PHYSICS-BASED DYNAMICS WITH APPLICATIONS TO FACIAL ANIMATION

by

Dimitar Dinev

A dissertation submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

School of Computing

The University of Utah

May 2020

The University of Utah Graduate School

STATEMENT OF DISSERTATION APPROVAL

The dissertation of **Dimitar Dinev**
has been approved by the following supervisory committee members:

Ladislav Kavan ,	Chair(s)	**TBD** Date Approved
Cem Yuksel ,	Member	**TBD** Date Approved
Hari Sundar ,	Member	**TBD** Date Approved
Eftychios Sifakis ,	Member	**TBD** Date Approved
Daniel Sýkora ,	Member	**TBD** Date Approved

by **Ross Whitaker** , Chair/Dean of
the Department/College/School of **School of Computing**
and by **David Kieta** , Dean of The Graduate School.

ABSTRACT

Human facial animation is an important part of computer graphics, with widespread use among the entertainment and medical industries alike. Most current methods for animating the human face are geometry-based techniques that work well with advanced capture systems, and then require a lot of manual labor on the part of digital artists to clean up any errors and inconsistencies. In this work, we begin by introducing a geometry-based method to help reduce the manual labor required to clean up inconsistencies. We then discuss physics-based approaches for human facial animation.

We begin the discussion by discussing the dynamics of passive tissue. Many of the methods used for simulating deformable solids were developed for scientific simulations and mechanical engineering, but these methods are not necessarily suitable for animation. Animators are usually not concerned about accuracy, instead prefer speed and the visual plausibility of the result. Many time integration techniques for advancing the states of the simulations are designed primarily for accuracy, but are prohibitively slow particularly for real-time applications. On the other extreme, integration techniques used in real-time applications such as backward Euler have many undesirable properties such as artificial numerical dissipation. We propose new integration methods that are stable, fast, and exhibit no artificial dissipation at the cost of numerical accuracy. This allows us to model the passive tissue in the face while preserving vivid motion under external forces.

Finally, we discuss the issue of simulating that active tissue in the face; the muscles. Building a good parameterization for the face is a very difficult problem, and we propose to find an anatomically-motivated model that can is closer to the real-world parameterization of the face. Using this new muscle model and combining it with the energy-conserving numerical integration techniques allows us to create a dynamic facial animation, which can make expressions while producing vivid motion under strong external forces, such as wind. We hope this dissertation provides useful tools to advance the field of anatomy-based human simulation.

I dedicate this thesis to my beloved parents, Dr. Petko Dinev and Dr. Tamara Dinev, whose support and encouragement helped me complete this journey.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Animation of the human face is an important part of computer graphics, especially as digital human models become more and more prevalent in both the entertainment and the medical industries. Achieving realistic facial animations and expressions is extremely important to the realism of these characters, yet remains an extremely difficult problem to solve. Partially, this is due to our enhanced perception of the subtleties of facial expressions [47], making even small inaccuracies noticeable and contributing to the "uncanny valley" effect. In the entertainment industry, facial tracking and capture technologies have become very advanced [12, 15, 14] and able to capture a large majority of the face, with some small yet important exceptions. One such area is the lips, specifically the inner lips. This area presents a unique challenge for capture technologies due to many factors: the high specularity caused by the wetness of the lips, the significant expression-based occlusion that occurs, and the highly varying lighting conditions.

Traditional techniques for correcting such issues are usually geometry-based: using the mesh vertices and elements to fix any inconsistencies. In this thesis, we will begin by discussion an example of one such technique. We begin by taking a sequence of meshes captured using state-of-the-art capture systems, and introduce a method to correct the lip shapes, restoring some of the "fleshiness" that was lost during all of the processing steps. This technique is a user-guided one: we first have a user fix a few example frames by editing the mesh. We then leverage the fact that the region around the lips is captured very accurately to reconstruct the inner lip region based on the user-provided input. By taking several such user-provided corrected examples, we are able to perform a regression and use this to propagate these changes to the rest of the performance. This results in a much improved lip shape throughout the entire performance using only several corrected frames. This work was published in Computer Graphics Forum [42].

Such geometric techniques can be very useful and can result in much-improved animations. However, geometry-based techniques have many problems, particularly when it comes to animating a mesh in an external environment, which can be subject to external forces. Since the surface of the human body (i.e., skin) is highly deformable, it experiences significant deformation when interacting with external forces (e.g., gravity, wind) or external objects or even with itself (i.e., lips colliding with themselves or fingers pinching together). Such deformation is extremely difficult to model using geometry-based techniques, which invites us to model it using physics-based techniques. While the choice of constitutive material model, FEM model, etc. are all very important in how accurate the deformation of the flesh is depicted, another important choice is that of the numerical time integrator used to advance the system through time.

Numerical integration is a long-studied field, beginning with the seminal work of Euler ca. 1707. For physics-based animation, we are interested in solving the ordinary differential equations of motion. One important choice for numerical integrators is the choice of the discrete timestep h. A small timestep yields a more accurate solution at the cost of requiring more computation and a longer execution time. An important question to ask is how do we evaluate how good a certain solution is? One way is to see how accurately our solution mimics known invariants of motion (such as energy/momentum conservation or symplecticity). Unfortunately, [59] showed that it is not possible to conserve everything with a fixed timestep h. This result led to adaptive timestepping schemes [40] being preferred in accuracy-critical applications (i.e., bridge modelling, aircraft modelling). Such simulations can take arbitrarily long (since the timestep can be arbitrarily small), which makes them less desirable even for offline graphics simulations (such as movies) and nigh-unusable for more time-critical applications (i.e., interactive applications such as games or surgery simulators).

Many techniques have been developed, from symplectic integrators [105, 66] to energy-projection methods [85, 133, 139]. However, these methods can exhibit stability issues, particularly when used with large timesteps. Surprisingly, for large timesteps the best choice of integrator seems to back backward Euler [145, 143, 144], a variation of Euler's original method that requires solving an implicit system of equations. While this integrator does not conserve energy, momentum, or symplecticity is has a property that makes it very

attractive for interactive applications: stability. This extremely important property makes it the integration scheme of choice, and many well-established methods for real-time physical simulations are fundamentally built on it, such as Position-Based Dynamics [114, 104, 17] and Projective Dynamics [95, 27]. While backward Euler exhibits astonishing stability, it has one major drawback: the introduction of artificial numerical damping. While in some cases this may be construed as a positive (after all, real-world systems exhibit damping), an important thing to note is that this damping is uncontrollable and inherently not physics-based: it is dependant on quantities such as mesh resolution, numerical stiffness, and timestep rather more physics-based quantities like drag coefficient or heat dissipation.

We attempt to remedy the damping without sacrificing stability by revisiting some ideas from previously-proposed energy projection methods. Our first projection method is inspired by two simple observations: 1) implicit symplectic methods (such as midpoint) tend to introduce small errors that accumulate over time and 2) backward Euler's damping properties. We introduce a scheme to use backward Euler to damp out any excess energy introduced by implicit midpoint, resulting in an energy-conserving integration scheme that will be stable for long runs. Furthermore, both of these steps can be accelerated using the same idea as Projective Dynamics, making this integration scheme suitable for real-time applications. This work was published in ACM Transaction on Graphics [43]. One issue with the previous method is that, while it is energy conserving, it does not conserve the momentum structure as backward Euler damps out the momentum which is conserved by implicit midpoint. To remedy this, we introduced a new energy-momentum projection method by formulating an optimization problem that projects the result of an arbitrary integrator to the constant energy manifold while preserving the integrator's momentum properties. While this is a constrained optimization problem with nonlinear constraints, we can leverage the fact that we only have 7 constraints (1 for energy, 3 for each of angular and linear momenta) to introduce some numerical tricks that greatly speed up the solve while giving a good approximation of the real solution. These energy-conserving techniques can then be applied to produce much more lively animations, which make our facial model interact with the environment in a more lively and animated way. However, currently, this model is unable to make facial expressions and is just a passive model that does not move. One final component to having a human facial model is how we parameterize the facial

expressions.

Traditional techniques have been geometry based, and parameterized the face via morphing, also called blendshapes [91]. This framework represents a facial expression as a linear combination of two or more "target" facial expressions, usually in extreme poses captured from a real subject. This representation has several issues. Aside from being unable to handle external interaction with physics-based phenomena, its parameterization of the face ignores how real-world facial expressions work. Human facial expressions are driven by sub-dermal muscles, and the resulting expressions are the combination of nonlinear interactions between the muscles and the external passive tissue (skin). This makes a linear interpolation of the *surface* (as in blendshapes) produce a wrong result and is part of what makes such facial animations look "stiff' and "plasticky". To remedy this, many muscle models have been proposed for facial animation [132, 39], however they usually require extensive manual labor to create a muscle model for a particular subject. Recently, several techniques have relaxed the parameterization of the muscle model in order to fit surface scans of a subject and automatically find a muscle model [71, 70]. In this work, we introduce a method that builds on this framework but reduces the parameter space by taking advantage of subject-specific parameters, specifically the muscle geometry of the subject.

In summary, by combining the energy-momentum conserving integration scheme for vivid simulations of the passive tissue of the face with the muscle model we propose for the active tissue, we are able to create a facial model that is capable of both creating facial expressions while under external forces and produce lively motion, similar to what a real human face looks like under the effect of external forces. We believe that such models will only become more and more popular as the field of physics-based human models becomes more and more developed, and we hope this work to this development.

CHAPTER 2

USER-GUIDED LIP CORRECTION FOR FACIAL PERFORMANCE CAPTURE

Facial performance capture has become the industry standard for generating facial animation for virtual characters, especially in the case of digital doubles of real actors. Despite great progress, one area that remains a challenge is the lip region, which poses problems because of the complex appearance properties caused by lip wetness, shadows, and continuous occlusion and dis-occlusion. There constructed geometry in this region is often plausible, capturing most of the gross lip motion, but effects such as bulging, precise lip curl and adhesion (so-called "sticky lips") are often missed. To make matters worse, this region is extremely important for realistic facial motion, in particular when it comes to speech, and omit-ting those subtle effects can be the difference between realistic and uncanny-looking characters. For this reason, for production quality digital humans, typically many hours of artist time go into manually sculpting 3D lip corrections, even on top of the highest quality performance capture results.

In this work we aim to alleviate the problem by providing a user-guided approach for correcting lips in facial performances. The main idea is to choose a small set of important frames in a performance sequence that identify problematic lip shapes that were not reconstructed well, and then manually correct these few frames.Our system will then learn the 3D correction and automatically propagate the sparse manual corrections to the full sequence, using a gradient-based regression framework. Our approach lends it-self well to incremental shot production, where the user can start with only a few corrected frames and gradually add corrections only where needed. Furthermore, since it can be time-consuming and cumbersome to manually sculpt 3D corrections for even a small number of problem frames, we additionally propose a simple method to apply lip corrections through an intuitive 2D sketching interface, which can generate plausible 3D lip corrections in little

time without the need for artistic skill.

Our approach draws on related ideas for facial capture that pay particular attention to the lips. In particular, Bhat et al. [21] show a production application of marker-based facial capture that uses hand-drawn lip contours to obtain improved lip shapes, how-ever they must annotate each and every frame of the performance.The key idea of our method is to learn the underlying corrections in order to reduce the amount of manual effort. With a similar goal, Garrido et al. [57] capture ground truth data of lip motions using high-frequency lip tattoos and apply automatic corrections in a monocular facial performance capture scenario. However, their goal is to create a generalized regression framework for basic improvements to low-resolution lip shapes, targeting more a mass-consumer audience, while our work is designed for shot-specific production-quality lip refinement at high quality, with an easy-to-use interface.

The key idea of our approach is to predict the inner shape of the lips from the shape of the surrounding area of the face, which is assumed to be captured with high accuracy because it is not affected by the artifacts due to wetness and occlusions. Inspired by anatomically-based facial animation [132, 39, 88, 70], where the facial deformation is determined from underlying muscle activations, we argue that the inner lip shape is inherently linked to the outer lip shape since the overall lip deformation is determined by the activation of the lip muscles underneath. In our work, we avoid the complexity of establishing an explicit anatomical facial model but instead train a regressor to directly predict the inner lip shape from the surrounding region, which we assume to be captured well. Our lip shape regressor operates in the gradient domain, which makes it robust to global translations of the mouth.Also, our method is designed specifically to operate only with small amounts of training data in order to minimize the burden on the user.

We demonstrate our user-guided lip correction method on several performance capture sequences of three different actors. Additionally, we provide an evaluation of our approach on previously presented ground truth lip performance data [57]. We believe this work can have a large impact on high-quality facial performance applications by providing improved reconstructions of a very important part of the face at little cost to the user.

2.1 Related Work

Face Capture and Animation. There are several approaches for creating facial animation, including blendshape animation [91], physically-based animation [146, 132, 39, 83, 70], marker-based motion capture [21], and dense marker-less performance capture [30, 15, 56, 86]. Capturing a dense surface performance currently offers the highest fidelity as it can faithfully reproduce all the subtle nonlinear effects of a kin, and is thus the method of choice for production-level facial animation.

Facial performance capture has received a tremendous amount of attention over the past decades. In recent years we have witnessed significant improvements in acquisition of high-fidelity facial expressions and performances [2, 12, 30, 60, 15, 56, 86]. In addition to the facial geometry, high fidelity eyelids [20], eyes [19, 18], skin microstructure [64, 115], facial hair [13] and scalp hair [99, 67] can also be captured at high accuracy.

Expressive and detailed lip reconstruction, however, remains an unsolved challenge, due to the extreme deformation of the lip tissue, dramatic color changes caused by blood flow, challenging specular reflectance due to saliva, as well as recurring occlusion and dis occlusion of the inner lips. Recently, Garrido et al. [57] succeeded at capturing high quality lip shapes by placing high-frequency tattoos on the lips, circumventing some of the aforementioned challenges. While suitable for generating a shape dataset, the tattoos destroy the natural appearance of the lips and are thus less attractive for creating digital doubles, and complicated tattooing is not always an option for actors in a high-end production setting. As a consequence, VFX workflows still rely on manually correcting 3D lip shapes, often on a per-frame basis, which is a lengthy and cumbersome task that requires talented artists. Our work is designed to dramatically reduce the artist work spent on 3D lip refinement for high-fidelity performance capture, by requiring only a handful of corrected frames and then filling in the remaining performance with a regression-based learning approach.

Lip Capture. High-fidelity lip appearance and deformation are especially important for high quality facial animation [78, 155] and mouth/lip tracking and reconstruction has been a long-studied problem. Many approaches use 2D contour lines to track lips in the image space [119, 147, 50, 10] but do not reconstruct the dense 3D geometry. In our paper, we provide a 2D image-based editing tool to refine the tracked lip shapes but with the

goal of generating corrections of 3D geometry. Similar to previous sketch-based interfaces [166, 89, 111], our 2D editing tool allows for intuitive control of lip shapes and is easy to use even for non-expert users. This idea for simple facial animation editing is akin to keyframe-based performance editing tools, such as those used during expression retargeting [128, 161].

Recently, there have been several works on modeling and reconstruction of 3D shape of the lips in multiview capture setups [30, 5, 21, 79] or even just from monocular video [93, 97, 57]. In particular, Bradley et al. [30] propose edge-based mouth tracking which improves the lip reconstruction with detected contour constraints. Anderson et al. [5] track the 2D lip contours and automatically register the mouth region to 3D geometry by aligning the lip contours with predefined isolines on the surface. A data-driven approach was also proposed by Liu et al. [97] for real-time lip shape refinement, and landmark-based lip correction is proposed int he real-time system of Li et al. [93]. Olszewski et al. are able to demonstrate lip animations such as "sticky-lips" using deep learning in a head-mounted capture system [121]. Even though impressive results have been achieved with these prior works, expressive lip shapes and subtle effects on the lips are still missing for production-level reconstruction. Our method is orthogonal to all of these 3D lip tracking approaches and enables corrections of the en-tire performance with minimal artist input. Bhat et al. [21] show a production application of marker-based facial capture and improve lip tracking with per-frame hand-drawn lip contours. With corrections required only for a sparse set of frames, our method automates the editing of the remaining facial animation frames, which significantly reduces the amount of manual effort. Furthermore, our approach provides temporal smoothness and editing consistency, which is challenging to achieve with per-frame correction.

Garrido et al. [57] learn the difference between high-quality lip shapes and coarse monocular lip reconstructions based on a regression function and automatically apply the corrections to low-resolution lip shapes from monocular video. Our approach shares a similar goal to their work. Unlike their approach, however,our method learns the lip corrections directly from the surrounding facial deformation and is designed for shot-specific, production-quality lip refinement on top of highest quality facial performance capture.

The following work was published in Computer Graphics Forum (Vol. 37 Issue 8, 2018) and present at the Symposium for Computer Animation 2018.

2.2 Automatic Lip Correction Learning

We take a user-guided approach for improving lip shapes in performance capture applications. Given a reconstructed mesh sequence,a user can select a sparse number of frames that contain systematic reconstruction errors caused by the challenges inherent to 3D lip capture, and manually provide the corrected shapes. Our system then learns the type of corrections the user desires, and applies a corresponding refinement to the entire sequence. The process can then be repeated incrementally if some frames remain problematic. Finally, we show results for various performances of three different actors, as well as a quantitative evaluation of our method using ground truth lip data Section 2.3.

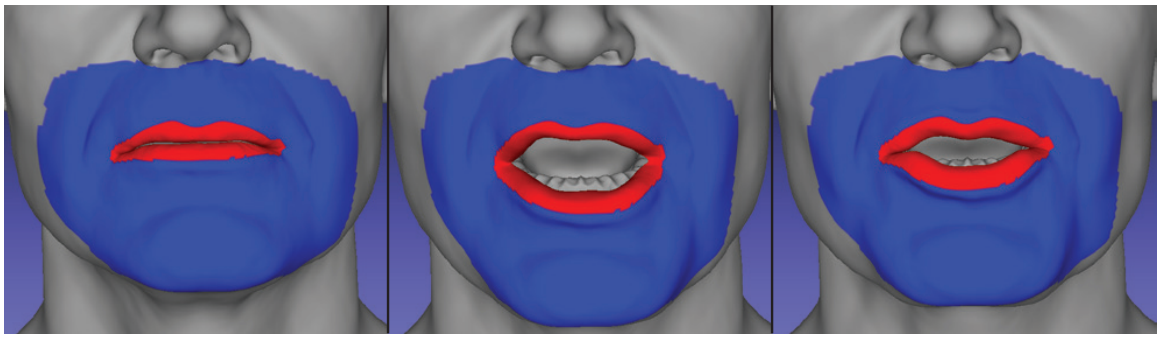

Figure 2.1: We segment the mesh into a mouth region (blue) and an inner lip region (red). The rest of the mesh (gray) is ignored.

Many facial capture systems can provide accurate and high-resolution reconstructions of the actor with only a few problematic areas, one of which are the inner lips. The key idea behind our method is to use the accurate portions of the mesh to predict the inaccurate parts. Since the inner lip shape is heavily influenced by the underlying muscles that also dictate the shape of the mouth area,we believe that this relationship can be learned. In

$$\min_{\theta} \frac{1}{2k} \sum_{i} ||\theta \mathbf{y}^{(i)} - \mathbf{x}^{(i)}||_F^2 \tag{2.1}$$

We then create the matrix $\mathbf{Y} \in \mathbb{R}^{9m \times k}$, whose ith column is $\mathbf{y}^{(i)}$,and an analogous matrix $\mathbf{X} \in \mathbb{R}^{9n \times k}$ for the $\mathbf{x}$ vectors, recasting the regression problem in matrix form:

$$\min_{\theta} \frac{1}{2k} ||\theta \mathbf{Y} - \mathbf{X}||_F^2 \tag{2.2}$$

Which can be solved using the normal equations

$$\theta \mathbf{Y}\mathbf{Y}^\mathsf{T} = \mathbf{X}\mathbf{Y}^\mathsf{T} \tag{2.3}$$

We observe that the resulting system is underdetermined as $k \ll n, m$. Due to the highly detailed facial geometry that is encoded in the meshes, the triangle counts are high (e.g., m=19315) and the matrix θ is large, suggesting high memory costs and the risks of over fitting. To avoid these issues, in the following we propose a reduced regression approach, where the number of unknowns depends on the number of corrected frames k (typically under 20) rather than the resolution-dependent quantities m and n. The first step towards this reduction is forming the singular value decompositions (SVDs) of $\mathbf{Y}$ and $\mathbf{X}$, $\mathbf{Y} = \mathbf{U_Y}\mathbf{C_Y}$ with $\mathbf{C_Y} = \mathbf{\Sigma_Y}\mathbf{V_Y^\mathsf{T}}$ and $\mathbf{X} = \mathbf{U_X}\mathbf{C_X}$ with $\mathbf{C_X} = \mathbf{\Sigma_X}\mathbf{V_X^\mathsf{T}}$, where we introduce coefficient matrices $\mathbf{C_Y}$ and $\mathbf{C_X}$, respectively.

Using the invariance of the Frobenius norm under orthonormal transformations and plugging in the coefficient matrices, we obtain

$$||\theta \mathbf{Y} - \mathbf{X}||_F^2 = ||\mathbf{U_X^\mathsf{T}}(\theta \mathbf{Y} - \mathbf{X})||_F^2 = ||\mathbf{U_X^\mathsf{T}}\theta \mathbf{U_Y}\mathbf{C_Y} - \mathbf{C_X}||_F^2 \tag{2.4}$$

We then define the reduced unknown matrix $\overline{\theta} = \mathbf{U_X^\mathsf{T}}\theta \mathbf{U_Y}$, solving the problem

$$\min_{\overline{\theta}} \frac{1}{2k} ||\overline{\theta}\mathbf{C_Y} - \mathbf{C_X}||_F^2 \tag{2.5}$$

which leads to normal equations

$$\overline{\theta}\mathbf{C_Y}\mathbf{C_Y^\mathsf{T}} = \mathbf{C_X}\mathbf{C_Y^\mathsf{T}} \tag{2.6}$$

where the matrix $\overline{\theta}$ has size $k \times k$ and can be therefore computed very quickly. For any new shape j, we can reconstruct the inner lip deformation gradient vector $\mathbf{x}_j$ by computing

$\mathbf{y}_j$ as above and applying our reduced regression model: $\mathbf{x}_j = \theta y_j$, where we recall that $\theta = \mathbf{U_X}\overline{\theta}\mathbf{U_Y^T}$.

Note that the coefficients of the large matrix θ are never explicitly evaluated and instead, the matrix is compactly represented in its factorized form $\mathbf{U_X}\overline{\theta}\mathbf{U_Y^T}$. Finally, we convert the resulting vector of deformation gradients $\mathbf{x}_j$ back to vertex positions. This corresponds to solving a sparse linear system, using the positions of the vertices at the border between the inner lips and outer mouth region as Dirichlet boundary conditions.

2.3 Results

We first ran our method on dialogue sequences from three different actors, using the 2D contour tool to selectively correct frames that we add to our linear regressor. Then, to evaluate the accuracy of our method we applied our method to the the data set from Garrido et al. [57]. This data set uses high-frequency patterns to capture the inner lip region more accurately, so we can use this as a ground-truth data set to evaluate our method.

2.3.1 Lip Correction Results

Figure 2.2 shows our method applied to a dialogue sequence from Actor 1. With only 11 frames corrected, we were able to greatly improve the quality of the capture. In the original capture, some expressions where the mouth should be closed have a small gap between the lips (top row) and the corners of the lips lose their fleshiness (second and third rows). After using our tool to correct a few of these frames, similar issues in other frames are automatically corrected. In expressions where the mouth is open, the capture system can sometimes produce plausible results (bottom row). For such frames, our method does not introduce errors when compared to the initial capture result. The frames shown in Figure 2.2 were not part of the training set; they were corrected by our method. The full training set is shown in Figure 2.3. Our method does not have a noticeable effect on the structure of the individual triangles, as shown in Figure 2.4.

In Figure 2.5 we applied our method to a different dialogue sequence from a different actor than Figure 2.2. In this sequence, the original capture exhibited artifacts due to interference with the teeth (note that neither the original capture system nor our method is attempting to reconstruct the motion of the teeth). These artifacts are circled in red in

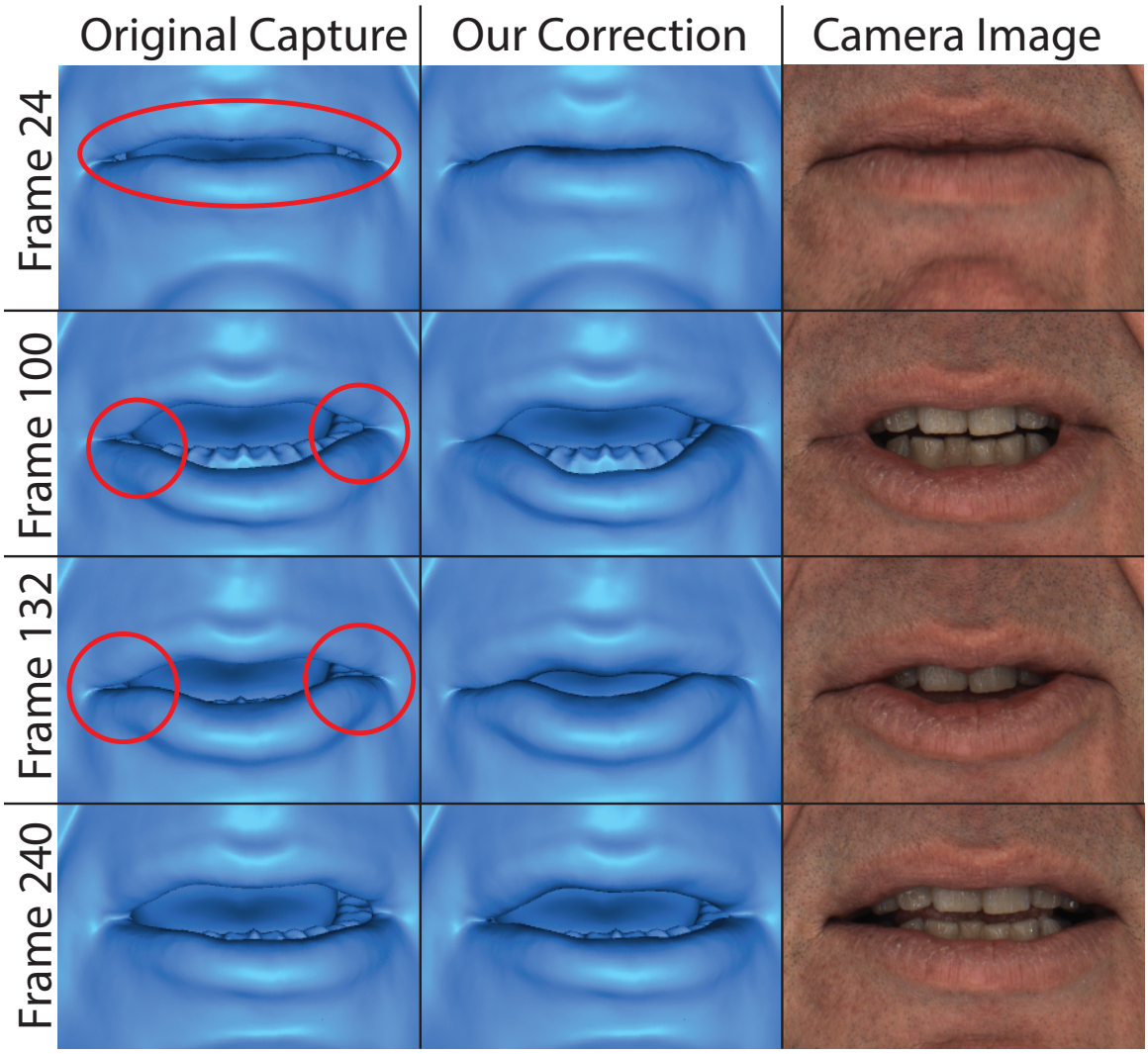

Figure 2.2: The performance was trained using the samples in Figure 2.3; the frames shown here are not in the training data set. We are able to correct many artifacts that occur during capture: the lips not being properly closed (top row) and loss of fleshiness in the corners (second and third column). In situations where the original capture is feasible (bottom row), our method does not downgrade the results.

Figure 2.5. After correcting the lip shapes in only 8 meshes, our method is able to propagate these corrections throughout the performance. As before, the frames shown in Figure 2.5 are not part of the training data set.

We show the versatility of our method by applying lip corrections to a third actor, performing extreme lip shapes, as shown in Figure 2.6. Again, only 8 frames are corrected from a sequence of 350, and the frames from Figure 2.6 are not part of the training set.

