):255, 2006.

[144] Gregory Lantoine, Ryan P Russell, and Thierry Dargent. Using multicomplex variables for automatic computation of high-order derivatives. *ACM Transactions on Mathematical Software (TOMS)*, 38(3):16, 2012.

[145] Louis B Rall. Automatic differentiation: Techniques and applications. 1981.

[146] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[147] Andreas Griewank and Andrea Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*, volume 105. Siam, 2008.

[148] Don Mitchell and Pat Hanrahan. Illumination from curved reflectors. In *ACM SIGGRAPH Computer Graphics*, volume 26, pages 283–291. ACM, 1992.

[149] Brian Guenter. Efficient symbolic differentiation for graphics applications. In *ACM Transactions on Graphics (TOG)*, volume 26, page 108. ACM, 2007.

[150] Robert Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural networks for perception*, pages 65–93. Elsevier, 1992.

[151] Richard D Neidinger. Introduction to automatic differentiation and matlab object-oriented programming. *SIAM review*, 52(3):545–563, 2010.

[152] Atilim Gunes Baydin and Barak A Pearlmutter. Automatic differentiation of algorithms for machine learning. *arXiv preprint arXiv:1404.7456*, 2014.

[153] Charles C Margossian. A review of automatic differentiation and its efficient implementation. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, page e1305, 2018.

[154] Robin J Hogan. Fast reverse-mode automatic differentiation using expression templates in c++. *ACM Transactions on Mathematical Software (TOMS)*, 40(4):26, 2014.

[155] Michael Betancourt. A geometric theory of higher-order automatic differentiation. *arXiv preprint arXiv:1812.11592*, 2018.

[156] Christoph W Ueberhuber. *Numerical computation 1: methods, software, and analysis*. Springer Science & Business Media, 2012.

[157] IEEE. Ieee standard for binary floating-point arithmetic. Institute of Electrical and Electronic Engineers, 1985.

[158] Jeffrey Fike and Juan Alonso. The development of hyper-dual numbers for exact second-derivative calculations. In *49th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, page 886, 2011.

[159] Jeffrey Alan Fike. *Multi-objective optimization using hyper-dual numbers*. PhD thesis, Stanford university, 2013.

[160] Edna E Kramer. Polygenic functions of the dual variable w= u+ jv. *American Journal of Mathematics*, 52(2):370–376, 1930.

[161] Mark J Ablowitz and Athanassios S Fokas. *Complex variables: introduction and applications*. Cambridge University Press, 2003.

[162] Boris Schäling. *The boost C++ libraries*. Boris Schäling, 2011.

[163] Gael Guennebaud, Benoit Jacob, et al. Eigen: a c++ linear algebra library. *URL http://eigen. tuxfamily. org, Accessed*, 22, 2014.

[164] Bradley M Bell. Cppad: a package for c++ algorithmic differentiation. *Computational Infrastructure for Operations Research*, 57:10, 2012.

[165] HM Nasir. A new class of multicomplex algebra with applications. *Mathematical Sciences International Research Journal*, 2(2):163–168, 2013.

[166] Ken Shoemake. Animating rotation with quaternion curves. In *ACM SIGGRAPH computer graphics*, volume 19, pages 245–254. ACM, 1985.

[167] Andreas Griewank, David Juedes, and Jean Utke. Algorithm 755: Adol-c: a package for the automatic differentiation of algorithms written in c/c++. *ACM Transactions on Mathematical Software (TOMS)*, 22(2):131–167, 1996.

[168] Abraham Lee. ad: Fast, transparent first- and second-order automatic differentiation, 2013.

[169] Jernej Barbič, Fun Shing Sin, and Daniel Schroeder. Vega fem library, 2012.

[170] Klaus-Jürgen Bathe. *Finite element method*. Wiley Online Library, 2008.

[171] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[172] Jui-Ting Huang, Jinyu Li, Dong Yu, Li Deng, and Yifan Gong. Cross-language knowledge transfer using multilingual deep neural network with shared hidden layers. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 7304–7308. IEEE, 2013.

[173] George E Dahl, Dong Yu, Li Deng, and Alex Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on audio, speech, and language processing*, 20(1):30–42, 2012.