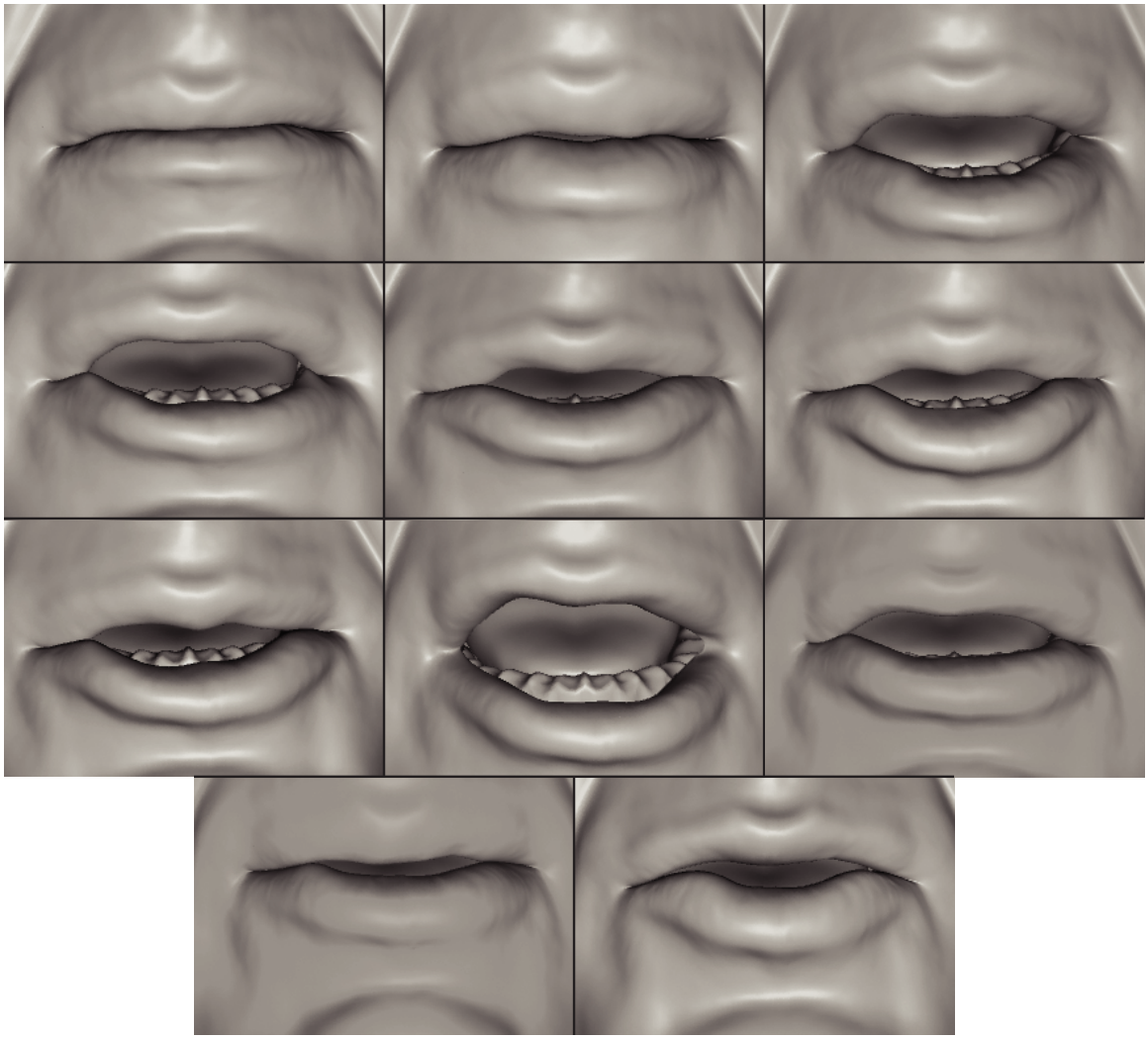

Figure 2.3: These 11 meshes are the entire training data set used to generate the corrected lips for the sequence shown in Figure 2.2.They were created by applying our sketch-based corrections on the corresponding captured meshes.

2.3.1.1 Evaluation on Lip Tattoo Data

To test the accuracy of our method, we used a performance from the lip tattoos data set from [57] as a ground truth (276 frames in total) and tested how accurately we were able to reconstruct the inner lip region. We selected a handful of expressions for our training data set and applied our correction to the rest of the frames in the performance. Figure 2.7 shows the results of this reconstruction when we added 10 poses to the training set. Figure 2.8 shows an error map of the Hausdorff distance between our reconstructed meshes and the ground truth meshes. While we are not able to perfectly reconstruct the original inner lip shape with only 10 training samples, we are able to obtain a very good

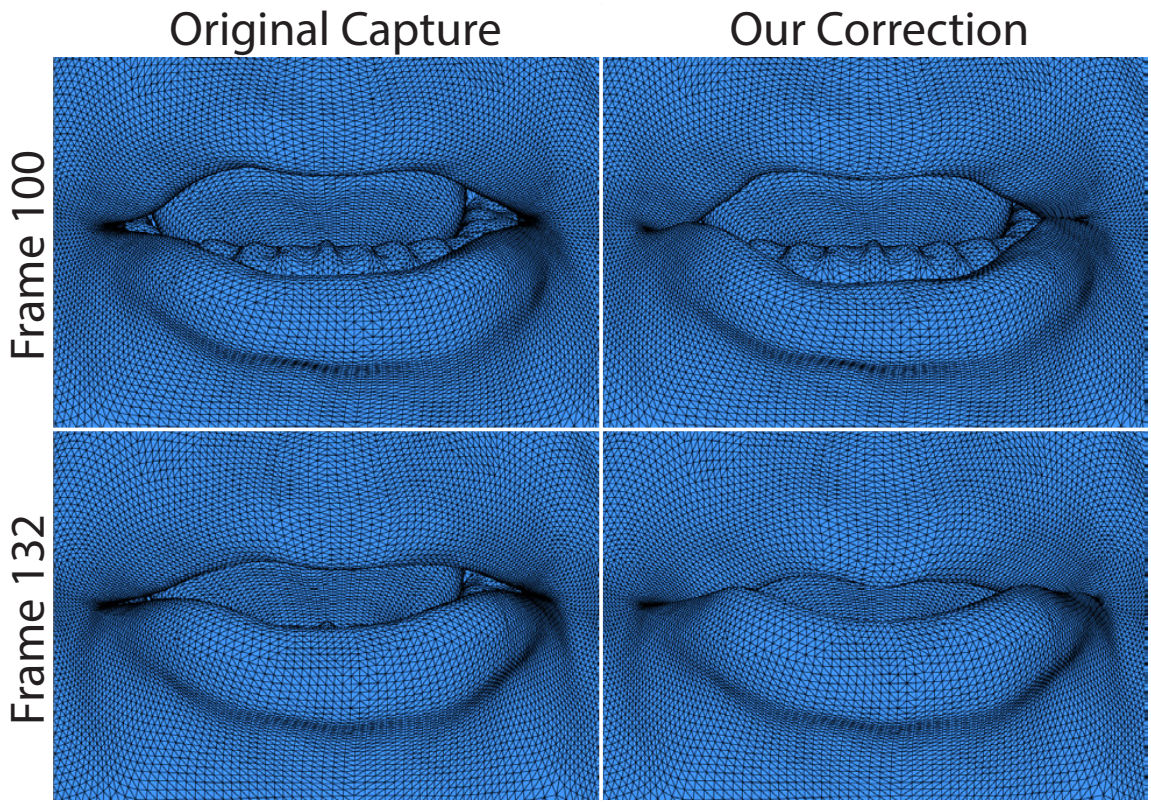

Figure 2.4: A wireframe rendering of two results from Figure 2.2, showing the influence our method has on the individual triangles.

approximation. To evaluate how the number of training points affects the final result, we selectively added meshes from this performance as training data and evaluated the absolute error of the deformation gradients: $\epsilon = ||\mathbf{x}^\star - \mathbf{x}||$. Figure 2.9 shows how the number of training samples affects the error of the reconstruction, taking the average ϵ over all frames in the sequence. The training samples were not chosen arbitrarily: we selected frames at the extreme poses where possible. This explains the diminishing returns observed in Figure 2.9: the sequence only has a limited number of extreme expressions, with several of them being similar to each other. Once the extreme poses have been added to the training data, adding the in-between expressions does not have as dramatic of an effect. We can see what effect this error has visually in Figure 2.10. As we increase the number of training samples, the observable differences in this test frame become increasingly subtle.

One important parameter that affects our results is how we choose the inner lip/mouth regions (red and blue regions from Figure 2.1). If we choose an inner lip region that is too small, then we will end up keeping portions of the mesh that were not reliably captured

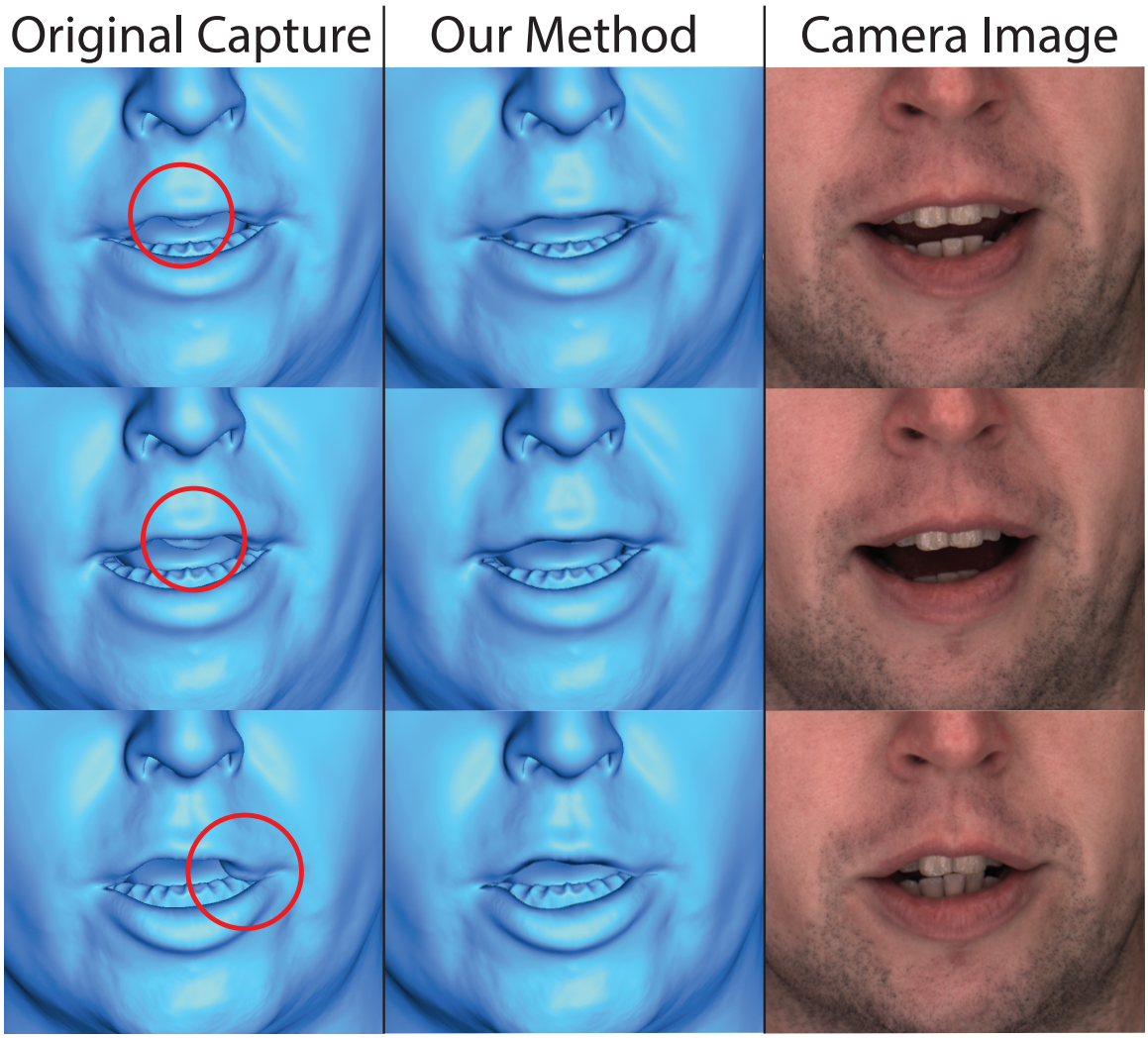

Figure 2.5: In this dialogue capture, the actor's teeth interfered with the tracking abilities of the system, causing strange geometric artifacts to appear in the lip regions (circled in red, left column). Our method is able to remove these artifacts from all frames by only correcting a few example frames. The frames depicted in this figure were not part of the training set.

(i.e., keeping bad corner shapes). Conversely, if the inner lip region is too large then we miss out important features that can help us predict the inner lip region (e.g., the outer lip can be important to accurately reconstructing the inner lip shape). It is important to select the regions such that they coincide with the boundary between the high/low confidence values of the capture system. We want to reconstruct the regions that are low confidence while maximally leveraging the regions that are high confidence.

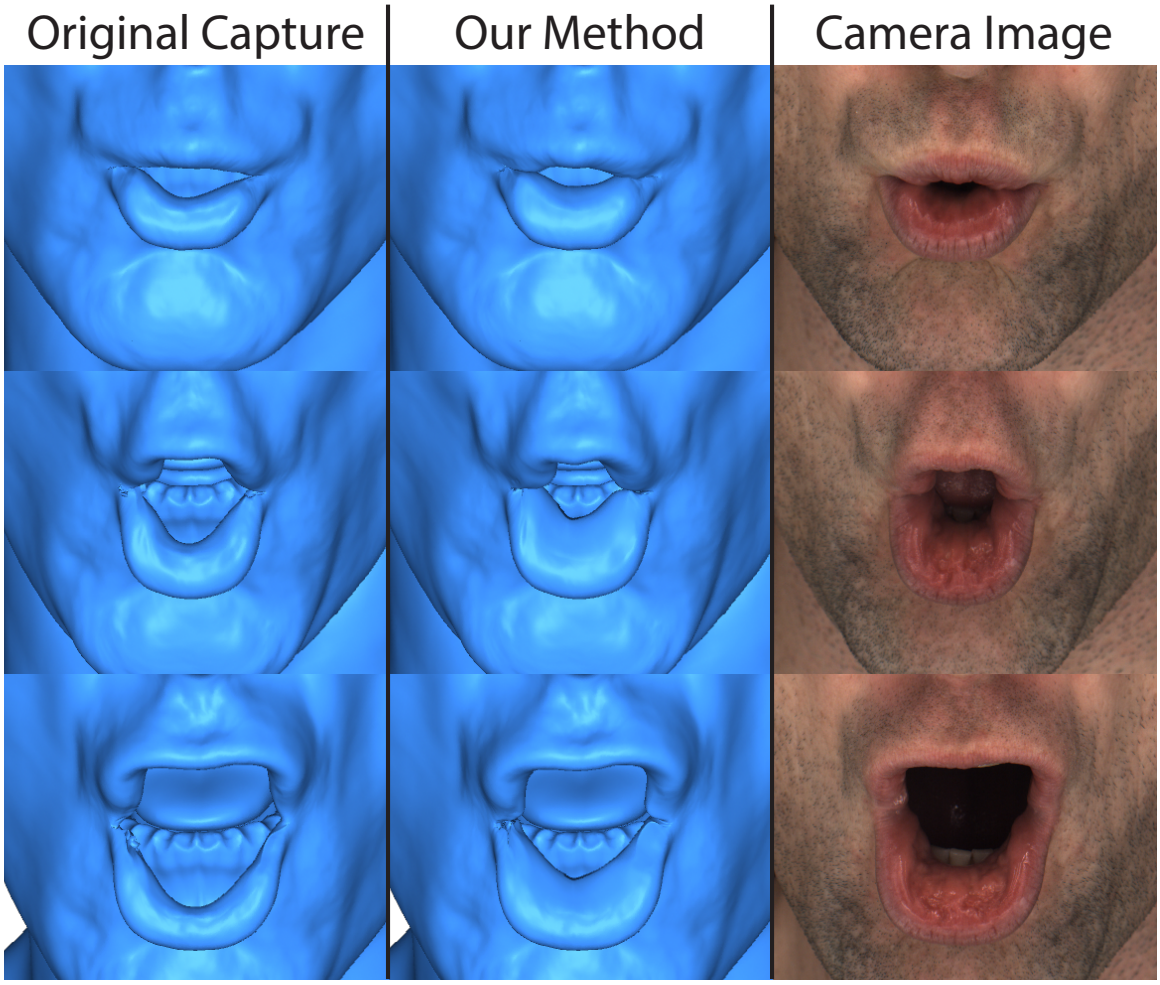

Figure 2.6: We show the versatility of our method by applying lip correction to a third actor, performing extreme expressions.

2.3.1.2 Performance

Once the training data has been obtained, training the network is much faster, for example the network used for Figure 2.2 took 5.47 seconds on an Intel i7-4720HQ CPU at 2.6 GHz. Once the network has been trained, applying the regression to each mesh takes 5.15 seconds per frame (taken as an average over 10 frames). This timing includes converting between the position and gradient domains for every input mesh.

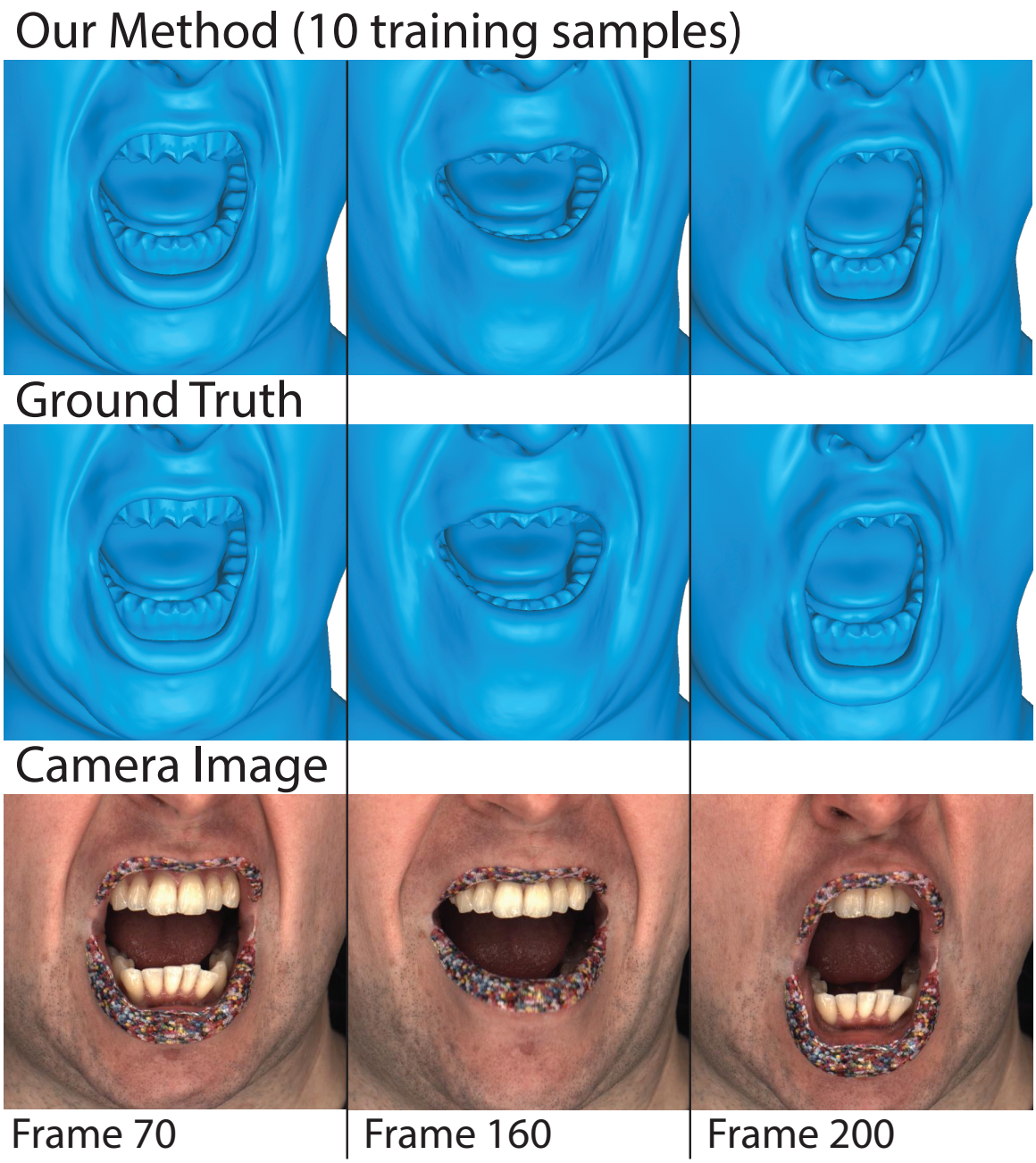

Figure 2.7: Using the data set from [57], we can evaluate how accurately our method reconstructs the inner lip region. With 10 training samples, our method can reconstruct the inner lip region with only subtle differences from the ground truth.

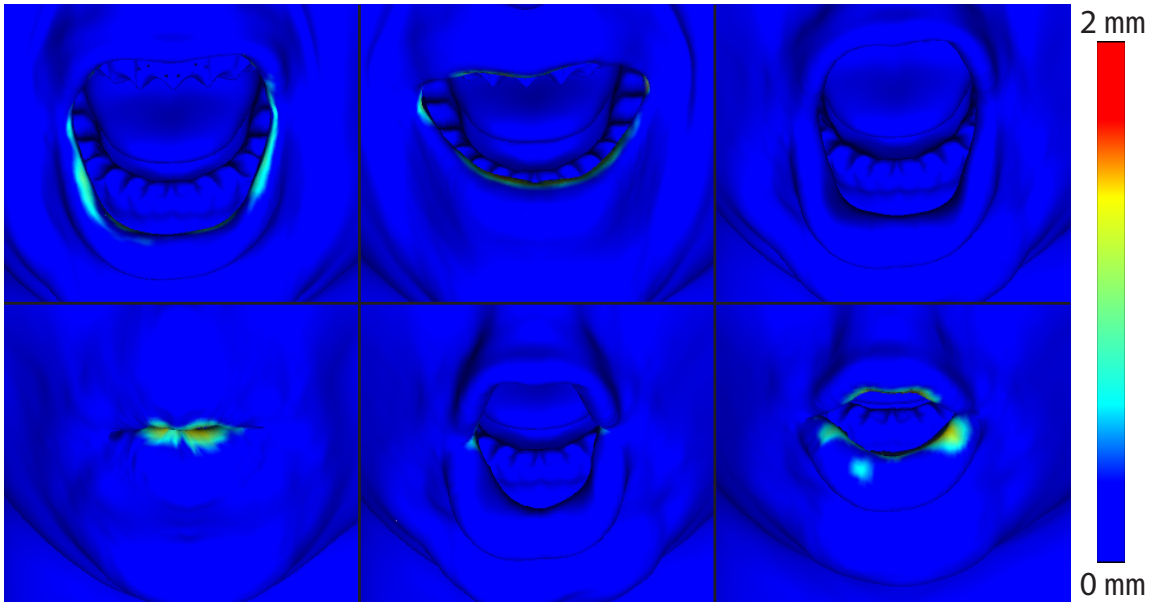

Figure 2.8: An error map of our reconstruction compared to the ground truth using the data set from [57]. The top row shows the same expressions as Figure 2.7 and the bottom row contains additional lip shapes from the sequence.

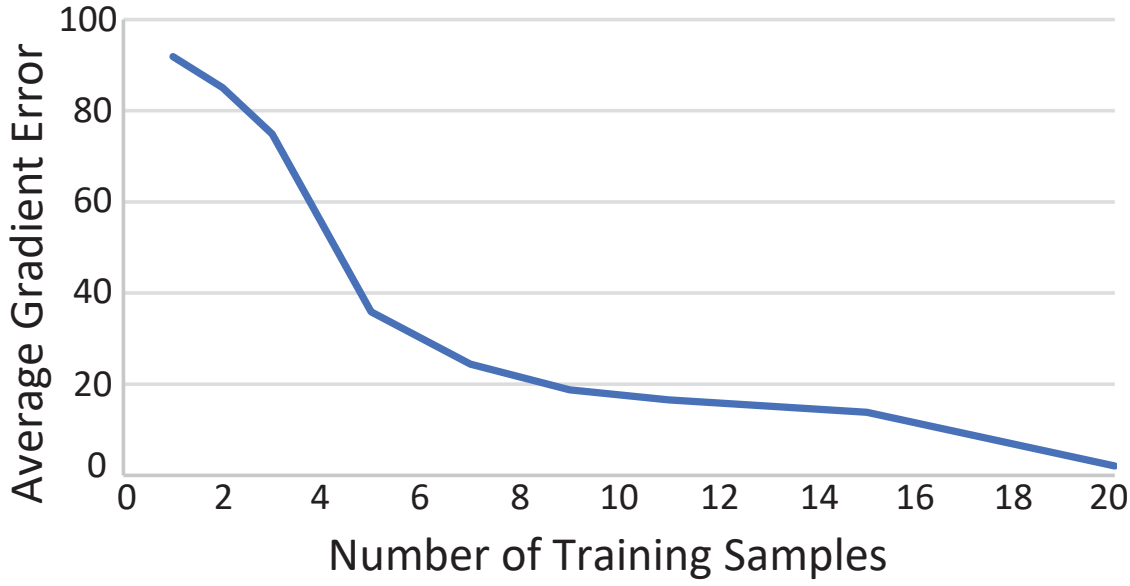

Figure 2.9: The effect of the number of training samples on the average error of the sequence from Figure 2.7. Once all of the important poses are in the training set, adding additional poses yields diminishing returns.

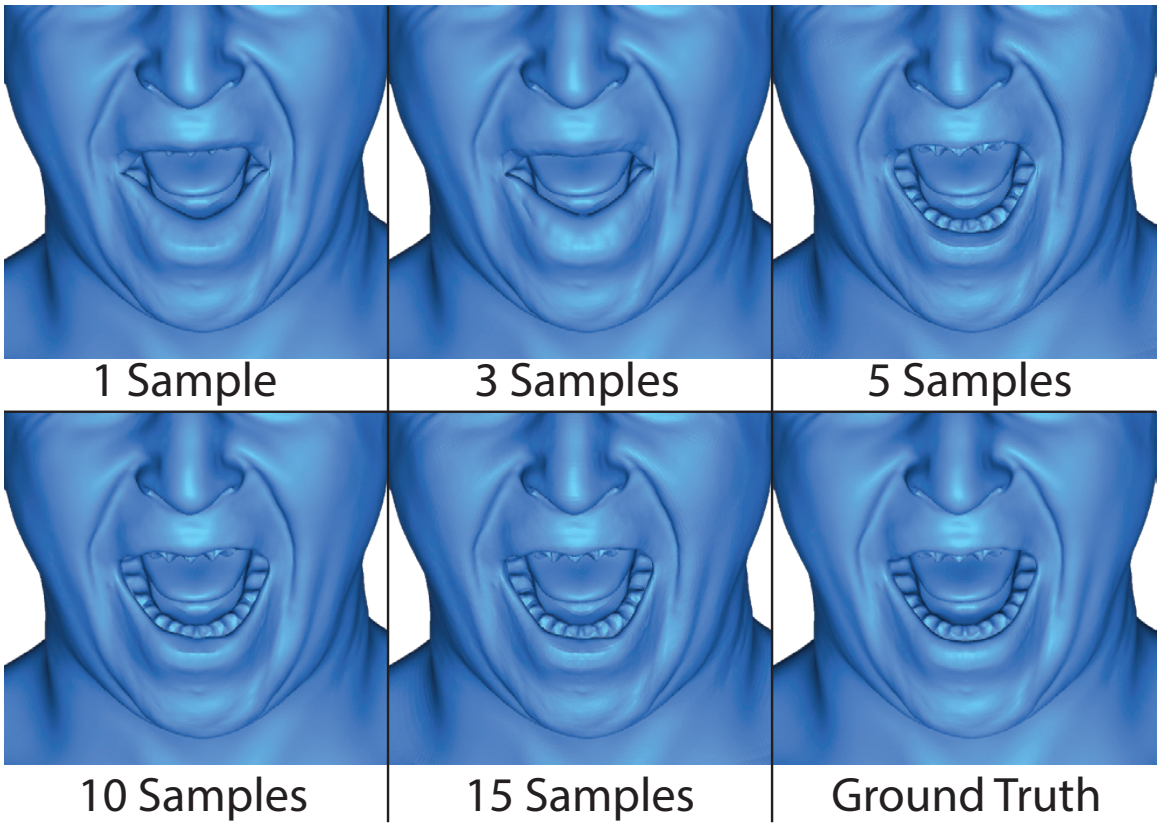

Figure 2.10: The visual effect the number of training samples has on a test frame. The first several additions of new training samples make a significant impact, but larger numbers of training samples yield diminishing returns.

CHAPTER 3

ENERGY-CONSERVING NUMERICAL TIME INTEGRATION

While geometric methods can be used to great effect for facial simulation, animating a face and capturing all of its motions in an external environment remains a challenging problem. As previously mentioned, geometry-based methods have a difficult time handling physical phenomena, such as contact or external forces (e.g., wind). Yet, the flesh is very elastic and can deform significantly under such conditions, such as applying a strong wind force from a leaf-blower, as shown in Figure 3.1.

Figure 3.1: 185 MPH is applied to sever subjects' faces from a leafblower [72]

Replicating such behavior in a simulated environment requires many components, such as a constitutive deformation model or a volumetric element choice. In this section, we will study the choice of numerical time integrator, the choice of which has a very significant effect on the resulting animation.

Adaptive time stepping methods with error control are popular in scientific computing. These methods are necessary in engineering applications, e.g., when designing an airplane

or nuclear reactor, where simulation accuracy may be critical. Unfortunately, adaptive time stepping is incompatible with the requirements of real-time applications, where we have only a limited computing budget per frame. Despite progress in parallel-in-time methods, time stepping is a fundamentally sequential process, difficult to parallelize. Real-time simulations therefore have to compromise accuracy in order to retain interactivity. However, the goal is to do this gracefully and retain physical plausibility by keeping the inevitable errors under control. The most striking manifestation of inaccuracy is the case of instabilities ("explosions"), where small discretization errors compound and the discrete approximation departs dramatically from the true continuous solution. The classical way of avoiding such catastrophic failures is to reduce the time step. Unfortunately, this is not an option in real-time physics which needs to operate within a limited computing budget.

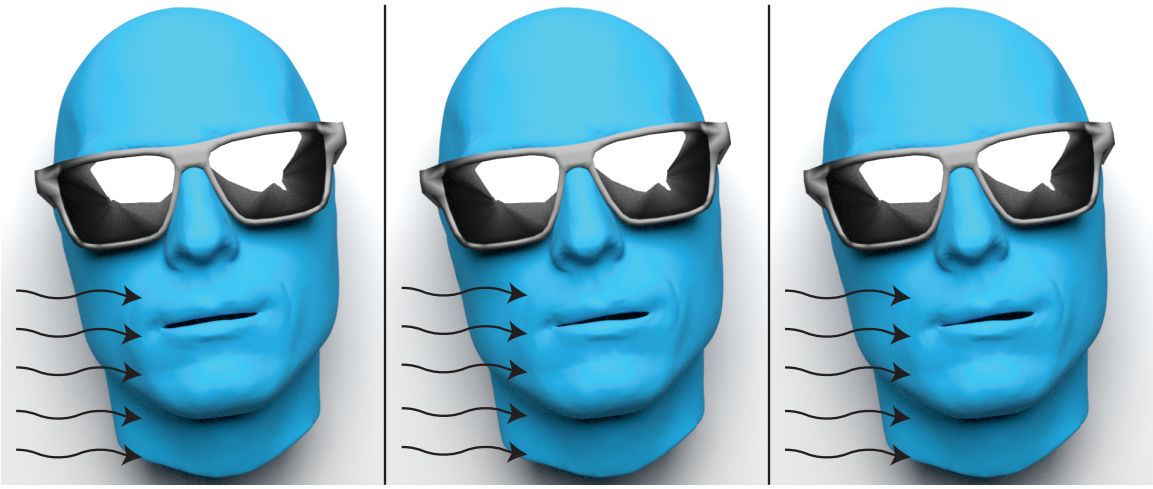

Figure 3.2: A wind force is applied to this face. Backward Euler damps out all of the detail, leaving an almost static image.

Unfortunately, for a fixed timestep [59] shows us that it is not possible to conserve all invariants of the equations of motion, and adaptive timestepping schemes can be prohibitively slow for the needs of computer graphics, where we usually want to use as large of a timestep as possible. Under such conditions, the integrator of choice is usually backward Euler, whose stability properties has been well-studied and applied to cloth [9]. Unfortunately, this introduces significant numerical damping, which can be dramatic. Figure 3.2 shows a simulation of a neutral facial expression with a wind force applied from

left to right. The high-frequency ripples we expect to see Figure 3.1 are damped out and the result is an almost static expression, with some very subtle differences between frame 0 and frame 10.

Our goal is to design an integration scheme that can preserve the vivid motion damped out by backward Euler while still being stable for the large timesteps desired by computer graphics. We accomplish this by observing that real-world solutions observe certain conservation properties, which arise from the conservation of many first integrals of the mechanical systems, notably the Hamiltonian (i.e., total energy) and momentum (both angular and linear), as well as the symplectic form of the system. While [59] shows us it is impossible to conserve everything, we will find that conserving the Hamiltonian and the momenta can produce very good results, while also enforcing stability. By employing energy/momenta projection schemes and using several numerical approximations to speed up the subsequent solves, we are able to produce integration schemes that preserve the vivid motion of the original system without sacrificing stability that is also suitable for real-time interactive applications.

3.1 Related Work

Integration in physics-based animation. Explicit integration methods are usually very fast and simple to implement, but they can be very unstable when applied to large time steps. Many adaptive schemes have been developed that choose a suitable time step to guarantee accuracy [40]. While these schemes can produce very accurate results, these approaches are usually not used in real-time physics due to their variable computing requirements.

Implicit methods have been used in physics-based animation [145, 143, 144] due to their good behavior when using large time steps. Baraff and Witkin [9] showed that backward Euler can work very well for cloth simulations using large time steps, while Choi and Ko [36] proposed to reduce the artificial damping of backward Euler (BDF-1) by its second order version, BDF-2. Volino and Thalmann [149] went even further and proposed to simulate cloth using implicit midpoint, which as a symplectic integrator does not introduce numerical dissipation. Combinations of implicit and explicit methods have also been explored with promising results [46, 32, 138, 53].

Numerical Solutions for Integration Methods. Many recent methods for fast physics-based simulation rely on implicit integration, often formulated as a minimization problem [106, 58]. The chief advantage of the optimization formulation is that instead of solving systems of (typically nonlinear) equations, we can instead solve a minimization problem which has numerical benefits [80]. The optimization formulation also enables generalizations such as inequality constraints or contact constraints [6, 73].

When computing time is limited, the minimization problem is usually not solved exactly and instead, an approximate solution is accepted. Many modern real-time physics simulators can be viewed as approximate solvers for the backward Euler optimization problem.

Position based methods. One popular family of methods is known as Position Based Dynamics (PBD) [114, 104, 17, 54], a physics-based simulation approach which specifically caters to the needs of real-time applications such as computer games. Closely related techniques have been followed in the Nucleus solver [135]. Position-based methods have been extended with more advanced solvers [112, 153, 82], finite element models [113, 16], fluid simulation [102] and a unified simulator supporting multiple phases of matter [104]. Recently, it was shown that a small modification of the original PBD algorithm [114] termed XPBD makes the method converge to the backward Euler result [103]. As pointed out by Liu [95], Position Based Dynamics can be derived from backward Euler integration (BDF1) and therefore inherits its artificial numerical damping. It is possible to upgrade Position Based Dynamics to BDF2 [17]. This helps, but it does not completely eliminate the undesired numerical dissipation. BDF2 can be replaced with, e.g., implicit midpoint or another implicit symplectic scheme which avoids numerical damping, but this compromises stability. Real-time applications cannot risk instability and therefore current systems accept the "lesser evil" of uncontrollable numerical damping.

Projective Dynamics. Faster constraint projection can be achieved using Newton's method [61, 49] or with Projective Dynamics (PD) [95, 27]. Newton's method is the classic method for computing accurate solutions to optimization problems, and is typically used to compute the solution to backward Euler in real-time animation. Unfortunately, it can be slow because the Hessian matrix changes at every iteration. In real-time applications, accuracy is usually a secondary concern to visual plausibility. To remedy these problems and provide an

integration technique suitable for real-time applications, Projective Dynamics, introduced by [27], provides a fast method for solving backward Euler which trades accuracy for speed while providing stability with fixed time steps. Wang et al. [152] further accelerated PBD and PD by applying a Chebychev semi-iterative method on the GPU, and then generalized this approach to support more general materials [154]. The *Vivace* solver [55] accelerates the linear system solve in PD using a novel graph-coloring approach to achieve fast Gauss-Seidel solves on the GPU. Overby et al. [122] showed that PD can be interpreted as a special case of Alternating Direction Method of Multipliers (ADMM) [28] while [96] reformulated PD as a Quasi-Newton method. Both the ADMM and the Quasi-Newton formulations enable more general hyperelastic material models compared to the original PD constraints.

Conserved quantities. In physics, Noether's theorem says that symmetries of the Lagrangian give rise to conserved quantities, such as energy and momenta. Another type of conserved quantity is the symplectic form, which provides constraints on how the motion can change with changing initial conditions. The ideal continuous solution of the equations of motions is symplectic and energy-momentum conserving. Unfortunately, it is usually impossible to preserve both symplecticity and energy-momentum with numerical integrators with a constant time step [59]. This negative result lead to the development of two branches on numerical methods: 1) symplectic integrators [105, 66] and 2) energy-momentum methods [133, 62].

Symplectic methods. Flows of mechanical systems without dissipation (Hamiltonian systems) exactly conserve not only energy and momenta, but also the symplectic form [45]. Numerical integrators which conserve the symplectic form are called symplectic integrators. Symplectic integrators can be elegantly derived by discretizing the principle of least action [157, 66, 137]. Using backward error analysis it is possible to show that symplectic integrators exhibit good long-time energy behavior [66], in particular, the energy oscillates about its true value. However, this is true only with sufficiently small time steps; otherwise, symplectic integrators (both explicit and implicit ones) can oscillate dramatically [84, 7], or even diverge. One can attempt to correct this via energy preservation as discussed in the previous paragraph. Unfortunately, there are known theoretical limitations – fixed time step methods cannot have all three of the following properties: 1) symplecticity, 2) energy conservation, 3) momentum conservation [165]. It is possible to achieve all three

with adaptive time stepping [77]. However, as previously discussed, adaptive time stepping does not meet the needs of real-time simulations. Recently, there has been renewed interest in exponential integrators which combine analytical solutions of the linear part of the ODEs with numerical solutions for the non-linear residual [110, 108, 35]. Exponential integrators are computationally efficient, robust, and well suited for applications e.g. in molecular dynamics [109]. However, just like with other symplectic schemes, stability is not guaranteed, which is problematic in real-time simulations where there is no "second chance" to re-run the simulation in case of instability.

Enforcing energy conservation. Unstable simulations are characterized by the total energy dramatically departing from its true value. A logical way to avoid this problem is by explicitly enforcing constraints on total energy. This can be implemented either as a post-processing (projection) step after running an arbitrary integrator [85, 133, 139] or by imposing an energy-conserving constraint using Lagrange multipliers [68]. Even though enforcing energy conservation can help in some cases, there are no guarantees of improved accuracy. It has been demonstrated that enforcing energy conservation can lead to less accurate results [65]. This may seem counter-intuitive; after all, the exact continuous solutions of Hamiltonian systems exactly conserve energy, so one could expect that enforcing this constraint would lead to better results. Unfortunately, this is not always the case, because numerical schemes inevitably deviate from the true continuous solution due to discretization errors. Strictly enforcing some aspects of the ideal continuous solution is dangerous, because we may destroy other desirable properties, such as symplecticity. For example, if numerical error is concentrated at one part of the simulated object, the energy-conservation mechanism can non-physically "teleport" energy to remote parts of the object, which can lead to unrealistic motion (often manifesting itself as unnatural oscillations). Another caveat is that stability is not guaranteed with projection-type methods [133, 139] because kinetic energy cannot be decreased below zero. Lagrange multiplier approaches guarantee stability with arbitrarily large time-steps [68], but do not guarantee accuracy. Figure 3.3 shows a simple mass-spring system cloth falling under gravity with energy conservation enforced using Lagrange Multipliers [68]. Even though the energy is conserved perfectly, the simulation is not visually plausible – the cloth gets "stuck" in one position and the individual elements begin to oscillate in place. This is because Newton's

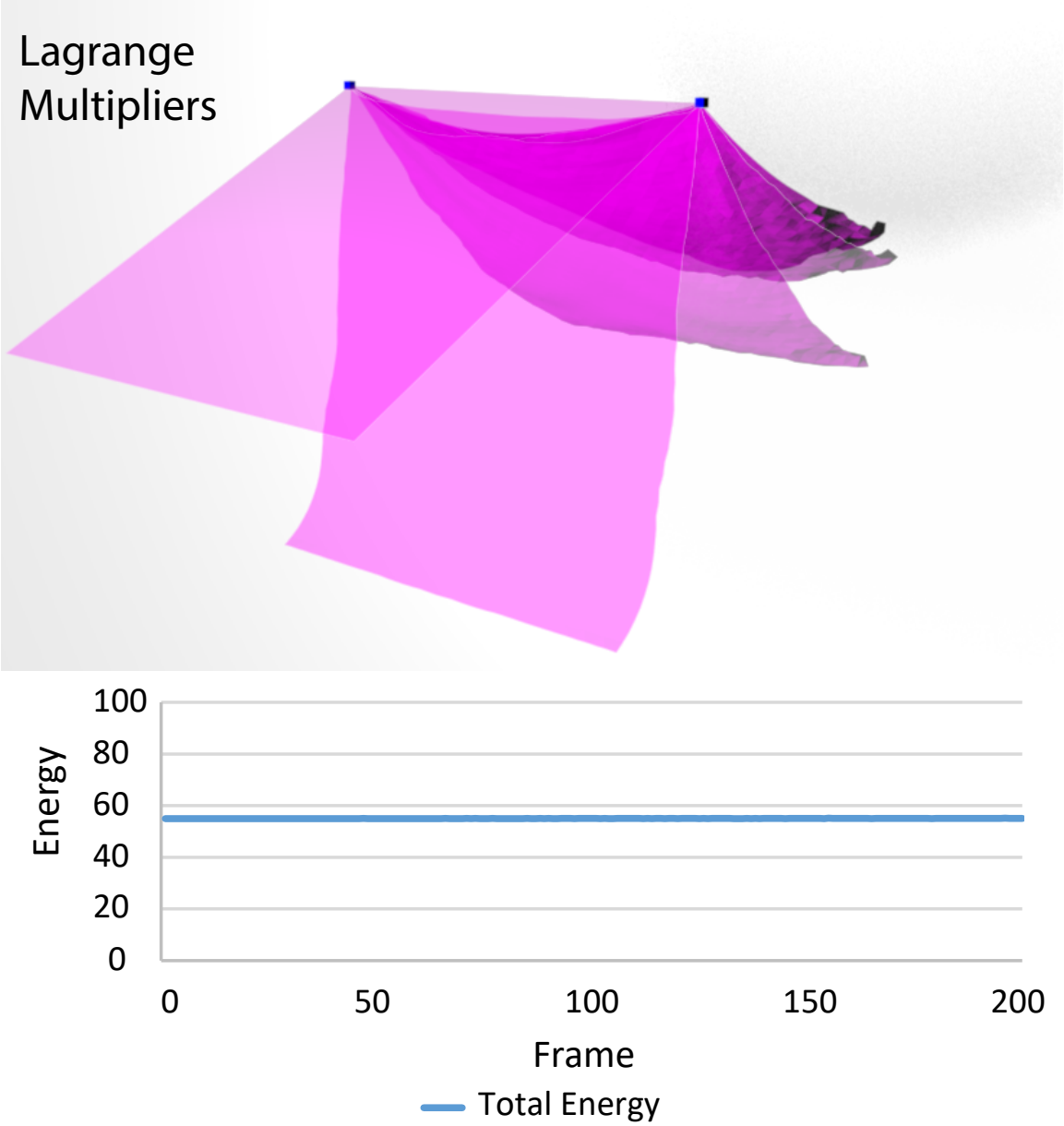

Figure 3.3: A mass-spring cloth model with gravity, with energy conservation enforced using the method of Lagrange Multipliers as described in [68] (top) and the corresponding energy graph (bottom). While the energy is perfectly conserved, the resulting motion gets "stuck," resulting in a very poor animation (see the accompanying video).

method arrives at a local minimum that satisfies the energy-conserving constraint, but does not correspond to plausible motion.

Generalized-α Methods. Good energy behavior is exhibited also by methods such as the well-known Newmark family of algorithms [118]. A more general schemes, such as the generalized alpha method, have been developed [37] and applied to computer

graphics [134]. These integrations methods provide several parameters. [76] showed that the Newmark-β method is symplectic with the common setting $\beta = 1/4, \gamma = 1/2$, even though not in the traditional sense of the canonical symplectic form. Unfortunately, similarly to the symplectic methods discussed above, with a fixed time step there is no guarantee of visually plausible results.

3.1.1 Background

3.1.1.1 Forward Euler

Forward Euler (sometimes called explicit Euler) uses only the current state to update the next state; it is an explicit method. The update rules are very simple:

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\mathbf{v}_n \tag{3.1}$$

$$\mathbf{v}_{n+1} = \mathbf{v}_n + h\mathbf{M}^{-1} f(\mathbf{x}_n) \tag{3.2}$$

where $\mathbf{x}_{n+1}$ and $\mathbf{v}_{n+1}$ are the positions and velocities at the next state, $\mathbf{x}_n$ and $\mathbf{v}_n$ are the positions and velocities at the current state, $\mathbf{M}$ is the mass matrix, h is the time step, and $f(\mathbf{x}_n)$ is the net force evaluated at $\mathbf{x}_n$. Unfortunately, this scheme is not very robust, unless the time step is very small. Even though using only the current state to estimate the next state is straightforward and intuitive, it typically results in an over-estimation of the total energy at every time step. With larger time steps, there are often significant increases of the total energy, leading to the commonly observed instabilities (or "explosions").

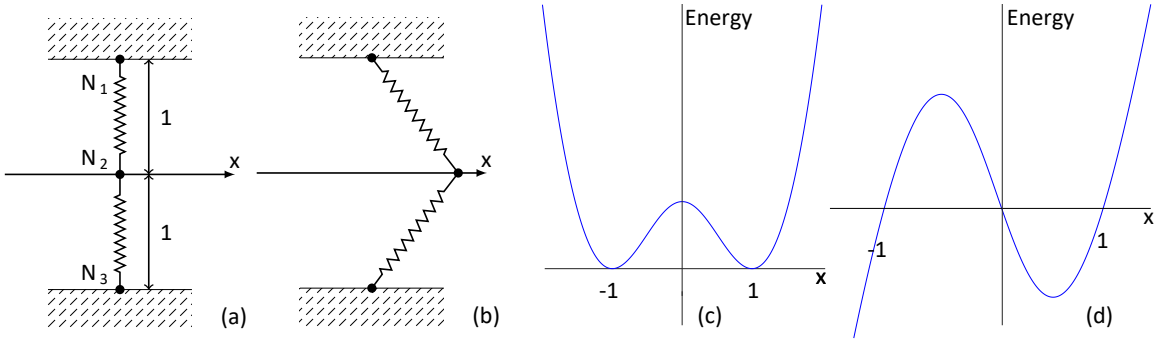

Figure 3.4: A simple two spring system where N_1, N_2, and N_3 are connected by springs. N_1 and N_3 are fixed at a distance of 1 from the **x** axis (a), and N_2 is free to move along the **x** axis (b). The resulting potential function is non-convex (c); its derivative is shown in (d).

We will present a proof for the instability of forward Euler in convex functions from a Hamiltonian point of view. Non-convex functions are discussed in Non-convex potentials

are more complex, and are discussed in Section 3.1.1.3. The Hamiltonian is the sum of the potential and the kinetic energies of the system:

$$H(\mathbf{x}, \mathbf{v}) = \frac{1}{2}||\mathbf{v}||_{\mathbf{M}}^2 + E(\mathbf{x}) \tag{3.3}$$

where $||\mathbf{v}||_{\mathbf{M}}^2 := \mathbf{v}^\mathsf{T}\mathbf{M}\mathbf{v}$ is a mass-matrix norm and $E(\mathbf{x})$ the potential energy function. To simplify our discussion, we will assume that $E(\mathbf{x})$ is defined and finite for any state $\mathbf{x}$. Because we assume the potential energy is time-invariant, in the continuous setting the Hamiltonian is exactly conserved, i.e., at any time $t > 0$, the Hamiltonian $H(\mathbf{x}(t), \mathbf{v}(t)) = H(\mathbf{x}_0, \mathbf{v}_0)$, where $\mathbf{x}_0, \mathbf{v}_0$ are the initial conditions.

This equality is no longer true in the case of numerical time integration due to discretization error. We can quantify the Hamiltonian error of a numerical scheme simply by subtracting the Hamiltonians at two consecutive steps:

$$H(\mathbf{x}_{n+1}, \mathbf{v}_{n+1}) - H(\mathbf{x}_n, \mathbf{v}_n) \tag{3.4}$$

This error depends on a specific numerical integration scheme. In this section, we focus on the forward Euler method. Using the forward Euler update rules we can write:

$$\begin{aligned} H(\mathbf{x}_{n+1}, \mathbf{v}_{n+1}) &= \frac{1}{2}||\mathbf{v}_{n+1}||_{\mathbf{M}}^2 + E(\mathbf{x}_{n+1}) \\ &= \frac{1}{2}||\mathbf{v}_n - h\mathbf{M}^{-1}\nabla E(\mathbf{x}_n)||_{\mathbf{M}}^2 + E(\mathbf{x}_{n+1}) \\ &= \frac{1}{2}||\mathbf{v}_n||_{\mathbf{M}}^2 - (\mathbf{x}_{n+1} - \mathbf{x}_n)^\mathsf{T}\nabla E(\mathbf{x}_n) \\ &+ \frac{1}{2}h^2||\mathbf{M}^{-1}\nabla E(\mathbf{x}_n)||_{\mathbf{M}}^2 + E(\mathbf{x}_{n+1}) \end{aligned} \tag{3.5}$$

Plugging this expression into Eq. 3.4 results in:

$$\begin{aligned} H(\mathbf{x}_{n+1},&\mathbf{v}_{n+1}) - H(\mathbf{x}_n, \mathbf{v}_n) = \\ &E(\mathbf{x}_{n+1}) - E(\mathbf{x}_n) - (\mathbf{x}_{n+1} - \mathbf{x}_n)^\mathsf{T}\nabla E(\mathbf{x}_n) \\ &+ \frac{1}{2}h^2||\mathbf{M}^{-1}\nabla E(\mathbf{x}_n)||_{\mathbf{M}}^2 \end{aligned} \tag{3.6}$$

This formula leads to an interesting fact if we assume the potential function E is convex (most practical potential functions are not convex, however, it is insightful to first study the case of convex E). With convex E, first-order convexity conditions (Section 3.1.3 in [29]) imply that $E(\mathbf{x}_{n+1}) - E(\mathbf{x}_n) - (\mathbf{x}_{n+1} - \mathbf{x}_n)^\mathsf{T}\nabla E(\mathbf{x}_n) \geq 0$ for any $\mathbf{x}_n$ and $\mathbf{x}_{n+1}$. Because the

term $h^2||\mathbf{M}^{-1}\nabla E(\mathbf{x}_n)||^2_{\mathbf{M}}$ is always non-negative (as the mass-matrix $\mathbf{M}$ is positive definite), we have just proven that with forward Euler, the Hamiltonian is weakly increasing.

This shows why the forward Euler method is highly unstable for convex potential functions. The Hamiltonian cannot decrease, therefore, in the best case scenario, it can remain constant. This is the expected behavior as $h \to 0$ when the discrete approximation converges to the continuous solution (where the Hamiltonian is indeed constant). However, with larger time steps h we often observe significant increases of the Hamiltonian and the Hamiltonian quickly approaches infinity, indicating instability. This behavior is often empirically observed also with common non-convex potential energy functions. However, with non-convex potentials it is possible to find examples where forward Euler actually *decreases* the Hamiltonian, i.e., $H(\mathbf{x}_{n+1}, \mathbf{v}_{n+1}) < H(\mathbf{x}_n, \mathbf{v}_n)$.

3.1.1.2 Backward Euler

While forward Euler uses the current state to compute the next state, backward Euler uses the next state, $\mathbf{x}_{n+1}$ and $\mathbf{v}_{n+1}$. Since we no longer have closed-form formulas for calculating the next state, we have to solve a system of (typically non-linear) equations to obtain $\mathbf{x}_{n+1}$ and $\mathbf{v}_{n+1}$:

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\mathbf{v}_{n+1} \tag{3.7}$$

$$\mathbf{v}_{n+1} = \mathbf{v}_n + h\mathbf{M}^{-1} f(\mathbf{x}_{n+1}) \tag{3.8}$$

This is more computationally intensive than explicit methods, but the advantage is that the usual CFL (Courant-Friedrichs-Lewy) limitations no longer apply. Simply put, explicit methods can propagate forces only to the immediately adjacent nodes. E.g, when simulating a cloth picked up at one corner, the external contact forces propagate very slowly throughout the entire system. Implicit methods can achieve such propagation during *one step*. The use of backward Euler in physics-based animation has been popularized by Baraff and Witkin [9] who demonstrated its superior behavior compared to forward Euler, especially with larger time steps and stiff systems. Backward Euler has the exact opposite error behavior compared to forward Euler: it underestimates the energy of the true solution. This is not always a problem because some amount of dissipation can in fact improve the visually plausibility of the motion. Unfortunately, the amount of numerical dissipation of backward Euler is not explicitly controllable by the user and instead indirectly depends on

resolution, time step, and stiffness. Backward Euler also results in loss angular momentum. Higher-order backward methods such as BDF2 produce more accurate solutions, but uncontrolled numerical dissipation is still present. Asynchronous timestepping methods such as [164] also mitigate the numerical damping of backward Euler, but at the cost of introducing variable computing demands. This is not desirable in real-time simulations where each frame should ideally take the same amount of computing resources.

Backward Euler is known to be very stable, but without much understanding as to why. While proofs exist for the unconditional stability of backward Euler for linear ODE's and basic stability analysis has been done for some non-linear ODE's [41, 87], we are not aware of any general proofs in the non-linear case, such as for the deformation energies commonly used in computer animation. Note that theoretical analysis assumes exact solutions of Eq. 3.7 and Eq. 3.8. Instability can be also introduced by inaccurate numerical solvers (which compute only approximate solution of Eq. 3.7 and Eq. 3.8), such as using only one iteration of Newton's method). These numerics-induced instabilities have been observed already by Baraff and Witkin [9] who proposed a solution by adapting the time step.

"Forward Euler takes no notice of wildly changing derivatives, and proceeds forward quite blindly. Backward Euler, however, forces one to find an output state whose derivative at least points back to where you came from, imparting, essentially, an additional layer of consistency (or sanity-checking, if you will)." (footnote 4, [9]). This remark provides beautiful intuition why backward Euler is more stable than forward Euler. We formalize this intuition by providing a formal stability proof of backward Euler for convex potential functions. With forward Euler, instability is characterized by increasing the Hamiltonian above all bounds. There are various interpretations of the term "stability" in the literature. We will say that simulation is stable if there is a constant $C > 0$ such that $H(\mathbf{x}_n, \mathbf{v}_n) \leq C$ for all $n = 0, 1, 2, \ldots$. The constant C can depend on the initial conditions and the parameters of the system, but cannot depend on n. In other words, regardless of how many time steps we compute, the Hamiltonian can never increase above C – it must remain bounded. We also assume that our potential $E(\mathbf{x})$ is bounded from below. This is trivially satisfied for all elastic energies. If we use the standard linear gravity potential, we assume that there is a ground plane below which the object cannot fall.