[174] Jia Pan, Cong Liu, Zhiguo Wang, Yu Hu, and Hui Jiang. Investigation of deep neural networks (dnn) for large vocabulary continuous speech recognition: Why dnn surpasses gmms in acoustic modeling. In *Chinese Spoken Language Processing (ISCSLP), 2012 8th International Symposium on*, pages 301–305. IEEE, 2012.

[175] Lijun Wang, Wanli Ouyang, Xiaogang Wang, and Huchuan Lu. Visual tracking with fully convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3119–3127, 2015.

[176] Matej Kristan, Jiri Matas, Ales Leonardis, Michael Felsberg, Luka Cehovin, Gustavo Fernández, Tomas Vojir, Gustav Hager, Georg Nebehay, and Roman Pflugfelder. The visual object tracking vot2015 challenge results. In *Proceedings of the IEEE international conference on computer vision workshops*, pages 1–23, 2015.

[177] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[178] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, 1989.

[179] Matthias Müller, Julie Dorsey, Leonard McMillan, Robert Jagnow, and Barbara Cutler. Stable real-time deformations. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 49–54. ACM, 2002.

[180] Jernej Barbič and Yili Zhao. Real-time large-deformation substructuring. In *ACM transactions on graphics (TOG)*, volume 30, page 91. ACM, 2011.

[181] Rina Dechter. *Learning while searching in constraint-satisfaction problems*. University of California, Computer Science Department, Cognitive Systems Laboratory, 1986.

[182] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 806–813, 2014.

[183] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[184] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[185] Clement Farabet, Camille Couprie, Laurent Najman, and Yann LeCun. Learning hierarchical features for scene labeling. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1915–1929, 2013.

[186] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

[187] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[188] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *arXiv preprint arXiv:1606.00915*, 2016.

[189] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1520–1528, 2015.

[190] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Region-based convolutional networks for accurate object detection and segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 38(1):142–158, 2016.

[191] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[192] Miguel A Otaduy, Bernd Bickel, Derek Bradley, and Huamin Wang. Data-driven simulation methods in computer graphics: cloth, tissue and faces. In *ACM SIGGRAPH 2012 Courses*, page 12. ACM, 2012.

[193] Huamin Wang, Florian Hecht, Ravi Ramamoorthi, and James F O'Brien. Example-based wrinkle synthesis for clothing animation. In *ACM Transactions on Graphics (TOG)*, volume 29, page 107. ACM, 2010.

[194] Ladislav Kavan, Dan Gerszewski, Adam W Bargteil, and Peter-Pike Sloan. Physics-inspired upsampling for cloth simulation in games. In *ACM Transactions on Graphics (TOG)*, volume 30, page 93. ACM, 2011.

[195] Huamin Wang, James F O'Brien, and Ravi Ramamoorthi. Data-driven elastic models for cloth: modeling and measurement. In *ACM Transactions on Graphics (TOG)*, volume 30, page 71. ACM, 2011.

[196] Eder Miguel, Derek Bradley, Bernhard Thomaszewski, Bernd Bickel, Wojciech Matusik, Miguel A Otaduy, and Steve Marschner. Data-driven estimation of cloth simulation models. In *Computer Graphics Forum*, volume 31, pages 519–528. Wiley Online Library, 2012.

[197] Doyub Kim, Woojong Koh, Rahul Narain, Kayvon Fatahalian, Adrien Treuille, and James F O'Brien. Near-exhaustive precomputation of secondary cloth effects. *ACM Transactions on Graphics (TOG)*, 32(4):87, 2013.

[198] Weiwei Xu, Nobuyuki Umetani, Qianwen Chao, Jie Mao, Xiaogang Jin, and Xin Tong. Sensitivity-optimized rigging for example-based real-time clothing synthesis. *ACM Trans. Graph.*, 33(4):107–1, 2014.

[199] Stelian Coros, Philippe Beaudoin, and Michiel Van de Panne. Robust task-based control policies for physics-based characters. In *ACM Transactions on Graphics (TOG)*, volume 28, page 170. ACM, 2009.

[200] Yongjoon Lee, Kevin Wampler, Gilbert Bernstein, Jovan Popović, and Zoran Popović. Motion fields for interactive character locomotion. In *ACM Transactions on Graphics (TOG)*, volume 29, page 138. ACM, 2010.