Unlike forward Euler, backward Euler does not provide explicit formulas to compute

the next step $(\mathbf{x}_{n+1}, \mathbf{v}_{n+1})$; instead, the next step is given implicitly by a system of (typically non-linear) equations. Nevertheless, we can still use a similar analysis as in Section 3.1.1.1 – the main trick being in performing the analysis backwards in time. We start by applying the backward Euler rules (Eq. 3.7 and Eq. 3.8) to expand:

$$\begin{aligned} H(\mathbf{x}_n, \mathbf{v}_n) &= \frac{1}{2}||\mathbf{v}_{n+1} + h\mathbf{M}^{-1}\nabla E(\mathbf{x}_{n+1})||^2_{\mathbf{M}} + E(\mathbf{x}_n) \\ &= \frac{1}{2}||\mathbf{v}_{n+1}||^2_{\mathbf{M}} + (\mathbf{x}_{n+1} - \mathbf{x}_n)^{\mathsf{T}}\nabla E(\mathbf{x}_{n+1}) \\ &+ \frac{1}{2}h^2||\mathbf{M}^{-1}\nabla E(\mathbf{x}_{n+1})||^2_{\mathbf{M}} + E(\mathbf{x}_n) \end{aligned} \tag{3.9}$$

Subtracting the Hamiltonian of the next step yields:

$$\begin{aligned} H(\mathbf{x}_n,&\mathbf{v}_n) - H(\mathbf{x}_{n+1}, \mathbf{v}_{n+1}) = \\ &E(\mathbf{x}_n) - E(\mathbf{x}_{n+1}) + (\mathbf{x}_{n+1} - \mathbf{x}_n)^{\mathsf{T}}\nabla E(\mathbf{x}_{n+1}) \\ &+ \frac{1}{2}h^2||\mathbf{M}^{-1}\nabla E(\mathbf{x}_{n+1})||^2_{\mathbf{M}} \end{aligned} \tag{3.10}$$

If we assume the potential E is convex, the first-order convexity conditions (Section 3.1.3 in [29]) imply that

$$E(\mathbf{x}_n) - E(\mathbf{x}_{n+1}) + (\mathbf{x}_{n+1} - \mathbf{x}_n)^{\mathsf{T}}\nabla E(\mathbf{x}_{n+1}) \geq 0 \tag{3.11}$$

for any $\mathbf{x}_n$ and $\mathbf{x}_{n+1}$. The term $h^2||\mathbf{M}^{-1}\nabla E(\mathbf{x}_{n+1})||^2_{\mathbf{M}}$ is non-negative because $h > 0$ and $\mathbf{M}$ is a positive-definite mass matrix. Therefore, with convex E, we can conclude that $H(\mathbf{x}_n, \mathbf{v}_n) \geq H(\mathbf{x}_{n+1}, \mathbf{v}_{n+1})$. This means that we can define the upper bound C simply as $C := H(\mathbf{x}_0, \mathbf{v}_0)$, in other words, the Hamiltonian never rises above its value specified by the initial positions and velocities. With $h \to 0$, we converge to the continuous case where the Hamiltonian is conserved. In real-time simulations, it is common to use relatively large h, in which case there can be relatively large decreases of the Hamiltonian, corresponding to the numerical dissipation of backward Euler.

With non-convex potentials, this analysis does not hold and it is possible to find counter-examples where $H(\mathbf{x}_n, \mathbf{v}_n) > H(\mathbf{x}_0, \mathbf{v}_0)$.

3.1.1.3 Backward Euler Stability for Non-Convex Potential Functions

Many potential functions used in physics-based animation are not convex – even simple mass-spring systems. To our knowledge, analysis of backward Euler's behavior with

non-convex potentials is an open problem. In this Appendix, we merely scratch the surface by analyzing a simple toy example of a non-convex potential. In this didactic case, we show that key result still holds, i.e., with backward Euler there exists an upper bound on the Hamiltonian $H(\mathbf{x}_n, \mathbf{v}_n)$. However, the analysis is more complicated than in the convex case studied in Appendix B where the Hamiltonian is weakly decreasing. In the non-convex case, the Hamiltonian can temporarily increase and we need to show that these increases are bounded.

We will analyze a very simple mass-spring system, consisting of three vertices connected by two springs, each of which has rest length $\sqrt{2}$ (seeFigure 3.4). In this system the two outside nodes ($N1, N3$) are fixed, and the middle node ($N2$) is free to move along the x axis. The state $x = 0$ corresponds to the situation in Figure 3.4(a), where both springs are compressed to length 1. Figure 3.4(b) depicts the case of $x = 1$, where both springs are at their rest length, i.e., the potential energy is zero. Let us derive a formula of the potential as a function of the free variable $x \in \mathbb{R}$. The length of each spring is, according to the Pythagorean theorem, $\sqrt{x^2+1}$. Plugging this into Hooke's law and summing the two springs, we obtain

$$E_{\text{test}}(x) = k\left(\sqrt{x^2+1} - \sqrt{2}\right)^2 \tag{3.12}$$

where $k > 0$ is the spring stiffness, which we assume is the same for both springs. The potential of our test system is graphed in Figure 3.4(c). We can immediately notice the potential is non-convex, with two local minima at -1 and 1 and a local maximum at 0. Let us also compute the derivative of this potential function:

$$E'_{\text{test}}(x) = 2k\left(\sqrt{x^2+1} - \sqrt{2}\right)\frac{x}{\sqrt{x^2+1}} \tag{3.13}$$

and graph it in Figure 3.4(d).

We can use the potential function E_{test} to construct an example where a backward Euler step actually *increases* energy, providing a counter-example that the approach from Appendix B does not trivially generalize to non-convex potentials. To construct such example, we can pick any starting point $x_0 \in (0, 1)$, and set the initial velocity $v_0 = -x_0/h$ (where h is the time step). In this case, the backward Euler rules (Eq. 3.7 and Eq. 3.8) are satisfied with $x_1 = 0, v_1 = v_0$, because $E'_{\text{test}}(0) = 0$. In this case the kinetic energy remains

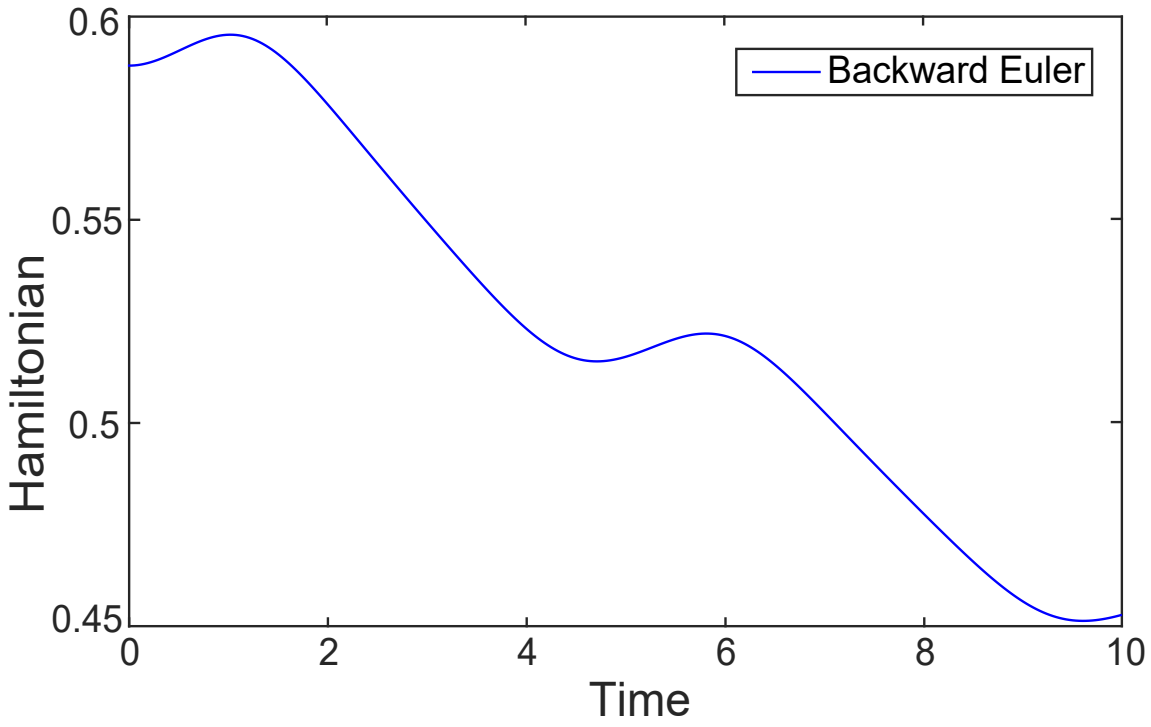

Figure 3.5: The Hamiltonian for a simulation of our simple test system from Figure 3.4 computed using backward Euler. The Hamiltonian can occasionally increase despite the overall dissipative trend.

the same, because $v_1 = v_0$, but the potential energy increases because $E_{\text{total}}(0) > E_{\text{total}}(x_0)$ (in fact we could even pick x_0 slightly larger than 1 and this inequality would still hold). In other words, in this case the non-convexity of E_{test} caused backward Euler to *increase* the Hamiltonian.

Numerical simulation of our simple test system reveals that backward Euler can consistently increase the Hamiltonian over several time steps, as shown in Figure 3.5. The increases are visible as little bumps in the otherwise generally decreasing Hamiltonian. This means that we cannot hope to prove the Hamiltonian will be weakly decreasing with non-convex potentials. However, we can still hope to prove stability, if we can show that the occasional Hamiltonian increases due to potential non-convexity are in fact bounded. The graph in Figure 3.5 suggests that this indeed may be the case. In the following, we prove this for the case of our simple potential E_{test}. The key idea of the proof is the fact that the non-convexity of E_{test} is localized to the interval $[-1, 1]$. In the following three lemmas we consider three different cases depending on which intervals are x_n and x_{n+1} in.

Lemma 1. *Let $x_n, x_{n+1} \in \mathbb{R}$ be two consecutive time steps of backward Euler applied to E_{test}. If $|x_{n+1}| > 1$, then $H(x_n, v_n) \geq H(x_{n+1}, v_{n+1})$.*

Proof. First we will define a function $G(x)$ such that

$$G(x) = \begin{cases} 0 & x \in [-1, 1] \\ E_{\text{test}}(x) & \text{otherwise} \end{cases} \tag{3.14}$$

Because $E_{\text{test}}(x)$ is a non-negative function, $G(x) \leq E_{\text{test}}(x)\ \forall x \in \mathbb{R}$. Because $G(x)$ is convex and differentiable (note that $E'_{\text{test}}(\pm 1) = 0$), we can apply the first-order convexity condition to get the inequality:

$$G(x_n) \geq G(x_{n+1}) + (x_n - x_{n+1})G'(x_{n+1}) \tag{3.15}$$

Because we assumed that $|x_{n+1}| > 1$, we can write $G(x_{n+1}) = E_{\text{test}}(x_{n+1})$ and $G'(x_{n+1}) = E'_{\text{test}}(x_{n+1})$. Putting these facts together yields the following inequality for E_{test}:

$$E_{\text{test}}(x_n) \geq G(x_n) \geq E_{\text{test}}(x_{n+1}) + (x_n - x_{n+1})E'_{\text{test}}(x_{n+1}) \tag{3.16}$$

Plugging this inequality into Eq. 3.10 shows that $H(x_n, v_n) \geq H(x_{n+1}, v_{n+1})$ just like in the convex case. □

Lemma 2. *There is a constant $\beta > 1$ such that for any two consecutive time steps $x_n, x_{n+1} \in \mathbb{R}$ of backward Euler applied to E_{test}, it is true that if $|x_n| \geq \beta$, $|x_{n+1}| \leq 1$ then $H(x_n, v_n) \geq H(x_{n+1}, v_{n+1})$.*

Proof. Lemma 1 showed that if $|x_{n+1}| > 1$, then $H(x_n, v_n) \geq H(x_{n+1}, v_{n+1})$ regardless of the value of x_n. However, if $x_{n+1} \in [-1, 1]$, there are situations when $H(x_n, v_n) \leq H(x_{n+1}, v_{n+1})$. Now we will show that this cannot happen if $|x_n| \geq \beta$. First, we will assume that $x_n \geq \beta$ (the case of $\mathbf{x}_n \leq -\beta$ is analogous) and show that:

$$E_{\text{test}}(x_n) \geq E_{\text{test}}(x_{n+1}) + (x_n - x_{n+1})E'_{test}(x_{n+1}) \tag{3.17}$$

This inequality leads to $H(x_n, v_n) \geq H(x_{n+1}, v_{n+1})$ as in the convex case (Eq. 3.10). Our first task is to find a suitable β. Let us consider the line

$$l(x) = E_{\text{test}}(0) + (x + 1)E'_{\text{test}}(x_{\max}) \tag{3.18}$$

The value $x_{\max}$ is defined as $x_{\max} := \arg\max_{x \in [-1,1]} E'(x)$, i.e., the maximal derivative on the interval $[-1, 1]$. $E_{\text{test}}(0)$ is the maximum value of the potential on the interval $[-1, 1]$.

We will then define β as the intersection point between $l(x)$ and $E_{\text{test}}(x)$ with $x > 0$, see Figure 3.6. In other words, we pick β as the positive solution of:

$$E_{\text{test}}(\beta) = E_{\text{test}}(0) + (\beta + 1)E'_{\text{test}}(x_{\max}) \tag{3.19}$$

While it is hard to evaluate β symbolically, we can easily approximate it numerically: $\beta \approx 2.209$. An important fact is that the β depends only on the potential E_{test}, i.e., it does not depend on the states x_n, x_{n+1}.

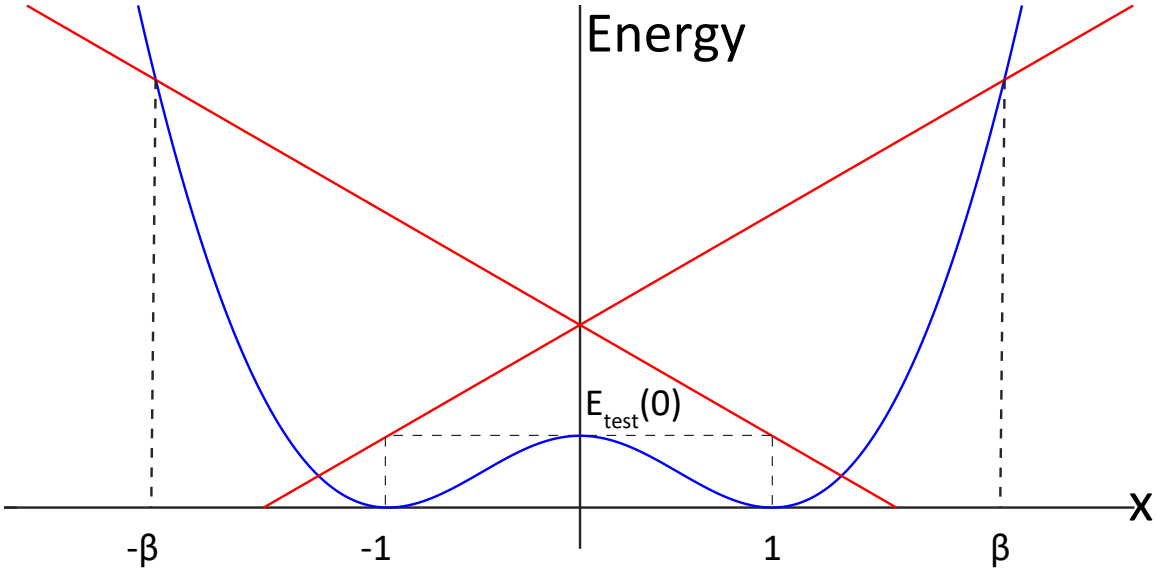

Figure 3.6: Illustration for Lemma 2: the potential function $E_{\text{test}}(x)$ (blue) and our lines $l(x)$ (red) determining the constant β.

Using the definition of the line in Eq. 3.18, we can show that the following inequality is also satisfied $\forall x_{n+1} \in [-1, 1]$:

$$E_{\text{test}}(\beta) \geq E_{\text{test}}(x_{n+1}) + (\beta - x_{n+1})E'_{\text{test}}(x_{n+1}) \tag{3.20}$$

To show this, we refer to Eq. 3.19 and observe that $E_{\text{test}}(0) \geq E_{\text{test}}(x_{n+1})$ because $E_{\text{test}}(0) \geq E_{\text{test}}(x)\ \forall x \in [-1, 1]$. The fact that $E'_{\text{test}}(x_{\max}) \geq E'_{\text{test}}(x_{n+1})$ follows from the definition of $x_{\max}$. Finally, $\beta + 1 \geq \beta - x_{n+1}$ follows from $|x_{n+1}| \leq 1$.

E_{test} is a convex function on interval $[1, \infty]$, which implies:

$$E'_{\text{test}}(\beta) \geq E'_{\text{test}}(x_{\max}) \tag{3.21}$$

because β is the intersection between $l(x)$ and $E_{\text{test}}(x)$. The convexity on $[1, \infty]$ further implies:

$$E_{\text{test}}(x_n) \geq E_{\text{test}}(\beta) + (x_n - \beta)E'_{\text{test}}(\beta) \tag{3.22}$$

due to first-order convexity conditions [29]. We can now substitute Eq. 3.20 into Eq. 3.22:

$$\begin{aligned} E_{\text{test}}(x_n) \geq E_{\text{test}}(x_{n+1}) + (\beta - x_{n+1})E'_{\text{test}}(x_{n+1}) \\ + (x_n - \beta)E'_{\text{test}}(\beta) \end{aligned} \tag{3.23}$$

Plugging in Eq. 3.21 and using the fact that $E'_{\text{test}}(x_{\max}) \geq E'_{\text{test}}(x_{n+1})$ and our assumption $x_n \geq \beta$ results, after some simplifications, in:

$$E_{\text{test}}(x_n) \geq E_{\text{test}}(x_{n+1}) + (x_n - x_{n+1})E'_{\text{test}}(x_{n+1}) \tag{3.24}$$

This is the same inequality as Eq. 3.11 and therefore the result $H(\mathbf{x}_n, \mathbf{v}_n) \geq H(\mathbf{x}_{n+1}, \mathbf{v}_{n+1})$ follows just like in the convex case (Appendix B). The proof for $x_n < -\beta$ is completely analogous due to the fact that $E_{\text{test}}(x) = E_{\text{test}}(-x)$. □

Lemma 3. *If $\beta > 1$ is as in Lemma 2, there exists a constant $H_c > 0$ such that for any two consecutive time steps $x_n, x_{n+1} \in \mathbb{R}$ of backward Euler applied to E_{test}, such that $|x_{n+1}| \leq 1, |x_n| \leq \beta$, it is true that $H(x_{n+1}, v_{n+1}) \leq H_c$, where $H_c = E_{\text{test}}(0) + \frac{m}{2}(\frac{\beta+1}{h})^2$, where m is the mass of vertex N_2 and h is the time step.*

Proof. Because $x_{n+1} \in [-1, 1]$, we immediately have a bound on the potential energy:

$$E_{\text{test}}(x_{n+1}) \leq E_{\text{test}}(0) \tag{3.25}$$

To obtain a bound on the kinetic energy, we use the update rule of backward Euler and the facts that $x_{n+1} \in [-1, 1]$ and $x_n \in [-\beta, \beta]$, which lead to:

$$|v_{n+1}| = \left| \frac{x_{n+1} - x_n}{h} \right| \leq \frac{\beta + 1}{h} \tag{3.26}$$

Combining Eq. 3.25 and Eq. 3.26 will get us a Hamiltonian bound:

$$H(x_{n+1}, v_{n+1}) = E_{\text{test}}(x_{n+1}) + \frac{m}{2}(v_{n+1})^2 \tag{3.27}$$

$$\leq E_{\text{test}}(0) + \frac{m}{2}\left(\frac{\beta + 1}{h}\right)^2 = H_c \tag{3.28}$$

□

Finally, we put the results of the previous three lemmas together in the following theorem:

Theorem 1. *If $x_0, v_0 \in \mathbb{R}$ are arbitrary initial conditions of our test problem, then the Hamiltonian at any step n of exactly solved backward Euler integration satisfies*

$$H(x_n, v_n) \leq \max(H(x_0, v_0), H_c)$$

where $H_c = E_{\text{test}}(0) + \frac{m}{2}(\frac{\beta+1}{h})^2$ as in Lemma 3.

Proof. The proof is by induction. The theorem is trivially satisfied for $n = 0$. Moving from n to $n + 1$, there are several possibilities. If $|x_{n+1}| > 1$, Lemma 1 applies and shows that the Hamiltonian must weakly decrease. If $|x_{n+1}| \leq 1$ and $|x_n| \geq \beta$, Lemma 2 applies and also asserts the Hamiltonian must weakly decrease. Finally, if $|x_{n+1}| \leq 1$ and $|x_n| < \beta$, the Hamiltonian may increase, but Lemma 3 shows that it cannot increase arbitrarily much, because $H(x_{n+1}, v_{n+1}) \leq H_c$. □

We have shown that for our simple but non-convex potential E_{test}, exactly solved backward Euler is stable. A logical question is whether a similar result would hold also for arbitrary mass-spring systems. Unfortunately, the situation becomes more complicated, because our bounds would have to be extended to the multi-variate case. For example, the two lines in Figure 3.6 could be replaced by a translated convex cone. However, there is a complication, because in general mass-spring systems, some springs can be contracted, while others can be extended. Even a single contracted spring can, in theory, introduce non-convexities. The backward Euler stability proof for general mass-spring systems will therefore be more complicated and we defer it to future work.

Implicit Midpoint. Intuitively, it is obvious that both forward and backward Euler are biased: the former relies entirely on the current state, while the latter relies entirely on the next state. Implicit midpoint can be seen as a compromise between the two: instead of using just the current or just the next state, implicit midpoint uses their average:

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \frac{h}{2}(\mathbf{v}_{n+1} + \mathbf{v}_n) \tag{3.29}$$

$$\mathbf{v}_{n+1} = \mathbf{v}_n + h\mathbf{M}^{-1} f\left(\frac{\mathbf{x}_{n+1} + \mathbf{x}_n}{2}\right) \tag{3.30}$$

This integrator has very nice properties, namely it is symplectic and momentum-preserving. A side effect of the preservation of the canonical symplectic form is good energy-preserving

behavior [66]. However, this does not mean that the total energy has to be close to its exact value. Even in very simple simulations it is not hard to obtain extreme energy oscillations. For example, the simulation in Figure 3.7 uses the potential function:

$$E_{\text{simpleSpring}}(\mathbf{x}) = \frac{1}{2}k(||\mathbf{x}||^2 - 1)^2 \tag{3.31}$$

which is a squared version of a classical Hookean spring of length 1 m with one endpoint fixed at the origin. For our experiment we used stiffness $k = 10^5$ N/m, a mass of 0.5 kg, two time steps $h_{\text{small}} = 0.00033$ s (Figure 3.7, left) and $h_{\text{large}} = 0.033$ s (Figure 3.7, right), an initial velocity of $(0, 10, 0)$ m/s, and initial length of 1.5 (corresponding to the spring being pre-stretched). Figure 3.7 shows that with h_{large}, the simulation does not behave plausibly even during the first few time steps – contrary to our expectations, the rotating spring does not return to its rest length (1 m) but instead further extends beyond the initial length of 1.5 m (indicated by the red circle). If we decrease the time step to h_{small}, this behavior disappears and the revolving spring contracts as expected, staying within the red circle.

What happened with the larger time step $h = 0.033$ s? We plot the total energy of this simple system in Figure 3.8. We can see that the simulation reaches incredibly large energy levels – the system starts out at an energy level of around 25,000 J but quickly proceeds to

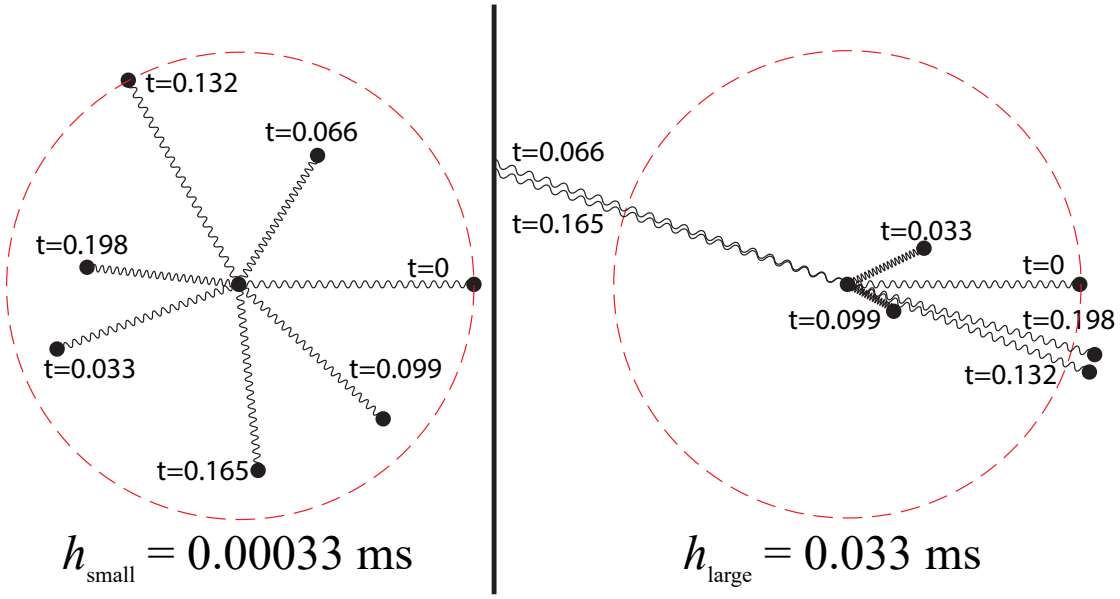

Figure 3.7: A simple spring-type potential run using implicit midpoint at a time step $h_{\text{small}} = 0.00033$ (left) and $h_{\text{large}} = 0.033$ (right). Each point represents a state at a time t. The red circle illustrates initial stretched length of the spring.

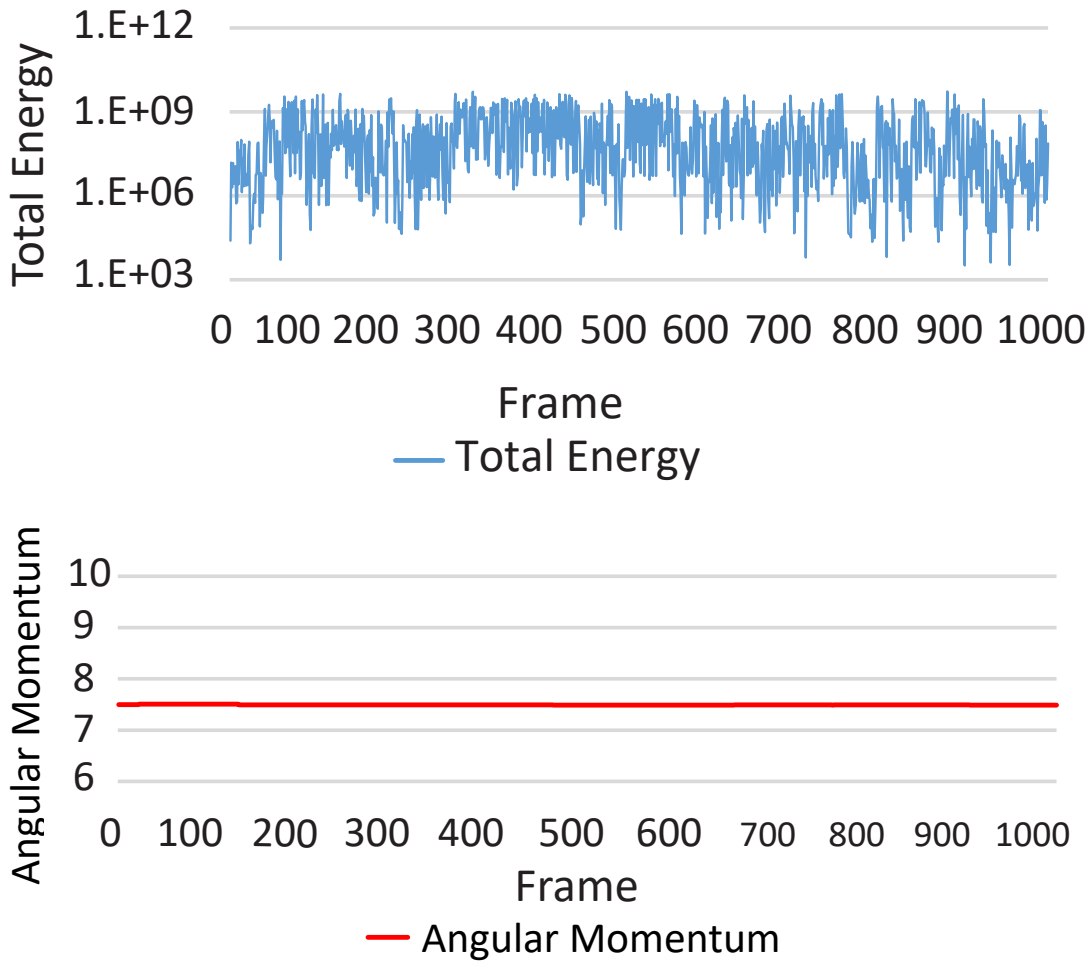

Figure 3.8: Energy (top) and angular momentum (bottom) corresponding to the simulation from Figure 3.7 with $h = 0.033$ s.

energy levels of several orders of magnitude higher, at which point the moving endpoint flies off the screen with unrealistic speed. Even though the energy is fluctuating wildly, the angular momentum is still conserved, as shown in Figure 3.8. This poor behavior of implicit midpoint can be observed even with more complex models, such as the rotating cube modeled with corotated elasticity in Figure 3.9.

A related integrator comes from using the trapezoid rule instead of the midpoint rule. This corresponds to the Newmark integrator with $\gamma = 1/2$ and $\beta = 1/4$. While this integrator can behave better than implicit midpoint, it is still prone to explosions (Figure 3.9).

As a final remark, we note that implicit midpoint can also be written as a composition of backward and forward Euler:

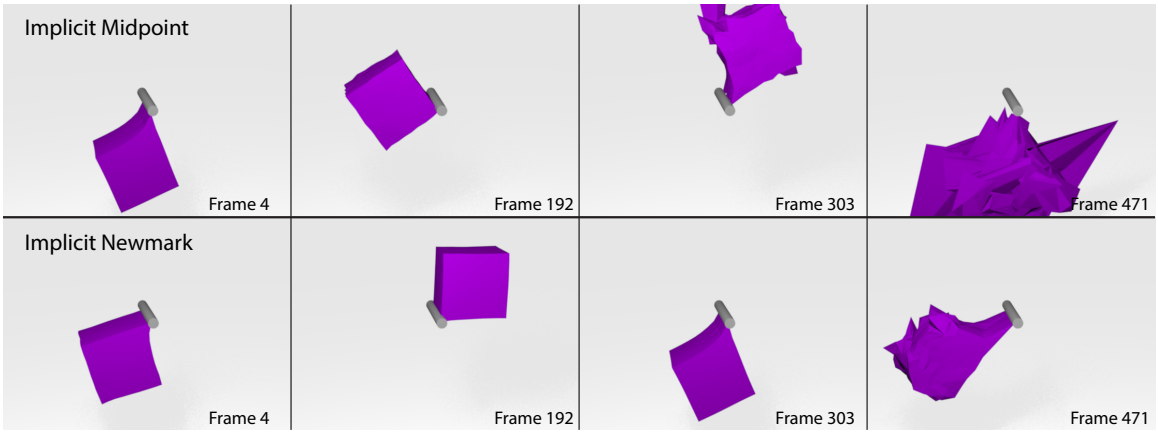

Figure 3.9: A spinning deformable cube modeled with corotated elasticity simulated with time step $h = 0.033$. After several hundred time steps, the implicit midpoint and implicit Newmark integrator start producing visually implausible results.

$$\mathbf{x}_{n+\frac{1}{2}} = \mathbf{x}_n + \frac{h}{2}\mathbf{v}_{n+\frac{1}{2}} \tag{3.32}$$

$$\mathbf{v}_{n+\frac{1}{2}} = \mathbf{v}_n + \frac{h}{2}\mathbf{M}^{-1}f(\mathbf{x}_{n+\frac{1}{2}}) \tag{3.33}$$

$$\mathbf{x}_{n+1} = \mathbf{x}_{n+\frac{1}{2}} + \frac{h}{2}\mathbf{v}_{n+\frac{1}{2}} \tag{3.34}$$

$$\mathbf{v}_{n+1} = \mathbf{v}_{n+\frac{1}{2}} + \frac{h}{2}\mathbf{M}^{-1}f(\mathbf{x}_{n+\frac{1}{2}}) \tag{3.35}$$

By substituting Eq. 3.34 into Eq. 3.32 and substituting Eq. 3.35 into Eq. 3.33, we get:

$$\mathbf{x}_{n+\frac{1}{2}} = \frac{\mathbf{x}_n + \mathbf{x}_{n+1}}{2} \tag{3.36}$$

$$\mathbf{v}_{n+\frac{1}{2}} = \frac{\mathbf{v}_n + \mathbf{v}_{n+1}}{2} \tag{3.37}$$

When we substitute these back into Eq. 3.34 and Eq. 3.35, we get the implicit Midpoint update equations Eq. 3.29 and Eq. 3.30. If we instead do a step of forward Euler first and backward Euler second, we get the trapezoidal rule.

3.1.1.4 Optimization Form for Time Integration Methods

Many popular implicit integration schemes can be expressed as special cases of the following optimization problem:

$$g(\mathbf{x}) = \frac{1}{2}||\mathbf{x} - \mathbf{y}||^2_{\mathbf{M}} + \alpha h^2 E(\beta\mathbf{x} + \mathbf{z}) \tag{3.38}$$

where h is the time step size, $||\mathbf{x}||^2_{\mathbf{M}} = \mathbf{x}^T\mathbf{M}\mathbf{x}$ stands for the mass-weighted norm and E is the potential energy of the system. Two scalars α, β and two vectors $\mathbf{y}$, $\mathbf{z}$ are time-

integrator-dependent constants. The goal is to minimize $g(\mathbf{x})$ to find the next state position $\mathbf{x}_{n+1} := \arg\min_{\mathbf{x}} g(\mathbf{x})$ (local minimum is sufficient). Once the positions $\mathbf{x}_{n+1}$ are found, we compute the velocities $\mathbf{v}_{n+1}$ explicitly based on the update rules of the specific integrator. The specific integrator-dependent constants and update rules are as follows.

For backward Euler (BDF-1), we have:

$$\begin{aligned} &\alpha = 1 \quad \beta = 1 \\ &\mathbf{y} = \mathbf{x}_n + h\mathbf{v}_n \quad \mathbf{z} = \mathbf{0} \\ &\mathbf{v}_{n+1} := \frac{1}{h}(\mathbf{x}_{n+1} - \mathbf{x}_n) \end{aligned}$$

For BDF-2, we have:

$$\begin{aligned} &\alpha = \frac{4}{9} \quad \beta = 1 \\ &\mathbf{y} = \frac{4\mathbf{x}_n - \mathbf{x}_{n-1}}{3} + h\frac{8\mathbf{v}_n - 2\mathbf{v}_{n-1}}{9} \quad \mathbf{z} = \mathbf{0} \\ &\mathbf{v}_{n+1} := \frac{3}{2h}(\mathbf{x}_{n+1} - \frac{4\mathbf{x}_n - \mathbf{x}_{n-1}}{3}) \end{aligned}$$

For implicit midpoint, we have:

$$\begin{aligned} &\alpha = 1 \quad \beta = \frac{1}{2} \\ &\mathbf{y} = \mathbf{x}_n + h\mathbf{v}_n \quad \mathbf{z} = \frac{\mathbf{x}_n}{2} \\ &\mathbf{v}_{n+1} := \frac{2}{h}(\mathbf{x}_{n+1} - \mathbf{x}_n) - \mathbf{v}_n \end{aligned}$$

Traditionally, the solution to this minimization problem is found by using Newton's Method with a line search [29]. While this can produce an accurate solution, it can be computationally intensive as every iteration requires building a hessian matrix and performing a sparse linear solve, which is usually a computational bottleneck. Projective Dynamics [27] accelerates this solve by applying a local/global approach and using a constant approximation for the hessian matrix, allowing us to pre-compute the factorization.

Implicit Midpoint Local/Global Derivation. Projective Dynamics provides a computational advantage over Newton's method by solving backward Euler optimization problem using a local/global method, which allows us to pre-compute sparse Cholesky factors of the system matrix. In this section, we will provide a brief overview of Projective Dynamics, and then present a similar local/global optimization strategy for solving the implicit midpoint optimization problem.

Projective Dynamics approximately solves the minimization formulation of backward Euler:

$$g(\mathbf{x}) = \frac{1}{2}(\mathbf{x}-\mathbf{y})^T\mathbf{M}(\mathbf{x}-\mathbf{y}) + h^2E(\mathbf{x}) \tag{3.39}$$

The key idea behind Projective Dynamics is constraint *projection*. First, we introduce an auxiliary "projection" variable $\mathbf{p}$. In order to take advantage of this auxiliary variable we need to define a special energy $E(\mathbf{x})$ for each element i:

$$E_i(\mathbf{x}) = \min_{\mathbf{p}_i \in \mathcal{M}} E(\mathbf{x}, \mathbf{S}_i\mathbf{p}), E(\mathbf{x}, \mathbf{S}_i\mathbf{p}) = ||\mathbf{G}_i\mathbf{x} - \mathbf{S}_i\mathbf{p}||_F^2 \tag{3.40}$$

Where $\mathbf{S}_i$ a selection matrix, $\mathbf{p}$ is a stacked vector of the projection variables that project onto the manifold $\mathcal{M}$, and $\mathbf{G}_i$ is a discrete differential operator. This Projective Dynamics energy can be used to express many common potentials, although not all potentials can be written in this form. For example, to model a mass-spring system, as [95] did, we use a sphere as our constraint manifold $\mathcal{M}$, and $\mathbf{G}_i \in \mathbb{R}^{3\times 3n}$ is an operator that subtracts two endpoints. $\mathbf{p} \in \mathbb{R}^{3s\times 1}$, then, is a variable that projects the current state onto the sphere $\mathcal{M}$, and the selection matrix has dimensions $\mathbf{S}_i \in \mathbb{R}^{3\times 3s}$ (where n is the number of vertices and s is the number of springs). To get a rigid-as-possible model [34], we instead use the manifold $\mathcal{M} = SO(3)$ and the deformation gradient operator [131] for $\mathbf{G}$.

The total potential $E(\mathbf{x})$ is simply a weighted sum of the element-wise potentials in Eq. 3.40, i.e., $E(\mathbf{x}) = \sum_i w_i E_i(\mathbf{x})$. The weights w_i are nonzero positive values that are typically the product of the rest-pose volume and stiffness of the element i. We can now rewrite the optimization form Eq. 3.39 using a few extra variables:

$$g(\mathbf{x}, \mathbf{p}) = \frac{1}{2}(\mathbf{x}-\mathbf{y})^\mathsf{T}\mathbf{M}(\mathbf{x}-\mathbf{y}) + h^2\left(\frac{1}{2}\mathbf{x}^\mathsf{T}\mathbf{L}\mathbf{x} - \mathbf{x}^\mathsf{T}\mathbf{J}\mathbf{p}\right) \tag{3.41}$$

$\mathbf{L} := \left(\sum w_i\mathbf{G}_i\mathbf{G}_i^\mathsf{T}\right)\otimes\mathbf{I}_3$ and $\mathbf{J} := \left(\sum w_i\mathbf{G}_i^\mathsf{T}\mathbf{S}_i\right)\otimes\mathbf{I}_3$, $\mathbf{I}_3 \in \mathbb{R}^{3\times 3}$ is the identity matrix, and $\otimes$ is the Kronecker product. The optimization is split up into a local and global step. In the local step, the positions $\mathbf{x}$ are assumed to be fixed and the projection variables $\mathbf{p}$ are solved for by being projected onto the constraint manifold (e.g., a sphere for simple springs or $SO(3)$ for corotated elasticity). The global step fixes the resulting projections $\mathbf{p}$ and solves a linear

system to compute the new state. Taking $\nabla g(\mathbf{x}, \mathbf{p}) = 0$ and rearranging the terms, we get the linear system:

$$(\mathbf{M} + h^2\mathbf{L})\mathbf{x} = (h^2\mathbf{J}\mathbf{p} + \mathbf{M}\mathbf{y}) \tag{3.42}$$

which is then solved to get the new $\mathbf{x}$ values. The state matrix $\mathbf{M} + h^2\mathbf{L}$ does not depend on $\mathbf{x}$, so it can be pre-factorized and reused. This pre-factorization is where the speedup compared to Newton's method comes from.

We need to make a slight modification to Eq. 3.41 in order to use the local/global process for implicit midpoint. The potential energy needs to be evaluated at $\frac{\mathbf{x}+\mathbf{x}_n}{2}$ due to the update rules for implicit midpoint. This changes the objective to:

$$\begin{aligned} g(\mathbf{x}, \mathbf{p}) = &\frac{1}{2}(\mathbf{x} - \mathbf{y})^\mathsf{T}\mathbf{M}(\mathbf{x} - \mathbf{y}) \\ &+h^2\left(\frac{1}{2}\left(\frac{\mathbf{x}+\mathbf{x}_n}{2}\right)^\mathsf{T}\mathbf{L}\left(\frac{\mathbf{x}+\mathbf{x}_n}{2}\right) - \left(\frac{\mathbf{x}+\mathbf{x}_n}{2}\right)^\mathsf{T}\mathbf{J}\mathbf{p}\right) \end{aligned} \tag{3.43}$$

3.2 Stabilizing Integrators for Real-Time Physics

This work was first published in ACM Transaction on Graphics. (Vol. 37 Issue 1 2018) and presented at ACM SIGGRAPH 2018. In this work, we propose a novel time integration method for real-time simulation of deformable objects which inherits the stability of backward Euler but does not suffer from artificial numerical damping. We achieve this by starting with the implicit midpoint integrator. Implicit midpoint does not introduce artificial damping but, unfortunately, the energy oscillations can be dramatic and produce visually catastrophic results. We observe that these oscillations can be tamed by adding only a small contribution of backward Euler integration. Similarly, in cases where the initial implicit midpoint solve underestimates the total energy, we can correct this by blending with forward Euler. The key is to determine the right amount of blending between implicit midpoint and forward/backward Euler. We accomplish this by tracking the total energy of our simulated system, taking account of energy-changing events such as damping (energy dissipation) or forcing (energy injection). When we detect that implicit midpoint overshoots the total energy (indicating that future time steps could develop instabilities), we calculate which blend between implicit midpoint and backward Euler will result in

an "as-energy-conserving-as-possible" state. Usually, only a very small contribution of backward Euler is sufficient to stabilize the simulation while introducing only a minimal amount of numerical damping. Similarly, we use forward Euler blending when implicit midpoint loses energy. This is not critical for guaranteeing stability but it helps to improve the visual quality of the resulting motion.

To obtain accurate solutions of backward Euler, we have to iterate Newton's method until convergence, which is impractically slow in real-time simulations. Therefore, a special class of quasi-Newton methods known as Projective Dynamics [27, 96], based on local/global optimization, has been developed as a computationally efficient alternative to Newton's method. Projective Dynamics is derived from backward Euler integration and therefore inherits the undesired numerical damping. In this paper, we also present a local/global acceleration for implicit midpoint and its stabilization via forward/backward Euler in order to facilitate high-quality real-time animations. Even though local/global solves are typically not iterated until convergence, our experiments demonstrate that we still obtain stable simulations while avoiding numerical damping. The additional computing overhead over standard Projective Dynamics is small, typically on the order of 20%-30% of extra computing time.

3.2.1 Method

Overview. In this section we propose a stable integration scheme. The key idea of the method is to start with implicit midpoint, an implicit symplectic integrator, and take advantage of the damping properties of backward Euler to damp out extra energy that is introduced. First, we calculate a time step according to the implicit midpoint rule, which conserves momenta and the symplectic form and therefore provides an excellent "initial guess". The main problem of implicit midpoint is that wide oscillations of the total energy can occur, departing dramatically from a visually plausible solution. In the second step, we therefore calculate the energy increase/decrease due to implicit midpoint (with a Hamiltonian system, the exact solution conserves the total energy). If the energy erroneously increased, we correct this by computing a backward Euler step, which dissipates energy. A detail which will be important later is that the backward Euler solution process uses the state computed by implicit midpoint as an initial guess. We then solve for the

optimal linear blending parameter which will bring the total energy as close as possible to its original value. Similarly, if we detect that implicit midpoint erroneously decreased energy, we perform analogous blending with the forward Euler step, taking advantage of its energy-injection property. Our experiments reveal that each time step typically only required a small amount of blending, meaning that we do not depart too far from the symplectic and momentum-conserving solution given by implicit midpoint.

Energy Error. We choose which blending to use by observing the total energy error:

$$e(\mathbf{x}, \mathbf{v}) = E(\mathbf{x}) + K(\mathbf{v}) - H_{\text{total}} \tag{3.44}$$

where $K(\mathbf{v}) = \frac{1}{2}\mathbf{v}^{\mathsf{T}}\mathbf{M}\mathbf{v}$ is the kinetic energy, $E(\mathbf{x})$ is the potential energy, and H_{total} is the initial total energy of the system, specified by the initial conditions $(\mathbf{x}_0, \mathbf{v}_0)$. If $e(\mathbf{x}^{\text{IM}}_{n+1}, \mathbf{v}^{\text{IM}}_{n+1}) > 0$ it means that implicit midpoint erroneously increased the total energy, in which case we take advantage of the numerical dissipation properties of backward Euler in order to reduce e. When $e(\mathbf{x}^{\text{IM}}_{n+1}, \mathbf{v}^{\text{IM}}_{n+1}) < 0$, it means that implicit midpoint incorrectly decreased the total energy, which we correct using forward Euler.

Blending. The way we stabilize energy overshoots is by calculating a blend between the implicit midpoint result (which resulted in erroneous energy injection) and the backward Euler result (which may potentially remove too much energy). The results of two integrators are blended linearly, using a blending parameter $\alpha \in [0, 1]$:

$$\mathbf{x}_{n+1} = (1 - \alpha)\mathbf{x}^{\text{IM}}_{n+1} + \alpha\mathbf{x}^{\text{BE}}_{n+1} \tag{3.45}$$

$$\mathbf{v}_{n+1} = (1 - \alpha)\mathbf{v}^{\text{IM}}_{n+1} + \alpha\mathbf{v}^{\text{BE}}_{n+1} \tag{3.46}$$

A higher α would yield a solution closer to backward Euler. At the extreme, $\alpha = 1$ would result in a full step of backward Euler. Assuming that backward Euler is stable, this implies stability of our method. Even though in Appendix D we prove stability of backward Euler only for certain potential functions, our experiments suggest that our method is stable even for more complex potentials. Even extreme initial conditions, as shown in Figure 3.12, do not introduce instabilities in our method. Most practical simulations do not contain such large deformations and we only need to introduce a small amount of backward Euler, i.e., α is close to zero. These small modifications early on in the simulation make a large difference in the long-term stability and perceived vividness of the motion. Without these small α

modifications, the energy will gradually drift upwards over a long period of time, leading to "explosions" [48].

In an analogous fashion we handle energy under-estimates, which we correct by blending with forward Euler. Keeping the convention that higher α yields a more "damped" solution, we replace $(\mathbf{x}_{n+1}^{\text{IM}}, \mathbf{v}_{n+1}^{\text{IM}})$ with $(\mathbf{x}_{n+1}^{\text{FE}}, \mathbf{v}_{n+1}^{\text{FE}})$ and replace $(\mathbf{x}_{n+1}^{\text{BE}}, \mathbf{v}_{n+1}^{\text{BE}})$ with $(\mathbf{x}_{n+1}^{\text{IM}}, \mathbf{v}_{n+1}^{\text{IM}})$ in Eq. 3.45 and Eq. 3.46. Now, $\alpha = 0$ corresponds to a full step of forward Euler and $\alpha = 1$ corresponds to a full step of implicit midpoint. This allows us to adjust the energy in a symmetric way, compensating for both over and under-shoots.

We can reformulate $e(\mathbf{x}, \mathbf{v})$ from Eq. 3.44 as a function of α. In the case of blending with backward Euler, we substitute Eq. 3.45 and Eq. 3.46 into Eq. 3.44, obtaining:

$$\begin{aligned} e(\alpha) = E\left((1-\alpha)\mathbf{x}_{n+1}^{\text{IM}} + \alpha\mathbf{x}_{n+1}^{\text{BE}}\right) \\ + K\left((1-\alpha)\mathbf{v}_{n+1}^{\text{IM}} + \alpha\mathbf{v}_{n+1}^{\text{BE}}\right) - H_{\text{total}} \end{aligned} \tag{3.47}$$

The problem now becomes finding an α value that brings the residual energy as close as possible to zero.

This can be done using a simple binary search algorithm because $e(\alpha)$ is continuous. During our experiments, we found that $e(\alpha)$ is always monotonic, making the binary search rapidly converge to a solution.

Binary search requires a terminating condition. We base our terminating condition on the total energy at the last frame:

$$|e(\alpha)| < \epsilon H_{\text{total}}(\mathbf{x}_n, \mathbf{v}_n) \tag{3.48}$$

$\epsilon \in [0, 1]$ is a variable that controls how accurately we will conserve the energy. A small value will result in strict energy conservation, while a larger ϵ will result in a looser terminating condition. In practice, we found that a value of $\epsilon = 0.01$ (corresponding to a 1% error in the energy) produced good results.

In rare cases, Backward Euler may not decrease the energy (we discuss an example in Appendix D). If both implicit midpoint and backward Euler increase the energy, then the binary search algorithm will not be able to find a root. When this happens, we simply take the endpoint $\alpha = 1$ as a solution. In other words, we take a full step of backward Euler and the extra energy will be removed in future frames.

3.2.2 Local/Global Solver

Projective Dynamics [27] provided a fast solution to the backward Euler minimization problem by introducing an auxiliary variable and applying block coordinate decent. To accelerate our method to real-time speeds, we derive a similar local/global formulation for the implicit midpoint minimization problem. The resulting questions are: (the derivation is shown in Appendix A)

$$\begin{aligned} g(\mathbf{x},\mathbf{p}) = &\frac{1}{2}(\mathbf{x}-\mathbf{y})^{\mathsf{T}}\mathbf{M}(\mathbf{x}-\mathbf{y}) \\ &+h^2\left(\frac{1}{2}\left(\frac{\mathbf{x}+\mathbf{x}_n}{2}\right)^{\mathsf{T}}\mathbf{L}\left(\frac{\mathbf{x}+\mathbf{x}_n}{2}\right)-\left(\frac{\mathbf{x}+\mathbf{x}_n}{2}\right)^{\mathsf{T}}\mathbf{J}\mathbf{p}\right) \end{aligned} \tag{3.49}$$

Where $\mathbf{p}$ is the auxiliary variable, $\mathbf{x}$ is the unknown position vector, $\mathbf{y}$, $\mathbf{M}$, $\mathbf{J}$, $\mathbf{L}$ are constants. Solving the optimization problem can be done using block coordinate descent. First, we assume $\mathbf{x}$ is fixed and solve for $\mathbf{p}$. This step can be done in parallel. Once we have $\mathbf{p}$, we assume it is fixed and solve for $\mathbf{x}$ by soling the following linear system:

$$\left(\mathbf{M}+\frac{h^2}{4}\mathbf{L}\right)\mathbf{x} = \left(\mathbf{M}\mathbf{y}+\frac{h^2}{2}\mathbf{J}\mathbf{p}-\frac{h^2}{4}\mathbf{L}\mathbf{x}_n\right) \tag{3.50}$$

The system matrix for implicit midpoint is $\mathbf{M}+\frac{h^2}{4}\mathbf{L}$, which is slightly different from the original Projective Dynamics matrix $\mathbf{M}+h^2\mathbf{L}$ derived from backward Euler. In our method, we pre-factorize both of these matrices. The sparse Cholesky factors of $\mathbf{M}+\frac{h^2}{4}\mathbf{L}$ are used in the initial implicit midpoint iterations and the factors of $\mathbf{M}+h^2\mathbf{L}$ are used for the backward Euler stabilization (blending with forward Euler does not require any linear solves). When using backward Euler to stabilize the local/global approximation of implicit midpoint, the initial guess becomes important. Projective Dynamics typically uses the inertia term $\mathbf{y} := \mathbf{x}_n + h\mathbf{v}_n$ as the initial guess, but since we already have an approximate implicit midpoint solution, we can use it as a more effective initial guess.

Projective Dynamics is usually iterated only for a fixed number of iterations [27, 96]. For our method, we typically run 10 iterations of the local/global process for the initial implicit midpoint solve and only one iteration for the backward Euler for stabilization. This is sufficient due to the fact that we use the result of implicit midpoint as a starting point, yielding a good backward Euler solution even with only one local/global iteration. In addition to backward Euler, our method also uses forward Euler to inject energy into the system if necessary. However, since the forward Euler step can be easily computed

explicitly, we do not need any approximate numerical solve; we directly use the update rules to blend with implicit midpoint.

3.2.3 Non-conservative Forces

Damping is an important aspect of real-world material behavior. In physics-based animation, damping is often used to control the amount of dynamic motion. As discussed in [139] (Section 4.3), undamped simulations may result in high-frequency oscillations that expose the underlying mesh structure, which is undesired. A very simple explicit damping model is an ether drag model:

$$\mathbf{v}_i^{\text{damped}} = \mathbf{v}_i - k\frac{h}{m_i}\mathbf{v}_i \tag{3.51}$$

This model has many problems, mainly that it also damps out the global motion. More sophisticated models, such as the implicit damping model proposed by [80], can mitigate these problems at the expense of extra computation. While this model worked well in our experiments, the implicit solve it requires was too slow for real-time physics.

This motivates our choice of damping model. Physically, we want damping that models the energy dissipation due to internal friction in the simulated material, transforming mechanical energy into heat (as opposed to modeling dissipation due to outside forces such as air drag). We also need our model to be fast while preserving the rigid body modes of the motion. Therefore, in our system we chose to use the momentum-preserving damping model introduced in [114]. The key idea is to calculate the difference between the velocity of each vertex and its velocity in the best-fit rigid body approximation and damp out only these non-rigid velocity components:

$$\Delta\mathbf{v}_i = \mathbf{v}_i - \mathbf{v}_{\text{cm}} + \boldsymbol{\omega}_{\text{cm}} \times \mathbf{r}_i \tag{3.52}$$

$$\mathbf{v}_i^{\text{damped}} = \mathbf{v}_i - k\Delta\mathbf{v}_i \tag{3.53}$$

where $k \in [0, 1]$ is the damping coefficient, $\mathbf{v}_i$ is the original (pre-damping) velocity of vertex i, $\mathbf{v}_{\text{cm}}$ is the velocity of the center of mass, and $\mathbf{r}_i = \mathbf{x}_i - \mathbf{x}_{\text{cm}}$ is the vector that points from the current position ($\mathbf{x}_i$) to the center of mass ($\mathbf{x}_{\text{cm}}$). $\mathbf{x}_{\text{cm}}$ and $\mathbf{v}_{\text{cm}}$ can be easily computed:

$$\mathbf{x}_{\text{cm}} = \frac{\sum_i m_i\mathbf{x}_i}{\sum_i m_i} \quad \mathbf{v}_{\text{cm}} = \frac{\sum_i m_i\mathbf{v}_i}{\sum_i m_i} \tag{3.54}$$

where m_i is the mass of vertex i. To compute the angular velocity $\boldsymbol{\omega}_{\text{cm}}$, we need the angular momentum $\mathbf{L}$ and inertia matrix $\mathbf{I}_{\text{cm}}$. $\mathbf{L}$ is easy to find using the definition of angular momentum:

$$\mathbf{L} = \sum_i \mathbf{r}_i \times m_i \mathbf{v}_i \tag{3.55}$$

The inertia matrix $\mathbf{I}_{\text{cm}}$ can be computed as:

$$\mathbf{I}_{\text{cm}} = \sum_i m_i \mathbf{R}_{\times i} \mathbf{R}_{\times i}^{\mathsf{T}} \tag{3.56}$$

where $\mathbf{R}_{\times i} \in \mathbb{R}^{3\times 3}$ is the cross-product matrix of $\mathbf{r}_i$ (i.e, given an arbitrary vector $\mathbf{a}$, $\mathbf{R}_{\times}\mathbf{a} = \mathbf{r} \times \mathbf{a}$). Finally, we can compute the angular velocity as $\boldsymbol{\omega}_{\text{cm}} = \mathbf{I}_{\text{cm}}^{-1}\mathbf{L}$. The rigid body component of the motion is fully described by $\mathbf{v}_{cm}$ and $\boldsymbol{\omega}_{cm}$. The damping model according to Eq. 3.53 can be thought of as damping velocities that go against this rigid body motion. Choosing $k = 0$ corresponds to no damping at all. Choosing $k = 1$ means that all non-rigid velocity components will be eliminated, resulting in pure rigid-body motion. Intermediate values of k allow us to control the non-rigid modes of the motion and are useful to suppress fast oscillations which may not be visually appealing.