[201] Xue Bin Peng, Glen Berseth, and Michiel Van de Panne. Dynamic terrain traversal skills using reinforcement learning. *ACM Transactions on Graphics (TOG)*, 34(4):80, 2015.

[202] Libin Liu, Michiel Van De Panne, and KangKang Yin. Guided learning of control graphs for physics-based characters. *ACM Transactions on Graphics (TOG)*, 35(3):29, 2016.

[203] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[204] Libin Liu and Jessica Hodgins. Learning to schedule control fragments for physics-based characters using deep q-learning. *ACM Transactions on Graphics (TOG)*, 36(3):29, 2017.

[205] Xue Bin Peng, Glen Berseth, KangKang Yin, and Michiel Van De Panne. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 36(4):41, 2017.

[206] Daniel Holden, Taku Komura, and Jun Saito. Phase-functioned neural networks for character control. *ACM Transactions on Graphics (TOG)*, 36(4):42, 2017.

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

[208] Hongyi Xu and Jernej Barbič. Example-based damping design. *ACM Trans. Graph.*, 36(4):53:1–53:14, July 2017.

[209] Meekyoung Kim, Gerard Pons-Moll, Sergi Pujades, Seungbae Bang, Jinwook Kim, Michael J Black, and Sung-Hee Lee. Data-driven physics for human soft tissue animation. *ACM Transactions on Graphics (TOG)*, 36(4):54, 2017.

[210] Ben Jones, Nils Thuerey, Tamar Shinar, and Adam W Bargteil. Example-based plastic deformation of rigid bodies. *ACM Transactions on Graphics (TOG)*, 35(4):34, 2016.

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

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

[213] 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.

[214] Mark Pauly, Richard Keiser, Bart Adams, Philip Dutré, Markus Gross, and Leonidas J Guibas. Meshless animation of fracturing solids. *ACM Transactions on Graphics (TOG)*, 24(3):957–964, 2005.

[215] Mathieu Desbrun, Peter Schröder, and Alan Barr. Interactive animation of structured deformable objects. In *Graphics Interface*, volume 99, page 10, 1999.

[216] Huamin Wang, James O'Brien, and Ravi Ramamoorthi. Multi-resolution isotropic strain limiting. In *ACM Transactions on Graphics (TOG)*, volume 29, page 156, 2010.

[217] Florian Hecht, Yeon Jin Lee, Jonathan R Shewchuk, and James F O'Brien. Updated sparse cholesky factors for corotational elastodynamics. *ACM Transactions on Graphics (TOG)*, 31(5):123, 2012.

[218] Yongning Zhu, Eftychios Sifakis, Joseph Teran, and Achi Brandt. An efficient multigrid method for the simulation of high-resolution elastic solids. *ACM Transactions on Graphics (TOG)*, 29(2):16, 2010.

[219] Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly. Projective dynamics: fusing constraint projections for fast simulation. *ACM Transactions on Graphics (TOG)*, 33(4):154, 2014.

[220] Rahul Narain, Matthew Overby, and George E. Brown. Admm $\supseteq$ projective dynamics: Fast simulation of general constitutive models. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '16, pages 21–28, 2016.

[221] Huamin Wang. A chebyshev semi-iterative approach for accelerating projective and position-based dynamics. *ACM Transactions on Graphics (TOG)*, 34(6):246, 2015.

[222] Theodore Kim and Doug L James. Skipping steps in deformable simulation with online model reduction. In *ACM transactions on graphics (TOG)*, volume 28, page 123, 2009.

[223] Jin Huang, Yiying Tong, Kun Zhou, Hujun Bao, and Mathieu Desbrun. Interactive shape interpolation through controllable dynamic deformation. *IEEE Transactions on Visualization and Computer Graphics*, 17(7):983–992, 2011.

[224] Zherong Pan, Hujun Bao, and Jin Huang. Subspace dynamic simulation using rotation-strain coordinates. *ACM Transactions on Graphics (TOG)*, 34(6):242, 2015.

[225] Siwang Li, Jin Huang, Fernando de Goes, Xiaogang Jin, Hujun Bao, and Mathieu Desbrun. Space-time editing of elastic motion through material optimization and reduction. *ACM Transactions on Graphics (TOG)*, 33(4):108, 2014.