This is a fast damping model that preserves the rigid motion, however, it is not a physically accurate damping model. If a body is highly deformed, for example a rope that is coiled, this model can cause non-physical damping of the non-rigid motion. Furthermore, it is not dependant on the time step, so changing the time step can change the result. However, we choose to use this damping model due to its explicit computation, which fits our real-time constraints.

Regardless of the damping model chosen, we only need to make a slight modification to our error function. Rather than viewing H_{total} in Eq. 3.44 as the starting energy, we can view it as the target energy we want to reach. In damped simulations, we need modify H_{total} by subtracting energy that dissipated away due to damping. We will call this value H_{diss} – the total energy intentionally dissipated from the system by our damping model. This approach enforces some limitation on the damping model; namely, it has to be done in a separate step from the optimization. We can apply the same logic for forcing, i.e., intentional energy injections into the system (e.g., if the user perturbs the simulated object by applying external forces). Similarly to damping, in the case of forcing we again update

H_{total} to take into account this extra injected energy. We introduce variable H_{added} that will represent the amount of energy intentionally inserted into the system.

Algorithm 1: Our Method

$\mathbf{x} := \mathbf{x}_0$
$\mathbf{v} := \mathbf{v}_0$
$H_{\text{total}} := \texttt{calculateTotalEnergy}(\mathbf{x}, \mathbf{v})$
while !*exit* **do**
  $(\mathbf{x}_n^{\text{UI}}, \mathbf{v}_n^{\text{UI}}) = \texttt{userInteraction}(\mathbf{x_n}, \mathbf{v_n})$
  H_{added} = $\texttt{calculateTotalEnergy}(\mathbf{x}_n^{\text{UI}}, \mathbf{v}_n^{\text{UI}}) -$
    $\texttt{calculateTotalEnergy}(\mathbf{x}_n, \mathbf{v}_n)$
  $H_{\text{total}} := H_{\text{total}} + H_{\text{added}}$
  $(\mathbf{x}_{\text{IM}}, \mathbf{v}_{\text{IM}}) = \texttt{IMsolve}(\mathbf{x}_n^{\text{UI}}, \mathbf{v}_n^{\text{UI}})$
  if $e(\mathbf{x}_{IM}, \mathbf{v}_{IM}) > 0$ **then**
    $(\mathbf{x}_{\text{BE}}, \mathbf{v}_{\text{BE}}) := \texttt{BEsolve}(\mathbf{x}_{\text{IM}}, \mathbf{v}_{\text{IM}}, \mathbf{x}_n, \mathbf{v}_n)$
    $(\mathbf{x}_{n+1}, \mathbf{v}_{n+1}) := \texttt{blend}(\mathbf{x}_{\text{IM}}, \mathbf{v}_{\text{IM}}, \mathbf{x}_{\text{BE}}, \mathbf{v}_{\text{BE}})$
  else if $e(\mathbf{x}_{IM}, \mathbf{v}_{IM}) < 0$ **then**
    $(\mathbf{x}_{\text{FE}}, \mathbf{v}_{\text{FE}}) := \texttt{FEsolve}(\mathbf{x}_n, \mathbf{v}_n)$
    $(\mathbf{x}_{n+1}, \mathbf{v}_{n+1}) := \texttt{blend}(\mathbf{x}_{\text{FE}}, \mathbf{v}_{\text{FE}}, \mathbf{x}_{\text{IM}}, \mathbf{v}_{\text{IM}})$
  end
  $\mathbf{v}_{\text{damped}} := \texttt{damp}(\mathbf{v_{n+1}})$
  H_{diss} = $\texttt{calculateTotalEnergy}(\mathbf{x}_{n+1}, \mathbf{v}_{n+1}) -$
    $\texttt{calculateTotalEnergy}(\mathbf{x}_{n+1}, \mathbf{v}_{\text{damped}})$
  $H_{\text{total}} := H_{\text{total}} - H_{\text{diss}}$
  $\mathbf{v}_{n+1} := \mathbf{v}_{\text{damped}}$
  $n := n + 1$
end

There are several ways to implement forcing. In our implementation, the user can interact with the simulated object by using the mouse to move a special set of user-controlled vertices. These user-controlled vertices are not degrees of freedom of the simulation (i.e., they are not part of the system state **x** or velocities **v**), but they are connected to the actual degrees of freedom (free vertices). For example, our hanging cloth example uses two user-controlled vertices as "attachment points" which are connected to the actual simulated vertices via springs, which propagate the user-specified (e.g., keyframed) motion to the simulation. The springs connecting the user-controlled and simulated vertices are included in the potential function. At the beginning of the frame, before we perform any integration, we check if the user has moved the user-controlled vertices compared to the last frame. If they have, then we can compute H_{added} by subtracting the potential value *after* user

manipulation from the previous potential value, *before* user manipulation. Since we have not yet performed any updates to the system, we know that this difference in potential was entirely a result of user interaction, i.e., the energy the user injected into the system.

Combining this with the damping term, we get:

$$H_{\text{total}} = H_{\text{initial}} + H_{\text{added}} - H_{\text{diss}} \tag{3.57}$$

where H_{initial} is the initial total energy of the system according to the initial conditions (e.g., pre-stretched starting states or initial velocities correspond to larger H_{initial}). This energy-tracking process is crucial to our blending as it allows our method to handle damping and forcing, which are very common actions in interactive simulations. The pseudocode of our method is outlined in Alg. 1. Here, BEsolve($\mathbf{x}_{\text{IM}}, \mathbf{v}_{\text{IM}}, \mathbf{x}_n, \mathbf{v}_n$) means running a backward Euler solve using $\mathbf{x}_{\text{IM}}, \mathbf{v}_{\text{IM}}$ as an initial guess. FEsolve($\mathbf{x}_n, \mathbf{v}_n$) means performing a standard forward Euler step.

Collisions. While [31] presented robust models for handling collisions (including self-collisions) using adaptive time-stepping, robust real-time solutions remain a challenge. To support collisions in our current system, we use the standard model of repulsion springs [107]. Specifically, if an inter-penetration is detected, we introduce collision springs with the following potential function:

$$E_c(\mathbf{x}) = \begin{cases} \frac{1}{2} k_c ||\mathbf{d} \cdot \mathbf{n}||^2 & \mathbf{d} \cdot \mathbf{n} \leq 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.58}$$

Here, k_c is the stiffness for the collision springs, $\mathbf{d} = \mathbf{x} - \mathbf{x}_s$ is penetration depth of the vertex ($\mathbf{x}$ is the current position, and $\mathbf{x}_s$ is the projection of $\mathbf{x}$ onto the nearest point on the surface), and $\mathbf{n}$ is the surface normal at $\mathbf{x}_s$. This potential is included in $E(\mathbf{x})$ along with regular deformation energies. The collision detection is executed at the beginning of every frame and all of the necessary collision springs are inserted into the potential function and taken into account during the subsequent integration step.

An important aspect of modeling collisions is modeling friction between the two colliding objects. We model friction by applying a damping force after the integration [58]. For every colliding vertex, we subtract the component of the acceleration that is tangent to collision normal $\mathbf{n}$:

$$v_i^f = v_i - k_f a_\perp(x_i) \tag{3.59}$$

Where $k_f \in (0, 1)$ is a friction coefficient, and $a_\perp(x_i) = h\frac{\nabla E_c(x_i)_\perp}{m_i}$ is the tangent component of the discrete acceleration caused by the collision penalty forces. From here, we can treat it just like damping – we simply update the H_{diss} term from Alg. 1 in the same way to take this intentionally dissipated energy into account. Like the damping model, this friction model is physically inaccurate and is chosen for its explicit evaluation, leading to a fast computation.

Perfectly elastic collisions (i.e., collisions without any damping) are energy-conserving phenomena. While the total energy of an individual object can increase (i.e., a large moving object colliding with a small stationary object will insert energy into the small object), the total energy of the system cannot increase. For example, a ball falling onto a surface should not bounce higher than its starting point. The extra energy induced from the collision springs is therefore not included in our energy tracking (Eq. 3.57), since it is not energy that was originally in either object before colliding; it is temporarily introduced to handle interpenetrations. However, sometimes we want to model inelastic collisions, where energy is lost during the collision event. While we did not use this in our examples, this energy dissipation could be tracked in exactly the same way as other damping models, i.e., by accounting for the amount of energy intentionally dissipated during inelastic collisions in the H_{diss} term (Eq. 3.57).

3.2.4 Results

Energy conservation. In Figure 3.10 we compare the energy preservation behavior of our method and other methods over the course of several thousand time steps. The simulation we used was a deformable cube modeled using corotated elasticity with linear finite elements ([34], [131]) spinning around a fixed axis. Backward Euler (BDF1) as well as its second order counterpart (BDF2) damp out most of the energy, whereas pure implicit midpoint quickly explodes. Implicit Newmark (using $\gamma = 1/2, \beta = 1/4$, i.e., the trapezoidal rule) survives longer than implicit midpoint but still eventually explodes. We also tried to apply the energy budgeting method of Su et al. [139] to correct the implicit midpoint behavior. Even though the energy-budgeted simulation survives longer, the method fails to stabilize systems in the long-term and also explodes. This is due to the fact that projection methods such as [139] only modify the velocities, but not positions and, therefore, large

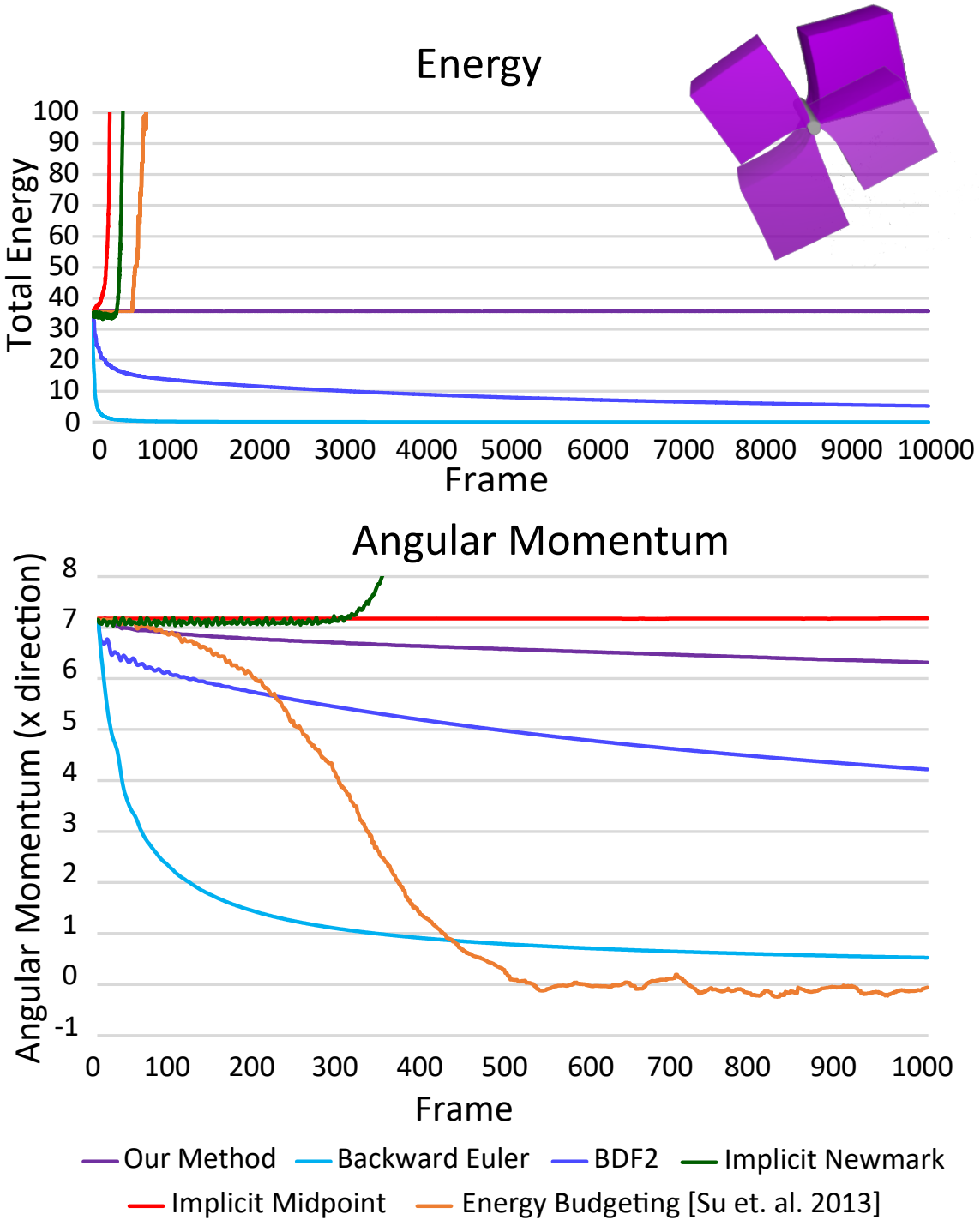

Figure 3.10: A graph showing the total energy (top) and angular momentum (bottom) of a spinning elastic cube integrated using various methods all evaluated using a time step of $h = 0.033$ s. Our method is stable and preserves energy very well even in long simulation runs. While our method does not exactly conserve angular momentum, it preserves it better than the other stable methods (backward Euler and BDF2).

erroneous potential energy can build up in the deformation modes. Our method avoids this problem by changing both positions and velocities and maintains the total energy over a long run.

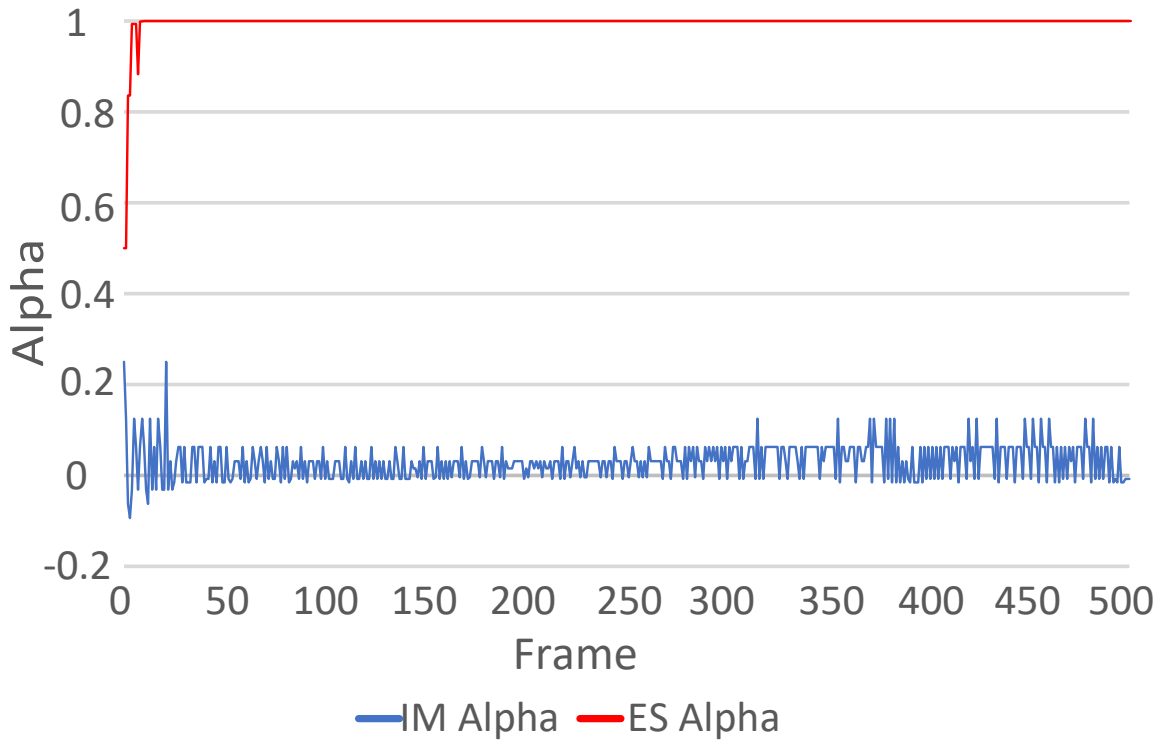

Figure 3.11: The choice of blending alpha values that were used for the simulation in Figure 3.10, using implicit midpoint (IM) and explicit symplectic Euler (ES) as starting points.

Momentum conservation. Momentum conservation is another important aspect of numerical time integration; Figure 3.10 also compares the angular momentum conservation properties of our and previous methods. While our method does not conserve angular momentum exactly, it preserves angular momentum much better than BDF1 or BDF2. Implicit midpoint preserves the angular momentum exactly, but often exhibits dramatic errors in total energy. Our method uses implicit midpoint as a starting point and corrects the energy under/overshoots using forward/backward Euler. Even though these corrections are often small, our method is not exactly angular momentum-conserving.

Alpha value. The chosen alpha for the blending is typically close to zero in reasonable simulations. Figure 3.11 shows the alpha values chosen for the cube example in Figure 3.10.

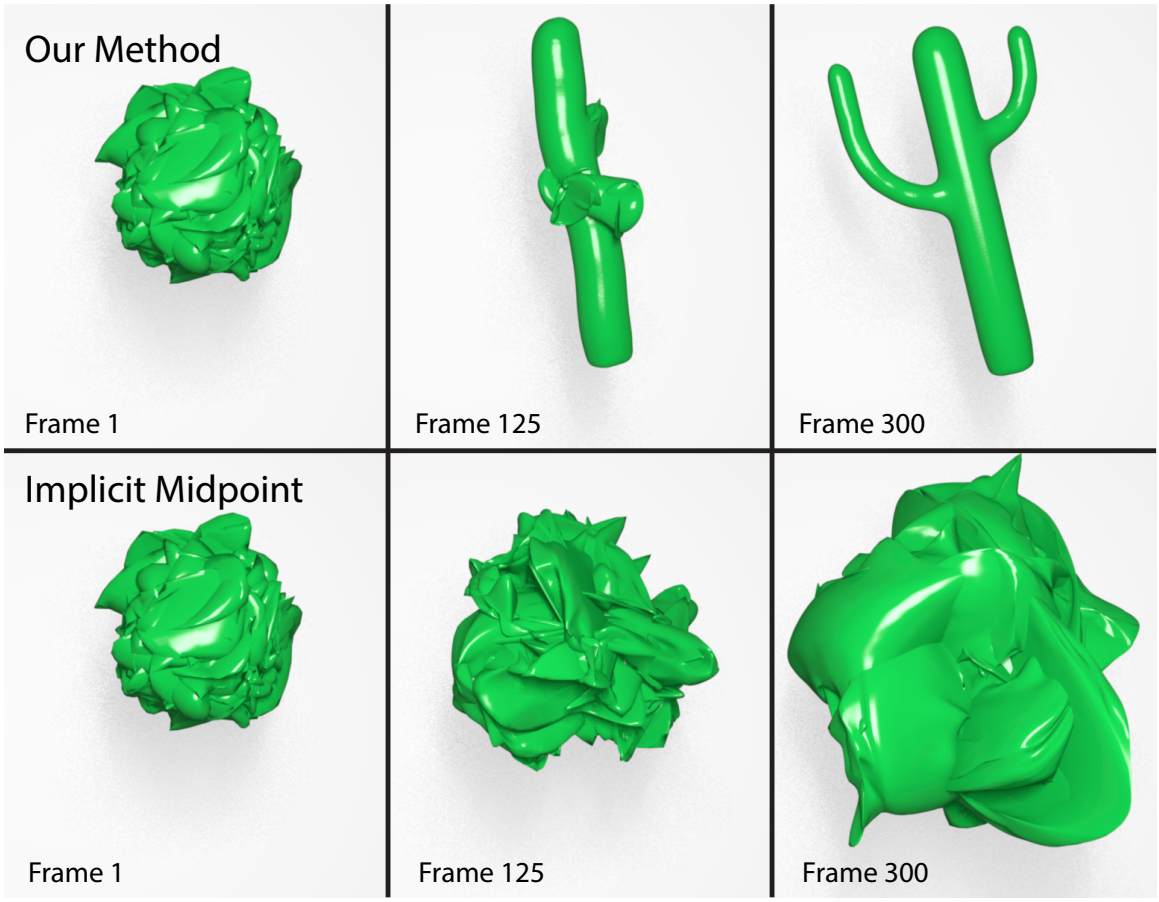

Figure 3.12: An elastic cactus with randomized initial positions integrated using our method (top) and implicit midpoint (bottom), both using time step $h = 0.033$ s. We added a small amount of damping so the cactus eventually returns to its rest pose. However, the energy overshooting of implicit midpoint overpowered the damping and the cactus failed to come to rest even after 10,000 time steps. This was not a problem with our method which quickly recovered the rest pose.

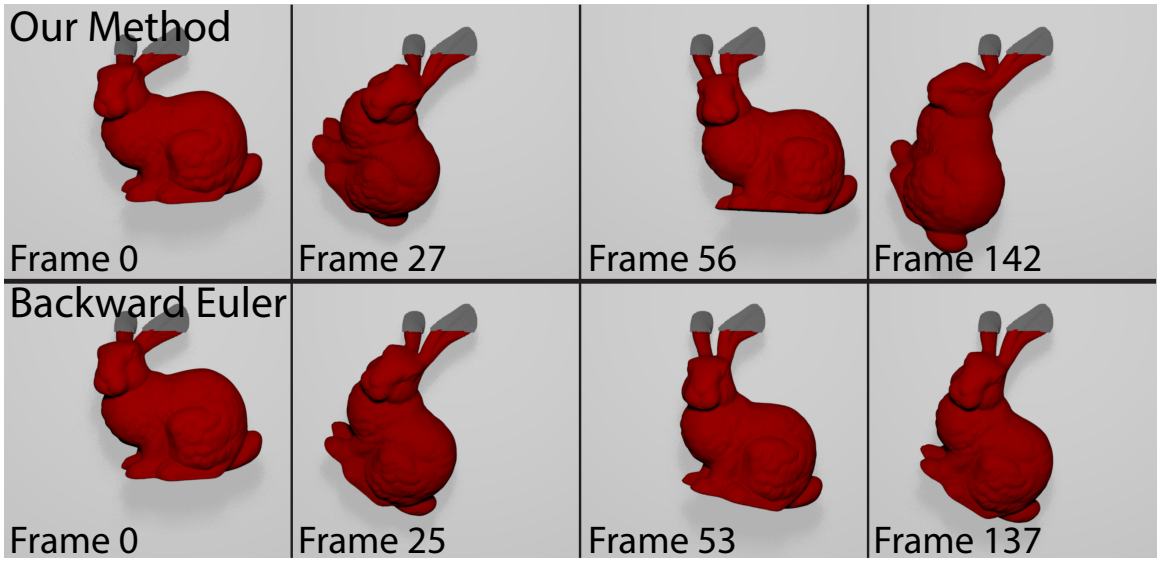

Figure 3.13: An elastic bunny swings under gravity. Our method (top) results in the bunny swinging vividly while backward Euler (bottom) damps out the pendulum motion. The frames chosen correspond to the apex of the pendulum motion for each example.

The forward Euler alphas have been re-scaled to an interval of $[-1, 0]$ for the purposes of showing both the forward and backward Euler blending. A value of -1 corresponds to a full forward Euler solution, and a value of 1 corresponds to a full backward Euler solution. Alpha values close to zero lead to less forward/backward Euler blending, which results in smoother and more vivid motion. Large alpha values correspond to large amounts of backward Euler blending, which can lead to a lot of damping in the angular momentum, as is the case if we use explicit symplectic Euler as a starting point. Explicit symplectic Euler's tendency to explode faster at large time steps causes our algorithm to use an α of almost 1 for most of the simulation, which causes the angular momentum to drop to nearly 0 very rapidly.

Stability. Figure 3.12 shows a stability stress-test of our method, featuring a cactus mesh subject to extreme initial conditions. Specifically, the initial vertex positions were randomized, as seen in Frame 1. The energy-conserving behavior keeps the simulation stable and at a constant energy level, so that introducing a small amount of damping allows us to recover the original mesh. We used the momentum-preserving damping model described by Eq. 3.52 and Eq. 3.53 with a damping coefficient $k = 0.08$. Implicit midpoint is unable to recover the cactus' rest shape with the same amount of damping, since the

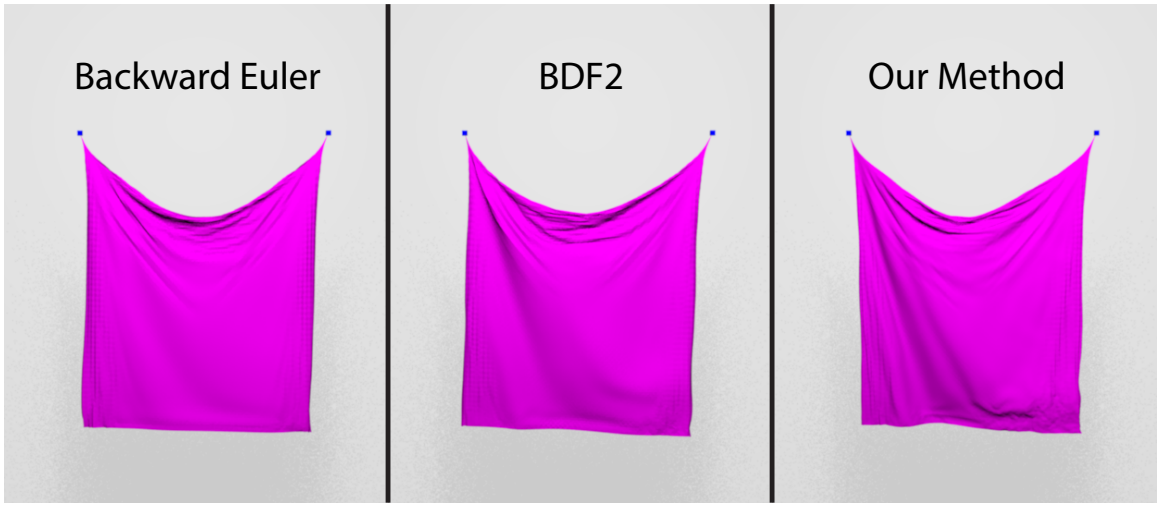

Figure 3.14: Mass-spring system cloth simulated with backward Euler, BDF2, and our method. Our method produces more vivid wrinkles due to lower numerical dissipation.

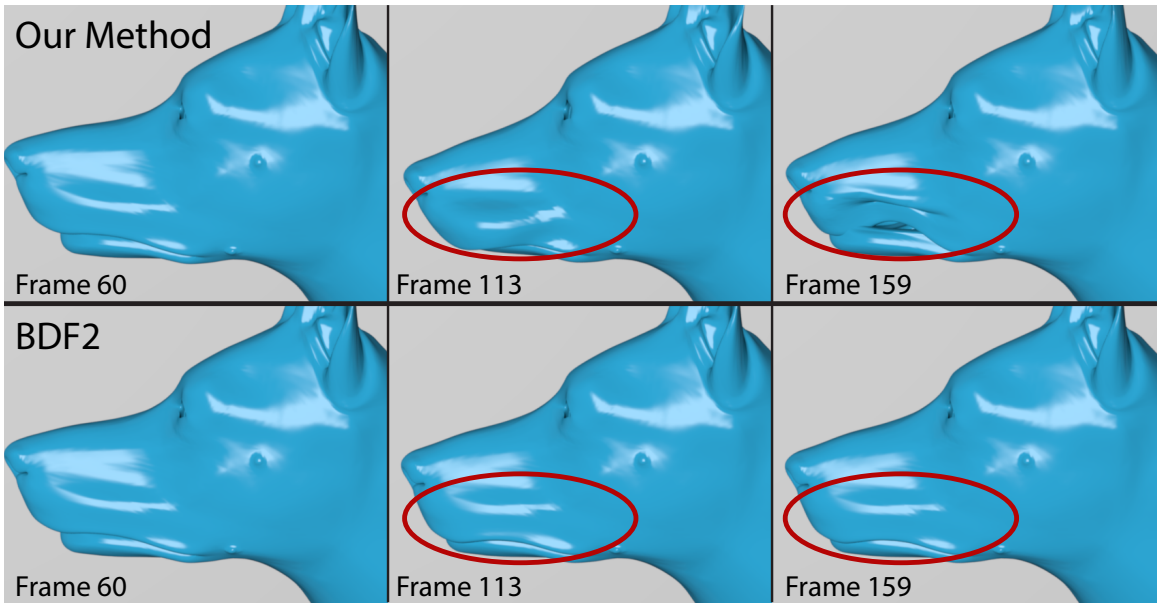

Figure 3.15: Elastic dog face model starting with a pre-stretched nose. Our method keeps the face oscillations while BDF2 damps out the motion.

numerical instabilities overpower the damping effect. We ran the simulation for implicit midpoint for much longer (another 10,000 frames) to see if it would eventually recover the rest pose, but the energy kept increasing despite the damping. Both simulations were run using a time step of $h = 0.033$ s.

Visual motion quality. As previously mentioned, backward Euler and its higher-order version BDF2 both cause artificial damping, which is particularly strong in higher-frequency deformation modes. This can lead to less visually attractive results. Figure 3.14 shows how our method produces more realistic wrinkles in a cloth simulation where the cloth is being shook by one of its corners. This causes waves to propagate through the cloth, which are quickly damped out by backward Euler and BDF2, but preserved by our method. Another example where this high-frequency damping causes undesired results is in Figure 3.15. In this simulation, we aimed for a cartoon-like effect where an elastic dog's nose is stretched out and released, resulting in a humorous jiggling effect on the snout and lips. With backward Euler or BDF2 we were unable to get the desired effect, since all the motion quickly died away and the dog's nose returned to the rest pose. With our method, we were able to get a comical, vivid motion. If required, the vividness can be reduced by introducing user-controlled damping, allowing us to achieve the desired visual effect.

Figure 3.13 shows a deformable bunny pinned at the ears, swinging in a pendulum-like motion. Backward Euler damps the deformation modes in the ears, causing the swinging motion to damp out. Our method keeps the bunny swinging in a lively manner due to conserving the total energy.

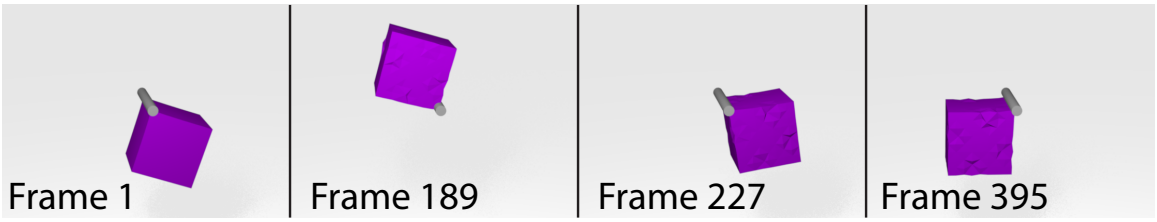

Figure 3.16: Using a looser energy conservation term ($\epsilon = 0.5$) results in poor visual quality.

Our method can be less strict in the conservation of energy, depending on the choice of ϵ in Eq. 3.2.1. For the results in this paper, we used $\epsilon = 0.01$ (i.e., tolerate an increase or decrease of 1%). In Figure 3.16, we demonstrate what happens when we use an

unreasonably loose $\epsilon = 0.5$. While the animation does not explode, the heavy oscillation between the damping of backward Euler and the energy injecting of forward Euler manifests itself as a slowing of the global motion and unreasonable vibrations in the mesh.

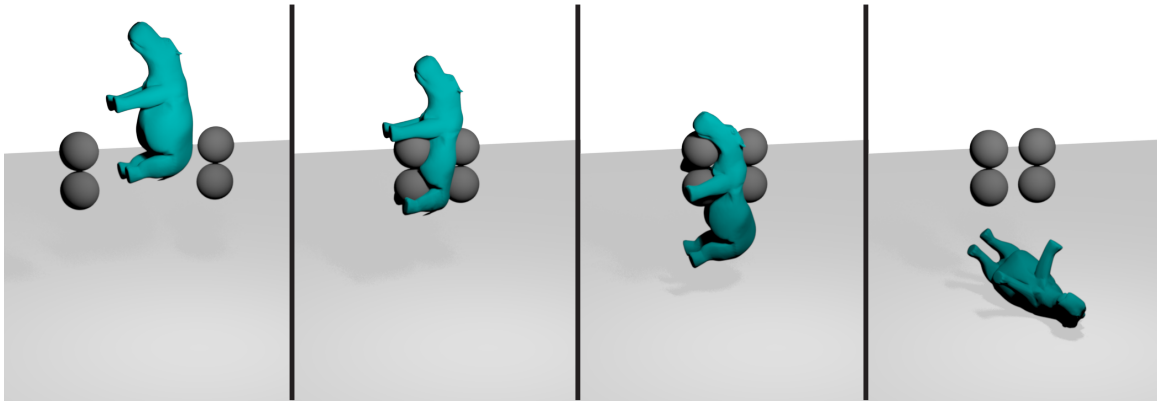

Figure 3.17: Collision test with our method: an elastic hippo collides with four spheres.

Collisions. To test our method's ability to cope with collisions, we designed an experiment featuring an elastic hippo colliding with spheres, see Figure 3.2.4. Because our collision detection and instancing of repulsion springs is executed only once per frame, we run this simulation at a slightly lower time step of $h = 0.01$ s in order to resolve all of the collisions accurately.

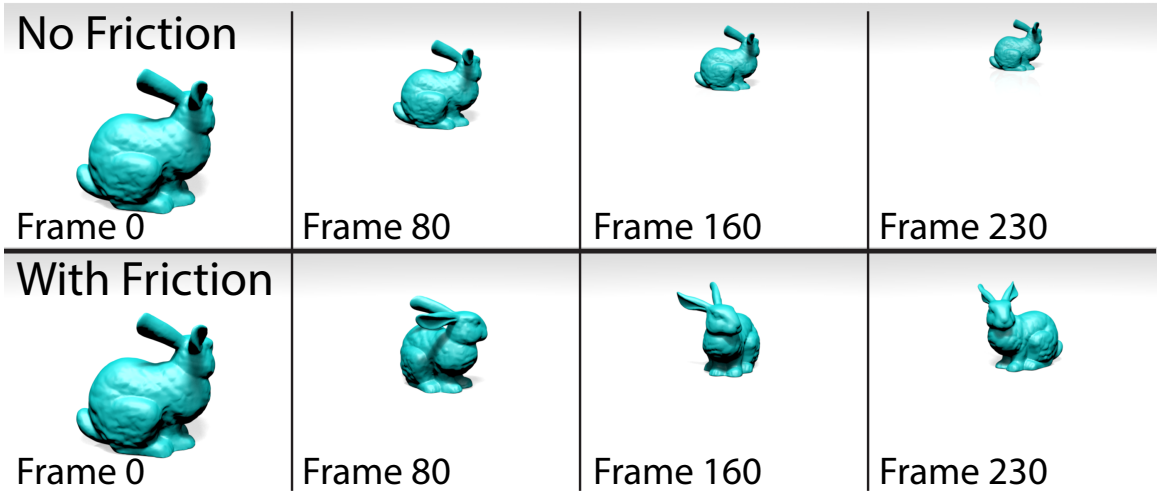

Figure 3.18: A bunny sliding along a plane using our method with no friction (top) and with friction (bottom).

The artificial damping of backward Euler degrades the visual quality of collision resolution, because the repulsion springs are usually stiff (to quickly remove inter-penetrations). This leads to significant artificial energy dissipation during collisions, resulting in rigid-like motion that looks unrealistic for elastic objects, as demonstrated in Figure 3.19. Our method executed in the same scenario produces a more elastic bouncing of the cube and less damping Figure 3.19.

We can incorporate frictional forces between colliding surfaces into our method. Figure 3.2.4 shows our method applied on a bunny sliding along a flat surface with and without friction. The frictionless surface results in the bunny sliding along the plane indefinitely. If we add some friction ($k_f = 0.8$), the bunny not only slows down but also begins rotating due to the asymmetric base. Since the frictional forces only affect the base, the bunny's ears also start moving as they are pulled back from their motion by the elastic forces of the mesh.

Performance. Table 3.1 shows details about our experiments including performance measurements. All experiments used a lumped mass matrix for the masses. We use backward Euler as a baseline for our comparisons, using a local/global solver (Projective Dynamics) with a fixed number of iterations. The α-search process used to determine the amount of blending between implicit midpoint and backward Euler steps is not a bottleneck, as can be seen from the measurements in Table 3.1. With the local/global solver, the situation is more favorable for our method, because the stabilization step uses only one local/global iteration. The motion quality is high because we blend the single-iteration result with a more accurate solution (10 iterations) of implicit midpoint. In our experience, this results in higher motion quality with only small computing overhead.

Our method derives many of its desirable properties from implicit midpoint. What if, instead of using our method, we simply reduced the time step for implicit midpoint? We study this question in Table 3.2, where we use the same simulation scenarios as in Table 3.1, but this time we compare against implicit midpoint with substepping, i.e., splitting one time step into several smaller "substeps". For example, with the original $h = 0.033$ s and substepping factor 10, we use ten steps with 0.0033 s each, advancing the simulated time by the same amount. Decreasing the time step improves the motion quality generated by implicit midpoint, however, the extra computing resources are significant and stability is not guaranteed even with very many substeps. This is especially obvious in collision scenarios,

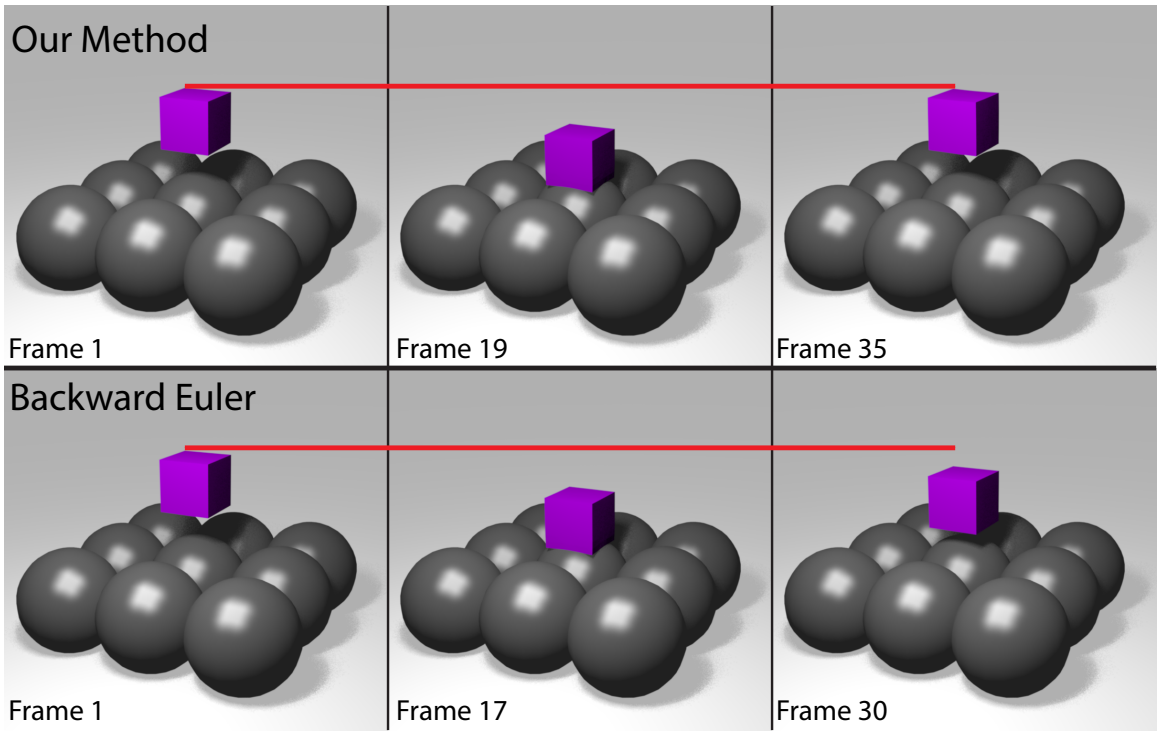

Figure 3.19: An elastic cube falls on a collection of spheres, simulated using our method (top) and backward Euler (bottom). The frames chosen are the maximum and minimum heights of the cube after one bounce for each method. With our method the cube bounces back to the original height, but not with backward Euler due to its numerical damping.

Table 3.1: Computing speed – comparison with backward Euler

model	#ver.	#ele.	time step	Local/Global Solver (10 iterations)		
				Backward Euler	Our Method	
					Total Time	α search
Bar	290	716	0.033	3.62 ms	5.73 ms	1.85 ms
Cube (Figure 3.10)	386	996	0.033	5.84 ms	8.41 ms	1.66 ms
Cube (Figure 3.19)	386	996	0.01	6.93 ms	9.50 ms	0.93 ms
Hippo	2387	8406	0.01	66.4 ms	73.3 ms	4.93 ms
Bunny (Figure 3.2.4)	4497	15408	0.033	194 ms	212 ms	9.62 ms
Cactus	5261	17187	0.033	115 ms	174 ms	39.90 ms
Cloth†	10201	50200	0.033	110 ms	127 ms	7.58 ms
Dog	28390	117423	0.033	916 ms	1205 ms	193 ms
Bunny (Figure 3.13)	34844	121058	0.033	1181 ms	1510 ms	215 ms

We compare the performance of our method to backward Euler using a Projective Dynamics Local/Global (right, using 10 iterations). All models use corotated elasticity with linear finite elements, except for the Cloth† model which is a mass-spring system. The α-search time is accounted for in the "Total Time" of our method. Both backward Euler (i.e., Projective Dynamics) and implicit midpoint used in first phase of our method always use 10 iterations. In the second (stabilizing) phase of our method, we use only one iteration of backward Euler. All of the reported times were taken as an average over 30 frames. The Cube (Fall) and Hippo are collision tests, using smaller time step to accurately resolve collisions. The Cactus example is a stress-test with randomized initial positions (i.e., not a typical case in practical simulations), which explains the longer α-search time.

where even very small time steps are not sufficient to guarantee stability. For example, in our Hippo example (Figure 3.2.4), even 100 substeps of implicit midpoint was not enough to produce a visually plausible result. Our method produces stable results despite relatively large time steps. In addition to stability, our method also delivers much better energy and angular momentum preservation than backward Euler. Applying a local/global solver rather than using Newton's method results in a fast yet robust integration algorithm, with only minimal computing overheads compared Projective Dynamics [27].

3.3 FEPR: Fast Energy Projection

This work was first published in ACM Transaction on Graphics. (Vol. 37 Issue 4 2018) and presented at ACM SIGGRAPH 2018. We present a new light-weight technique to improve the quality and robustness of dynamic simulations, especially useful in real-time simulations. Our method is an "add-on" which can be applied after any combination of an integration rule and its iterative solution process, unlike the previous method which

Table 3.2: Comparison of our method and implicit midpoint (local/global solver, 10 iterations)

model	Our Method	Implicit Midpoint				
		1 ss	5 ss	10 ss	20 ss	100 ss
Bar	5.73 ms	3.79 ms	18.22 ms	36.42 ms	72.73 ms	361 ms
Cube (Figure 3.10)	8.41 ms	6.03 ms	29.86 ms	54.95 ms	111 ms	541 ms
Cube (Figure 3.19)	9.50 ms	7.16 ms	56.85 ms	122 ms	245 ms	1232 ms
Hippo	73.3 ms	68.4 ms	312 ms	666 ms	1230 ms	6615 ms
Bunny (Figure 3.2.4)	212 ms	184 ms	830 ms	2014 ms	3568 ms	18907 ms
Cactus	174 ms	120 ms	590 ms	1242 ms	2442 ms	13849 ms
Cloth	127 ms	109 ms	478 ms	997 ms	1976 ms	8388 ms
Dog	1205 ms	904 ms	4413 ms	9079 ms	16660 ms	79369 ms
Bunny (Figure 3.13)	1510 ms	1101 ms	5616 ms	11293 ms	22577 ms	120631 ms

Our examples from Table 3.1 compared against implicit midpoint with substepping (ss), i.e., reducing the time step size by 5, 10, 20, and 100 times. A cell color of green indicates that the simulation is stable at this substep level, while a cell color of red indicates that the simulation is not stable. Reducing the time step improves stability for implicit midpoint but at significant computing costs.

required using implicit midpoint as a base integrator. This method is motivated by two elementary observations: 1) numerical "explosions" are usually characterized by the total energy of the system increasing indefinitely and 2) artificial numerical damping is characterized by the decay of the total energy. These behaviors are due to numerical errors which contradict an important principle of nature – conservation of energy.More precisely, total energy, i.e., the sum of potential and kinetic energies, is constant in conservative dynamical systems. Nevertheless, numerical methods commonly used in real-time physics do not conserve energy, leading to artifacts such as the artificial damping of backward Euler. Our method can be viewed as the projection of the state of a dynamical system (positions and velocities) which restores the correct energy behavior. If this is done carefully, i.e., by not altering the state too much and paying attention also to linear and angular momentum in addition to the total energy, we observe qualitatively improved simulations in a number of practical real-time-physics settings. Our method guarantees that 1) the simulation cannot "explode" in the sense of unbounded energy increases; 2) artificial numerical damping – if present in the underlying integrator– is eliminated.

3.3.1 Method

The previous method does a good job of producing a stable simulation, but it is not without its drawbacks. The most significant drawback is that it does not respect the momentum structure of the system: since implicit midpoint is momentum-conserving, blending with backward Euler will introduce some damping in the momentum. We would like for a projection that respects the momentum structure if possible. We start by explaining the primary definitions and notations before we detail our method. The total energy of our system is defined as $H(\mathbf{x}, \mathbf{v}) = E(\mathbf{x}) + \frac{1}{2}||\mathbf{v}||^2_{\mathbf{M}}$, i.e. the sum of potential and kinetic energies. We also define the total linear momentum $P(\mathbf{v}) = \sum_i m^i \mathbf{v}^i$ and total angular momentum $L(\mathbf{x}, \mathbf{v}) = \sum_i \mathbf{x}^i \times m^i \mathbf{v}^i$ where i indexes individual particles, $\mathbf{x}^i \in \mathbb{R}^3$, $\mathbf{v}^i \in \mathbb{R}^3$ are positions and velocities of each particle and m^i are masses of corresponding vertices. If the potential is time invariant, i.e., the forces are conservative, the exact solution conserves the total energy: $H(\mathbf{x}_{n+1}, \mathbf{v}_{n+1}) = H(\mathbf{x}_n, \mathbf{v}_n)$. If the potential is translational and rotational invariant (e.g., an elastic object without external forces or boundary conditions), the exact solution conserves momenta: $P(\mathbf{v}_{n+1}) = P(\mathbf{v}_n)$ and $L(\mathbf{x}_{n+1}, \mathbf{v}_{n+1}) = L(\mathbf{x}_n, \mathbf{v}_n)$.

Numerical integrators do not compute the exact solution, but only its approximation. The error in this approximation can be further exacerbated if the integration rules are not solved exactly, but only approximated e.g. via linearization. Particularly problematic errors are consistent increases or decreases of the total energy, which can manifest themselves as "explosions" or numerical damping. To correct for these errors, we propose an energy-momentum projection method. We project the results of any time integration method onto a constant-energy manifold by solving the following optimization problem:

$$\begin{aligned} \underset{\mathbf{x},\mathbf{v},s,t}{\text{minimize}} \quad & \frac{1}{2}\|\mathbf{x} - \mathbf{x}_{n+1}\|^2_{\mathbf{M}} + \frac{h^2}{2}\|\mathbf{v} - \mathbf{v}_{n+1}\|^2_{\mathbf{M}} + \frac{\epsilon}{2}(s^2 + t^2) \\ \text{subj. to} \quad & H(\mathbf{x}, \mathbf{v}) = H(\mathbf{x}_n, \mathbf{v}_n) \\ & P(\mathbf{v}) = P(\mathbf{v}_{n+1}) + s(P(\mathbf{v}_n) - P(\mathbf{v}_{n+1})) \\ & L(\mathbf{x}, \mathbf{v}) = L(\mathbf{x}_{n+1}, \mathbf{v}_{n+1}) + \\ & \qquad t(L(\mathbf{x}_n, \mathbf{v}_n) - L(\mathbf{x}_{n+1}, \mathbf{v}_{n+1})) \end{aligned} \tag{3.60}$$

where $(\mathbf{x}, \mathbf{v})$ is the resulting projected state and $s \in \mathbb{R}$ and $t \in \mathbb{R}$ are auxiliary variables (discussed below). The constant $(\mathbf{x}_{n+1}, \mathbf{v}_{n+1})$ is the state computed with an arbitrary time integration method and the term $\|\mathbf{x} - \mathbf{x}_{n+1}\|^2_{\mathbf{M}} + \frac{h^2}{2}\|\mathbf{v} - \mathbf{v}_{n+1}\|^2_{\mathbf{M}}$ ensures that our projection

deviates from $(\mathbf{x}_{n+1}, \mathbf{v}_{n+1})$ as little as possible. In our experiments, the diagonal $\mathbf{M}$ matrix are either nodal masses for mass-spring systems or a lumped mass-matrix of linear finite elements. While not strictly necessary, we recommend to scale weight of the velocity term with h^2 in order to make the units in both terms consistent to facilitate numerical optimization. The constant ϵ is a regularization weight, in all of our experiments set to $\epsilon = 0.001$.

Variables s, t. In an exact solution, if the potential E is translation invariant, linear momentum is conserved, i.e., $P(\mathbf{v}_n) = P(\mathbf{v}_{n+1})$ and the s parameter becomes moot as the constraint reduces to $P(\mathbf{v}) = P(\mathbf{v}_{n+1})$. Similarly, if E is rotation invariant, angular momentum is conserved, i.e., $L(\mathbf{x}_n, \mathbf{v}_n) = L(\mathbf{x}_{n+1}, \mathbf{v}_{n+1})$ and the t parameter becomes moot as the constraint reduces to $L(\mathbf{x}, \mathbf{v}) = L(\mathbf{x}_{n+1}, \mathbf{v}_{n+1})$. Therefore if E is both translation and rotation invariant, we can discard the s and t variables and simplify the momentum constraints to: $P(\mathbf{v}) = P(\mathbf{v}_{n+1}), L(\mathbf{x}, \mathbf{v}) = L(\mathbf{x}_{n+1}, \mathbf{v}_{n+1})$. We originally tried this approach with momentum conserving integrators and tested with potentials E both with and without translation or rotation invariance. This did not always work, because if the linear or angular momentum varies, numerical integrators (even exactly-solved momentum conserving integrators) do not compute the exact value of the momenta, but only their numerical approximation. Due to these numerical errors, requesting the projection step to exactly match the momenta $P(\mathbf{v}_{n+1})$ and $L(\mathbf{x}_{n+1}, \mathbf{v}_{n+1})$ and the total energy $H(\mathbf{x}_n, \mathbf{v}_n)$ may be impossible. Specifically, in some cases we observed the numerical integrator (specifically, we used accurately solved implicit midpoint) overshoots the momenta, resulting in momentum constraints incompatible with the total energy constraint – the momentum constraints are asking for so large velocities the total energy constraint cannot accommodate them. To be able to guarantee feasibility of our optimization problem, we introduced the auxiliary scalar variables s and t. To prove that the constrained optimization problem in Eq. 3.60 is always feasible, we plug in $\mathbf{x} := \mathbf{x}_n, \mathbf{v} := \mathbf{v}_n, s := 1, t := 1$ and see that all of the constraints are indeed satisfied.

3.3.2 Numerical Solution

Eq. 3.60 is an optimization problem with a quadratic objective and 7 equality constraints: 3 linear constraints for linear momentum, 3 quadratic constraints for angular momentum

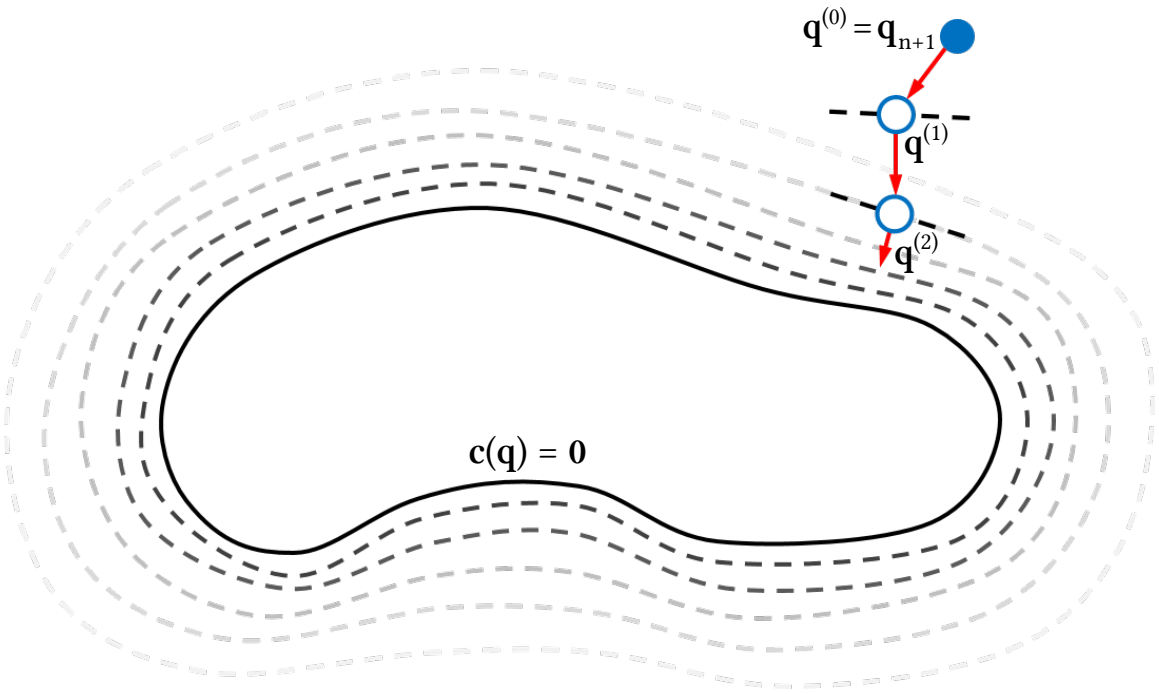

Figure 3.20: Geometric interpretation of Eq. 3.67. Instead of projecting $\mathbf{q}_{n+1}$ directly onto the manifold $\mathbf{c}(\mathbf{q}) = \mathbf{0}$, Eq. 3.67 projects the next iterate $\mathbf{q}^{(k+1)}$ from the current one, $\mathbf{q}^{(k)}$.

and 1 nonlinear constraint for energy. We had initially employed a general-purpose interior-point solver IPOPT [151] which worked well, but was too slow. In this section we propose our own solver which takes advantage of the special structure of our optimization problem to achieve fast runtime performance.