[226] Philip Wolfe. Convergence conditions for ascent methods. *SIAM review*, 11(2):226–235, 1969.

[227] Huanhuan Xu, Wuyi Yu, Shiyuan Gu, and Xin Li. Biharmonic volumetric mapping using fundamental solutions. *IEEE Transactions on Visualization and Computer Graphics*, 19(5):787–798, 2013.

[228] Xiao-Ming Fu, Yang Liu, and Baining Guo. Computing locally injective mappings by advanced mips. *ACM Transactions on Graphics (TOG)*, 34(4):71, 2015.

[229] Noam Aigerman and Yaron Lipman. Injective and bounded distortion mappings in 3d. *ACM Transactions on Graphics (TOG)*, 32(4):106, 2013.

[230] Yili Zhao and Jernej Barbič. Interactive authoring of simulation-ready plants. *ACM Transactions on Graphics (TOG)*, 32(4):84, 2013.

[231] Bohan Wang, Yili Zhao, and Jernej Barbič. Botanical materials based on biomechanics. *ACM Transactions on Graphics (TOG)*, 36(4):135, 2017.

[232] Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun, and Xin Tong. O-cnn: Octree-based convolutional neural networks for 3d shape analysis. *ACM Trans. Graph.*, 36(4):72:1–72:11, July 2017.

[233] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1912–1920, 2015.

[234] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[235] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

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

[237] Demetri Terzopoulos and Andrew Witkin. Physically based models with rigid and deformable components. *IEEE Computer Graphics and Applications*, 8(6):41–51, 1988.

[238] Mahabalagiri N Hegde. *Introduction to communicative disorders*. Pro Ed, 1995.

[239] KJ GRENABO. Atlas of topographical and applied human anatomy. head and neck, 1965.

[240] Hironori Takemoto. Morphological analyses of the human tongue musculature for three-dimensional modeling. *Journal of Speech, Language, and Hearing Research*, 44(1):95–107, 2001.

[241] Ray D Kent. Research on speech motor control and its disorders: A review and prospective. *Journal of Communication disorders*, 33(5):391–428, 2000.

[242] Joseph Perkell, Melanie Matthies, Harlan Lane, Frank Guenther, Reiner Wilhelms-Tricarico, Jane Wozniak, and Peter Guiod. Speech motor control: Acoustic goals, saturation effects, auditory feedback and internal models. *Speech communication*, 22(2):227–250, 1997.

[243] Rob W Bisseling and At L Hof. Handling of impact forces in inverse dynamics. *Journal of biomechanics*, 39(13):2438–2444, 2006.

[244] Makoto Hirayama, Eric Vatikiotis-Bateson, and Mitsuo Kawato. Inverse dynamics of speech motor control. In *Advances in neural information processing systems*, pages 1043–1050, 1994.

[245] Ian Stavness, John Lloyd, and Sidney Fels. Inverse-dynamics simulation of muscular-hydrostat finite-element models. In *23rd International Society of Biomechanics Congress (ISB)*, volume 933, 2011.

[246] Shin Suzuki, Takeshi Okadome, and Masaaki Honda. Determination of articulatory positions from speech acoustics by applying dynamic articulatory constraints. In *ICSLP*, 1998.

[247] Yin Yang, Xiaohu Guo, Jennell Vick, Luis G Torres, and Thomas F Campbell. Physics-based deformable tongue visualization. *IEEE transactions on visualization and computer graphics*, 19(5):811–823, 2013.

[248] Pierre Badin, Gerard Bailly, Lionel Reveret, Monica Baciu, Christoph Segebarth, and Christophe Savariaux. Three-dimensional linear articulatory modeling of tongue, lips and face, based on mri and video images. *Journal of Phonetics*, 30(3):533–553, 2002.

[249] T Baer, JC Gore, S Boyce, and PW Nye. Application of mri to the analysis of speech production. *Magnetic resonance imaging*, 5(1):1–7, 1987.

[250] Maureen Stone, Edward P Davis, Andrew S Douglas, Moriel Ness Aiver, Rao Gullapalli, William S Levine, and Andrew Jon Lundberg. Modeling tongue surface contours from cine-mri images. *Journal of speech, language, and hearing research*, 44(5):1026–1040, 2001.

[251] Sajan Goud Lingala, Brad P Sutton, Marc E Miquel, and Krishna S Nayak. Recommendations for real-time speech mri. *Journal of Magnetic Resonance Imaging*, 43(1):28–44, 2016.

[252] Maojing Fu, Marissa S Barlaz, Joseph L Holtrop, Jamie L Perry, David P Kuehn, Ryan K Shosted, Zhi-Pei Liang, and Bradley P Sutton. High-frame-rate full-vocal-tract 3d dynamic speech imaging. *Magnetic resonance in medicine*, 2016.

[253] Peter W Iltis, Jens Frahm, Dirk Voit, Arun A Joseph, Erwin Schoonderwaldt, and Eckart Altenmüller. High-speed real-time magnetic resonance imaging of fast tongue movements in elite horn players. *Quantitative imaging in medicine and surgery*, 5(3):374–381, 2015.

[254] Sajan Goud Lingala, Yinghua Zhu, Yoon-Chul Kim, Asterios Toutios, Shrikanth Narayanan, and Krishna S Nayak. A fast and flexible mri system for the study of dynamic vocal tract shaping. *Magnetic resonance in medicine*, 2016.

[255] Maureen Stone. A three-dimensional model of tongue movement based on ultrasound and x-ray microbeam data. *The Journal of the Acoustical Society of America*, 87(5):2207–2217, 1990.

[256] Thomas H Shawker, Barbara Sonies, Maureen Stone, and Bruce J Baum. Real-time ultrasound visualization of tongue movement during swallowing. *Journal of Clinical Ultrasound*, 11(9):485–490, 1983.

[257] Barbara C Sonies, Thomas H Shawker, Thomas E Hall, Lynn H Gerber, and Stephen B Leighton. Ultrasonic visualization of tongue motion during speech. *The Journal of the Acoustical Society of America*, 70(3):683–686, 1981.

[258] Scott A King and Richard E Parent. A 3d parametric tongue model for animated speech. *The Journal of Visualization and Computer Animation*, 12(3):107–115, 2001.

[259] Andrew J Lundberg and Maureen Stone. Three-dimensional tongue surface reconstruction: Practical considerations for ultrasound data. *The Journal of the Acoustical Society of America*, 106(5):2858–2867, 1999.

[260] Maureen Stone and Andrew Lundberg. Three-dimensional tongue surface shapes of english consonants and vowels. *The Journal of the Acoustical Society of America*, 99(6):3728–3737, 1996.

[261] Yusuf Sinan Akgul, Chandra Kambhamettu, and Maureen Stone. Automatic extraction and tracking of the tongue contours. *IEEE Transactions on Medical Imaging*, 18(10):1035–1045, 1999.

[262] Thomas Hueber, Guido Aversano, G Cholle, Bruce Denby, Gérard Dreyfus, Yacine Oussar, Pierre Roussel, and Maureen Stone. Eigentongue feature extraction for an ultrasound-based silent speech interface. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07*, volume 1, pages I–1245. IEEE, 2007.

[263] Min Li, Chandra Kambhamettu, and Maureen Stone. Automatic contour tracking in ultrasound images. *Clinical linguistics & phonetics*, 19(6-7):545–554, 2005.

[264] Bishnu S Atal, Jih Jie Chang, Max V Mathews, and John W Tukey. Inversion of articulatory-to-acoustic transformation in the vocal tract by a computer-sorting technique. *The Journal of the Acoustical Society of America*, 63(5):1535–1555, 1978.

[265] Slim Ouni and Yves Laprie. Modeling the articulatory space using a hypercube codebook for acoustic-to-articulatory inversion. *The Journal of the Acoustical Society of America*, 118(1):444–460, 2005.

[266] Sadao Hiroya and Masaaki Honda. Estimation of articulatory movements from speech acoustics using an hmm-based speech production model. *IEEE Transactions on Speech and Audio Processing*, 12(2):175–185, 2004.

[267] Le Zhang and Steve Renals. Acoustic-articulatory modeling with the trajectory hmm. *IEEE Signal Processing Letters*, 15:245–248, 2008.

[268] Tomoki Toda, Alan W Black, and Keiichi Tokuda. Statistical mapping between articulatory movements and acoustic spectrum using a gaussian mixture model. *Speech Communication*, 50(3):215–227, 2008.

[269] Korin Richmond. *Estimating articulatory parameters from the acoustic speech signal*. PhD thesis, University of Edinburgh, 2002.

[270] Benigno Uria, Iain Murray, Steve Renals, and Korin Richmond. Deep architectures for articulatory inversion. In *INTERSPEECH*, pages 867–870, 2012.

[271] Zhiyong Wu, Kai Zhao, Xixin Wu, Xinyu Lan, and Helen Meng. Acoustic to articulatory mapping with deep neural network. *Multimedia Tools and Applications*, 74(22):9889–9907, 2015.

[272] Stéphanie Buchaillard, Pascal Perrier, and Yohan Payan. A biomechanical model of cardinal vowel production: Muscle activations and the impact of gravity on tongue positioning. *The Journal of the Acoustical Society of America*, 126(4):2033–2051, 2009.

[273] Jean-Michel Gérard, Pascal Perrier, and Yohan Payan. 3d biomechanical tongue modeling to study speech production. *Speech production: Models, phonetic processes, and techniques*, pages 85–102, 2006.

[274] Ian Stavness, John Lloyd, Yohan Payan, and Sidney Fels. Towards speech articulation simulation with a dynamic coupled face-jaw-tongue model. In *ISSP'2011*, pages 1–4, 2011.

[275] Ian Stavness, John E Lloyd, and Sidney Fels. Automatic prediction of tongue muscle activations using a finite element model. *Journal of biomechanics*, 45(16):2841–2848, 2012.

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

[277] Hang Si. Tetgen, a delaunay-based quality tetrahedral mesh generator. *ACM Transactions on Mathematical Software*, 41(2):11, 2015.

[278] Jianwen Luo, Kui Ying, and Jing Bai. Savitzky–golay smoothing and differentiation filter for even number data. *Signal Processing*, 85(7):1429–1434, 2005.

[279] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

[280] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[281] Jean-Michel Gérard, Jacques Ohayon, Vincent Luboz, Pascal Perrier, and Yohan Payan. Indentation for estimating the human tongue soft tissues constitutive law: application to a 3d biomechanical model. In *Medical Simulation*, pages 77–83. Springer, 2004.

[282] Florian Vogt, John E Lloyd, Stéphanie Buchaillard, Pascal Perrier, Matthieu Chabanas, Yohan Payan, and Sidney S Fels. Efficient 3d finite element modeling of a muscle-activated tongue. In *International Symposium on Biomedical Simulation*, pages 19–28. Springer, 2006.

[283] Geoffrey Irving, Craig Schroeder, and Ronald Fedkiw. Volume conserving finite element simulations of deformable models. *ACM Trans. Graph.*, 26(3), July 2007.

[284] J. Teran, S. Blemker, V. Ng Thow Hing, and R. Fedkiw. Finite volume methods for the simulation of skeletal muscle. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '03, pages 68–74, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

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

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

[287] Alan Wrench. The mocha-timit articulatory database, 1999.

[288] Korin Richmond. A trajectory mixture density network for the acoustic-articulatory inversion mapping. In *Interspeech*, 2006.

[289] Sharynne Mcleod. Speech–language pathologists' knowledge of tongue/palate contact for consonants. *Clinical linguistics & phonetics*, 25(11-12):1004–1013, 2011.

[290] Paolo Cignoni, Claudio Rocchini, and Roberto Scopigno. Metro: measuring error on simplified surfaces. In *Computer Graphics Forum*, volume 17, pages 167–174. Wiley Online Library, 1998.

[291] Matthias Müller, Bruno Heidelberger, Matthias Teschner, and Markus Gross. Meshless deformations based on shape matching. SIGGRAPH '05, pages 471–478, 2005.

[292] Yun Teng, Mark Meyer, Tony DeRose, and Theodore Kim. Subspace condensation: Full space adaptivity for subspace deformations. *ACM Transactions on Graphics (TOG)*, 34(4):76, 2015.

[293] Bruce P Halpern. Functional anatomy of the tongue and mouth of mammals. In *Drinking behavior*, pages 1–92. Springer, 1977.