First, we simplify the notation used in Eq. 3.60. Let $\mathbf{q} := [\mathbf{x}; \mathbf{v}; s; t] \in \mathbb{R}^{6m+2}$ be a stacked vector containing all variables, where m is the number of vertices of our simulated system. Next, let us denote $\mathbf{q}_{n+1} := [\mathbf{x}_{n+1}; \mathbf{v}_{n+1}; 0; 0] \in \mathbb{R}^{6m+2}$ (a constant vector), $\mathbf{D} := diag(\mathbf{M}; h^2\mathbf{M}; \epsilon; \epsilon)$ is a $\mathbb{R}^{(6m+2)\times(6m+2)}$ diagonal matrix and the constraints from Eq. 3.60 are denoted as function $\mathbf{c} : \mathbb{R}^{6m+2} \to \mathbb{R}^7$. All constraints are satisfied iff $\mathbf{c}(\mathbf{q}) = \mathbf{0}$. With this notation, we can rewrite Eq. 3.60 as:

$$\begin{aligned} \underset{\mathbf{q}}{\text{minimize}} \quad & \frac{1}{2}\|\mathbf{q} - \mathbf{q}_{n+1}\|^2_{\mathbf{D}} \\ \text{subj. to} \quad & \mathbf{c}(\mathbf{q}) = \mathbf{0} \end{aligned} \tag{3.61}$$

where the minimizer $\mathbf{q}^*$ can be interpreted as projection of $\mathbf{q}_{n+1}$ to the energy-momentum manifold $\mathbf{c}(\mathbf{q}) = \mathbf{0}$ in the $\mathbf{D}$-metric. To find the constrained minimizer, we can adopt a Sequential Quadratic Programming (SQP) approach [120]. The corresponding Lagrangian

function is $\mathcal{L}(\mathbf{q}, \lambda) = \frac{1}{2}||\mathbf{q} - \mathbf{q}_{n+1}||^2_{\mathbf{D}} + \mathbf{c}(\mathbf{q})^\mathsf{T}\lambda$ and setting its partial derivatives to zero leads to:

$$\mathbf{D}(\mathbf{q} - \mathbf{q}_{n+1}) + \nabla\mathbf{c}(\mathbf{q})\lambda = \mathbf{0} \tag{3.62}$$

$$\mathbf{c}(\mathbf{q}) = \mathbf{0} \tag{3.63}$$

where $\nabla\mathbf{c}(\mathbf{q}) = [\nabla c_1(\mathbf{q}), \nabla c_2(\mathbf{q}), \ldots, \nabla c_7(\mathbf{q})] \in \mathbb{R}^{(6m+2)\times 7}$ is the Jacobian of $\mathbf{c}$, and $c_i(\mathbf{q})$ denotes the i-th constraint. We can linearize Eq. 3.62 and Eq. 3.63 to solve them using Newton's method, which forms a sequence of iterates $(\mathbf{q}^{(1)}, \lambda^{(1)})$, ... , $(\mathbf{q}^{(k)}, \lambda^{(k)})$ converging to $(\mathbf{q}^*, \lambda^*)$. We can start with an initial guess $\mathbf{q}^{(0)} := \mathbf{q}_{n+1}$ and $\lambda^{(0)} = \mathbf{0}$. Given $(\mathbf{q}^{(k)}, \lambda^{(k)})$ for any $k = 0, 1, 2, \ldots$, the next iterate $(\mathbf{q}^{(k+1)}, \lambda^{(k+1)})$ is computed by solving the following Newton-KKT (Karush-Kuhn-Tucker) system:

$$\begin{aligned}&\begin{bmatrix}\mathbf{D} + \Sigma_i \lambda_i^{(k)} \nabla^2 c_i(\mathbf{q}^{(k)}) & \nabla\mathbf{c}(\mathbf{q}^{(k)}) \\ \nabla\mathbf{c}(\mathbf{q}^{(k)})^\mathsf{T} & \mathbf{0}\end{bmatrix}\begin{bmatrix}\mathbf{q}^{(k+1)} - \mathbf{q}^{(k)} \\ \lambda^{(k+1)} - \lambda^{(k)}\end{bmatrix} \\ &= -\begin{bmatrix}\mathbf{D}(\mathbf{q}^{(k)} - \mathbf{q}_{n+1}) + \nabla\mathbf{c}(\mathbf{q}^{(k)})\lambda^{(k)} \\ \mathbf{c}(\mathbf{q}^{(k)})\end{bmatrix}\end{aligned} \tag{3.64}$$

which can be simplified to:

$$\begin{aligned}&\begin{bmatrix}\mathbf{D} + \Sigma_i \lambda_i^{(k)} \nabla^2 c_i(\mathbf{q}^{(k)}) & \nabla\mathbf{c}(\mathbf{q}^{(k)}) \\ \nabla\mathbf{c}(\mathbf{q}^{(k)})^\mathsf{T} & \mathbf{0}\end{bmatrix}\begin{bmatrix}\mathbf{q}^{(k+1)} - \mathbf{q}^{(k)} \\ \lambda^{(k+1)}\end{bmatrix} \\ &= -\begin{bmatrix}\mathbf{D}(\mathbf{q}^{(k)} - \mathbf{q}_{n+1}) \\ \mathbf{c}(\mathbf{q}^{(k)})\end{bmatrix}\end{aligned} \tag{3.65}$$

This SQP approach converges quickly, but each iterate requires us to solve a $(6m + 9) \times (6m + 9)$ linear system to compute $(\mathbf{q}^{(k+1)}, \lambda^{(k+1)})$, which is too costly for real-time physics. We can accelerate the linear system solve using a quasi-Newton approximation by dropping the term $\Sigma_i \lambda_i^{(k)} \nabla^2 c_i(\mathbf{q}^{(k)})$, leaving only a diagonal matrix $\mathbf{D}$ in the upper-left corner. This approximation is equivalent to linearizing all constraints at the current iterate $\mathbf{q}^{(k)}$ and computing the next iterate $\mathbf{q}^{(k+1)}$ as:

$$\begin{aligned}\mathbf{q}^{(k+1)} := \underset{\mathbf{q}}{\text{argmin}} \quad & \frac{1}{2}||\mathbf{q} - \mathbf{q}_{n+1}||^2_{\mathbf{D}} \\ \text{subj. to} \quad & \mathbf{c}(\mathbf{q}^{(k)}) + \nabla\mathbf{c}(\mathbf{q}^{(k)})^\mathsf{T}(\mathbf{q} - \mathbf{q}^{(k)}) = \mathbf{0}\end{aligned} \tag{3.66}$$

Unfortunately, there is a catch: even though each iteration of Eq. 3.66 can be computed quickly, the number of iterations to convergence increase significantly. We observed very oscillatory sequences of iterates; intuitively, this is because constraints linearized at different

states can vary wildly and fight with objective term which pulls the result towards $\mathbf{q}_{n+1}$. We found that a simple modification avoids this problem:

$$\begin{aligned} \mathbf{q}^{(k+1)} := \underset{\mathbf{q}}{\text{argmin}} \quad & \frac{1}{2}||\mathbf{q}-\mathbf{q}^{(k)}||^2_{\mathbf{D}} \\ \text{subj. to} \quad & \mathbf{c}(\mathbf{q}^{(k)}) + \nabla\mathbf{c}(\mathbf{q}^{(k)})^{\mathsf{T}}(\mathbf{q}-\mathbf{q}^{(k)}) = \mathbf{0} \end{aligned} \tag{3.67}$$

The only difference is that in Eq. 3.67 we use $\mathbf{q}^{(k)}$ in the objective instead of $\mathbf{q}_{n+1}$ as in Eq. 3.66. This way, the oscillatory convergence paths are avoided and the iterative process quickly converges to a solution which exactly satisfies all of our constraints. A geometric interpretation of the sequence generated by this iterative process can be seen in Figure 3.20.

After this modification, the solution is no longer exactly minimizing the $\mathbf{D}-$distance from $\mathbf{q}_{n+1}$. However, since we use $\mathbf{q}_{n+1}$ as our initial guess, this approximation is sufficiently close and produces plausible results. In order to verify this experimentally, we ran the same simulation with both Eq. 3.61 and Eq. 3.67. As shown in Figure 3.21, there is only a minor visual difference between the two results (please see also the acccompanying video).

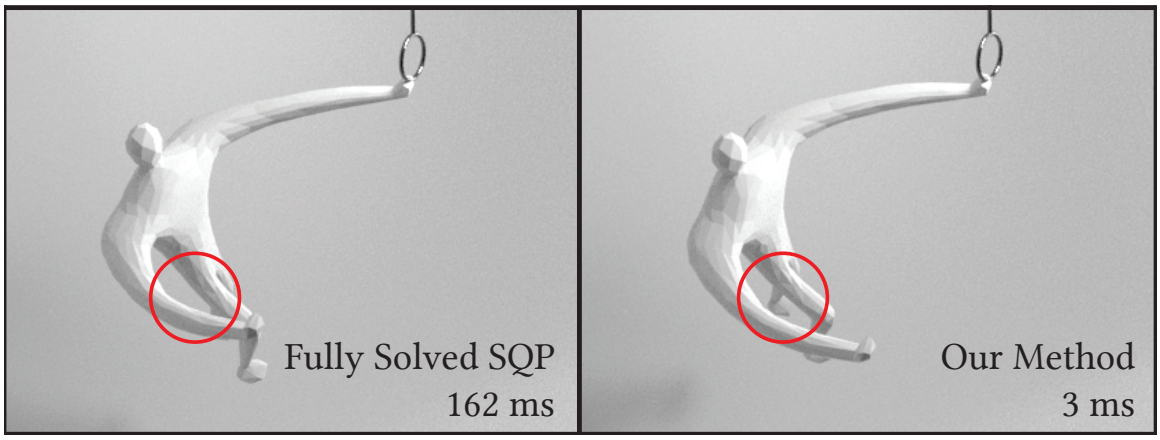

Figure 3.21: Results produced by fully solved SQP (Eq. 3.61) (left) and by our modified optimization problem (Eq. 3.67) (right). Although different (as circled), our method produces qualitatively similar results with fully converged SQP, while being much faster.

In a similar way as before, we solve Eq. 3.67 using Lagrange multipliers. The corresponding Lagrangian is:

$$\hat{\mathcal{L}}(\mathbf{q},\lambda) = \frac{1}{2}||\mathbf{q}-\mathbf{q}^{(k)}||^2_{\mathbf{D}} + \left(\mathbf{c}(\mathbf{q}^{(k)}) + \nabla\mathbf{c}(\mathbf{q}^{(k)})^{\mathsf{T}}(\mathbf{q}-\mathbf{q}^{(k)})\right)^{\mathsf{T}}\lambda \tag{3.68}$$

and the solution $(\mathbf{q}^{(k+1)}, \lambda^{(k+1)})$ is characterized by vanishing partial derivatives of the Lagrangian:

$$\mathbf{D}\left(\mathbf{q}^{(k+1)} - \mathbf{q}^{(k)}\right) + \nabla\mathbf{c}(\mathbf{q}^{(k)})\lambda^{(k+1)} = \mathbf{0} \tag{3.69}$$

$$\mathbf{c}(\mathbf{q}^{(k)}) + \nabla\mathbf{c}(\mathbf{q}^{(k)})^{\mathsf{T}}(\mathbf{q}^{(k+1)} - \mathbf{q}^{(k)}) = \mathbf{0} \tag{3.70}$$

We can rearrange these equations into the familiar Karush-Kuhn-Tucker (KKT) matrix form:

$$\begin{bmatrix} \mathbf{D} & \nabla\mathbf{c}(\mathbf{q}^{(k)}) \\ \nabla\mathbf{c}(\mathbf{q}^{(k)})^{\mathsf{T}} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{q}^{(k+1)} - \mathbf{q}^{(k)} \\ \lambda^{(k+1)} \end{bmatrix} = - \begin{bmatrix} \mathbf{0} \\ \mathbf{c}(\mathbf{q}^{(k)}) \end{bmatrix} \tag{3.71}$$

This time, the matrix of this KKT system is a symmetric $(6m + 9) \times (6m + 9)$ matrix with special structure which can be exploited to achieve fast solves. Specifically, $\mathbf{D}$ is a constant diagonal matrix, which invites us to compute its Schur complement:

$$\left(\nabla\mathbf{c}(\mathbf{q}^{(k)})^{\mathsf{T}}\mathbf{D}^{-1}\nabla\mathbf{c}(\mathbf{q}^{(k)})\right)\lambda^{(k+1)} = \mathbf{c}(\mathbf{q}^{(k)}) \tag{3.72}$$

This is a dense 7×7 linear system, therefore, solving for $\lambda^{(k+1)}$ is very efficient. The "slowest" part is the matrix multiplication in $\nabla\mathbf{c}(\mathbf{q}^{(k)})^{\mathsf{T}}\mathbf{D}^{-1}\nabla\mathbf{c}(\mathbf{q}^{(k)})$ which has asymptotic complexity $O(m)$.

Because $\mathbf{D}$ is a diagonal matrix with positive elements, the Schur complement $\mathbf{S} = \nabla\mathbf{c}(\mathbf{q}^{(k)})^{\mathsf{T}}\mathbf{D}^{-1}\nabla\mathbf{c}(\mathbf{q}^{(k)})$ is guaranteed to be a positive semi-definite matrix. Furthermore, if the constraint Jacobian has full rank, $\mathbf{S}$ will be positive definite and thus invertible. For robustness, we have implemented a test by checking the determinant of $\mathbf{S}$ (which is fast because $\mathbf{S}$ is just a 7×7 matrix). If the determinant is close to zero, we apply regularization $\mathbf{S} + \epsilon\mathbf{I}$.

After we have computed $\lambda^{(k+1)}$, we compute:

$$\mathbf{q}^{(k+1)} := \mathbf{q}^{(k)} - \mathbf{D}^{-1}\nabla\mathbf{c}(\mathbf{q}^{(k)})\lambda^{(k+1)} \tag{3.73}$$

Eq. 3.73 has also asymptotic complexity $O(m)$ and all of the computations involve only dense numerical algebra subroutines with small dense matrices.

We stop iterating our projection when a state $\mathbf{q}^{(k)}$ is close enough to the energy-momentum manifold $\mathbf{c}(\mathbf{q}) = \mathbf{0}$. Specifically, we check the L^1-norm of the constraint function and stop if $|\mathbf{c}(\mathbf{q}^{(k)})|_1 < 10^{-7}$. In practice, the number of iterations for our projection method is quite small, see Table 3.3.

3.3.3 Attachments, collisions and damping

Attachments. External forces such as gravity can be included as part of the potential function E and do not require any special treatment. We also implemented attachment constraints (also known as "pin constraints"), which are zero rest-length springs with one endpoint controlled kinematically [114, 27]. The attachment constraints can be either fixed in space or moving, e.g., to enable user interaction with the simulated object.

Collisions. A similar strategy is employed to handle collisions. If we detect inter-penetrations, we first project the collided vertices to their closest surface point as in PBD [114]. However, this strategy can lead to oscillations if the projected vertices immediately try to return back to their penetrated states. We prevent these oscillations by temporarily including the following "collision potential" similar to repulsion springs [107] which repels the projected vertices from the penetrated configuration:

$$E_{\text{col}}(\mathbf{x}) = \begin{cases} -((\mathbf{S}\mathbf{x} - \mathbf{x}_{\text{surf}})^{\mathsf{T}}\mathbf{n})^3 & (\mathbf{S}\mathbf{x} - \mathbf{x}_{\text{surf}})^{\mathsf{T}}\mathbf{n} < 0 \\ 0 & \text{otherwise} \end{cases}$$

where $\mathbf{S}$ is a selector matrix extracting the colliding vertex from $\mathbf{x}$, $\mathbf{x}_{\text{surf}} \in \mathbb{R}^3$ is the surface point where the colliding vertex was projected and $\mathbf{n} \in \mathbb{R}^3$ is the surface normal at $\mathbf{x}_{\text{surf}}$. E_{col} can be interpreted as a cubic penalty function for inequality constraint $(\mathbf{S}\mathbf{x} - \mathbf{x}_{\text{surf}})^{\mathsf{T}}\mathbf{n} \geq 0$. We chose a cubic penalty because it yields a $\mathcal{C}^2$-continuous function E_{col}.

See Figure 3.22 for an example of collision handling.

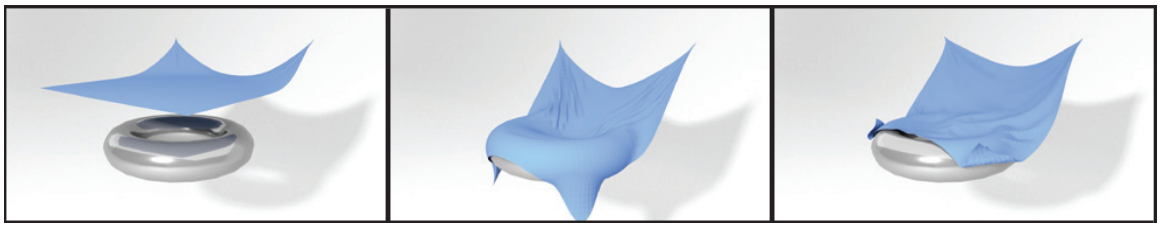

Figure 3.22: A piece of draping cloth colliding with a torus.

Damping. Our method can be combined with a variety of damping models. A simple yet useful example is "ether drag", which corresponds to multiplying all velocities by a coefficient less than one, potentially spatially varying. This slows down all motion, including rigid-body modes. If we want to model damping only due to internal friction, rigid-body modes of motion should not be affected. This can be achieved by more

sophisticated momentum-conserving damping models such as [9] and [80] integrated in a separate implicit step [32]. However, the computing overhead of these approaches may be too high for real-time physics. Instead, we implemented the simple yet fast momentum-conserving damping method used in Position Based Dynamics [114] which explicitly factors out global linear and angular velocities and damps out only the residual velocities corresponding to non-rigid motion.

3.3.4 Results

Example	#Verts.	#Elems.	Integration Method	Integration Time	FEPR #Iters.	FEPR. Time
Human	571	1886	IM	30 ms	3.0	3 ms
Cube	602	1668	IM	18 ms	5.2	7 ms
Ball	889	1772	IM	30 ms	16.3	24 ms
Squirrel	1507	5330	IM	71 ms	2.7	11 ms
Hippo	2387	8879	BE	190 ms	8.9	28 ms
Hippo	2387	8879	BDF-2	208 ms	11.2	35 ms
Trampoline	3721	18120	BE	53 ms	9.1	14 ms
Cloth	3721	18120	BE	92 ms	4.1	6 ms
Cactus	5261	17631	IM	211 ms	2.0	21 ms
Jelly	6073	22054	BE	350 ms	9.2	119 ms
Octopus	7502	30010	IM	1864 ms	5.1	120 ms
Rabbit	14261	52090	BDF-2	1868 ms	4.2	180 ms

Table 3.3: Statistics for all our testing scenarios. The reported times and numbers of iterations are averages over the entire simulation run. Both the "Integration Time" and the "FEPR Time" columns report total time, i.e., time for *all* iterations.

Performance. Table 3.3 summarizes our testing scenarios and run times for both 1) an iterative solver of an integration rule and 2) our fast energy-projection (FEPR) algorithm. All experiments were executed on an Intel i7-6700HQ CPU at 2.6 GHz. All scenarios are simulated using a fixed time step $h = 1/30$ seconds. We tested different iterative solvers for the nonlinear optimization problem (see the Appendix for implementation details). In summary, we experimented with an L-BFGS-accelerated Projective Dynamics solver [96], eXtended Position Based Dynamics (XPBD) [103] and a linearized solve analogous to one iteration of Newton's method [9]. With implicit midpoint we also tried to compute a fully converged solution because we wanted to test whether the implicit midpoint instabilities

are caused by an approximate solve. We found this was not the case – implicit midpoint explodes even if the update rules are resolved to machine-precision accuracy.

Our fast energy projection adds only a small computing overhead, typically around 10%. There are a few exceptions, for example, stiffer systems require more iterations resulting in longer runtimes. However, even these challenging examples are still faster than the solve of the integration rule.

Artificial damping. Many integrators commonly used in real-time simulations are dissipative, i.e., introduce artificial numerical damping, such as backward Euler. While this improves stability, a side-effect is that the resulting motions can appear very damped. Figure 3.23 shows a simulation where we shook a plate that had a delicious serving of Jell-O. Running this simulation using backward Euler resulted in a very rigid-looking snack, even though we did not add any damping to the simulation – all of the damping is due to backward Euler. When we applied our FEPR post-processing, we managed to restore the natural wiggling of the gelatin. In fact we added a small amount of damping using the PBD damping method (see Section 3.3.3), because real-world materials dissipate energy. However, aided by FEPR, this dissipation is user-controllable, which is not the case of backward Euler where the artificial damping is due to the integration rule itself and depends on the time step size and other simulation parameters.

A very popular method to simulate deformable objects is Position-Based Dynamics (PBD) [114]. Recently, Macklin et al. [103] introduced a small but powerful modification called eXtended Position-Based Dynamics (XPBD) which correctly accounts for material stiffness and can thus theoretically converge to an exact backward Euler solution. However, in practice, XPBD is rarely iterated until convergence. Interestingly, numerical dissipation is not the main issue of early-terminated XPBD. Instead, XPBD can produce motion with artificially increased flexibility of the simulated object, as we can see in the *trampoline* example in Figure 3.24. Applying FEPR after XPBD results in improved wrinkle formation. It is important to note that our method cannot correct all of the errors introduced by an underlying simulator such as XPBD. In the *trampoline* example, the increased "stretchiness" caused by under-solved XPBD is still present even after FEPR, even though we make the cloth more wrinkled and its motion more vivid.

The artificial numerical damping of backward Euler can be mitigated by its second order

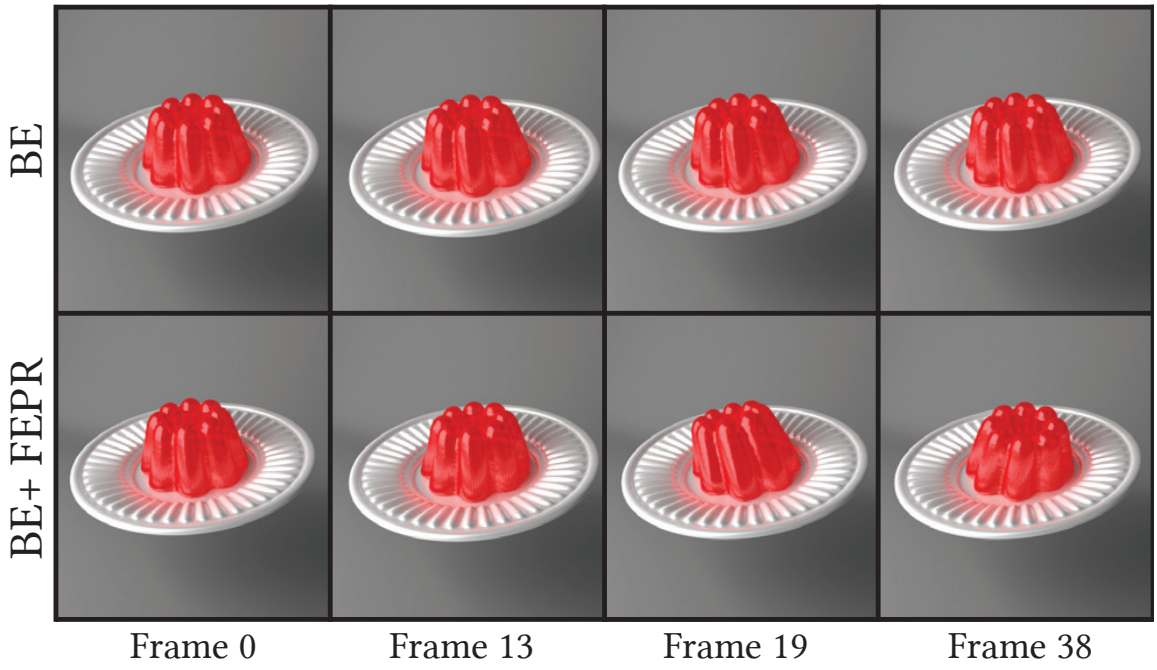

Figure 3.23: A simulation of a serving of Jell-O being shaken. The motion produced by backward Euler (BE) looks rigid; adding our method (FEPR) produces vivid motion.

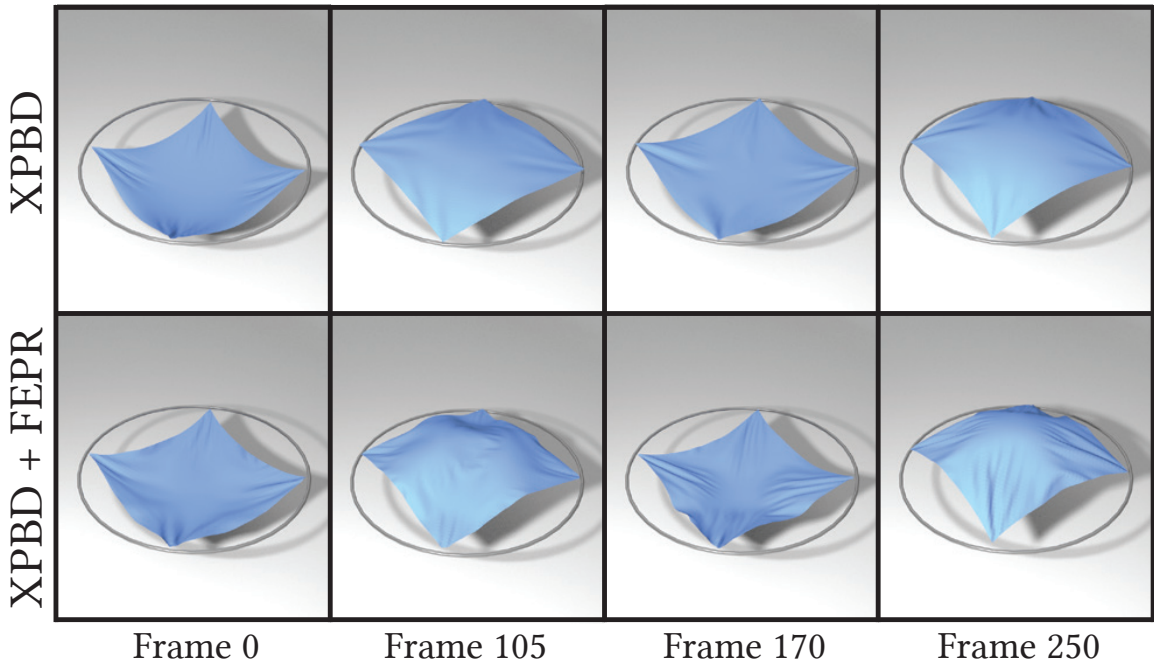

Figure 3.24: FEPR (our method) applied on top of eXtended Position Based Dynamics leads to improved wrinkle formation.

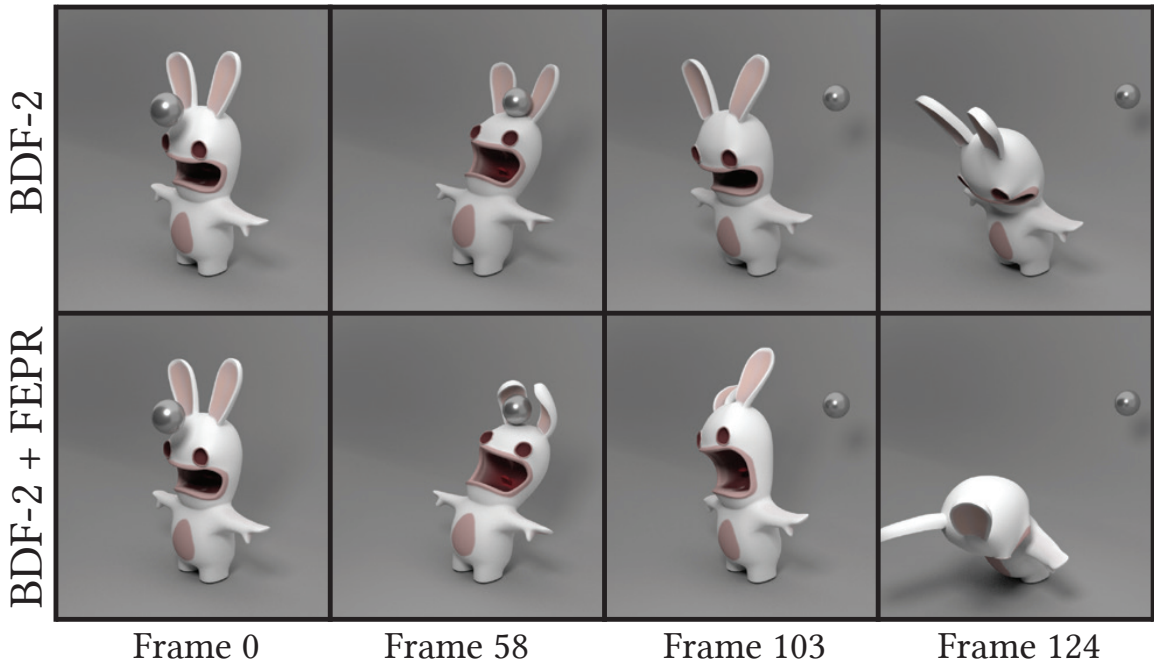

Figure 3.25: Even though BDF-2 produces a nice-looking animation, adding FEPR makes it even more lively and humorous.

version, BDF-2, which produces more vivid motion than backward Euler, but still contains numerical damping. In Figure 3.25, we took a comical rabbit character and tortured it by pelting it with a cannonball. While BDF-2 produced a nicely swinging rabbit, adding FEPR resulted in much more vivid animation and created a very panicked-looking rabbit.

Explosions. Instabilities are potentially even more problematic than numerical damping, especially in real-time simulations. Even though integrators such as backward Euler are very stable due to their dissipative properties discussed above, this is not guaranteed if the implicit time stepping rules (nonlinear equations) are not solved exactly. A classical approximate solve of implicit integration is linearization of the non-linear equations, analogous to an undamped step of Newton's method. This approach is common in off-line simulators [9], but also in real-time applications such as games [125]. Unfortunately, this approach can introduce instabilities ("explosions"), as demonstrated on a *hippo* example in Figure 3.26. The simulation explodes due to the nonlinear backward Euler equations not being accurately solved. Applying FEPR stabilizes the simulation due to energy conservation and the hippo survives.

Previous energy-conserving methods. Discrete gradient methods preserve energy and

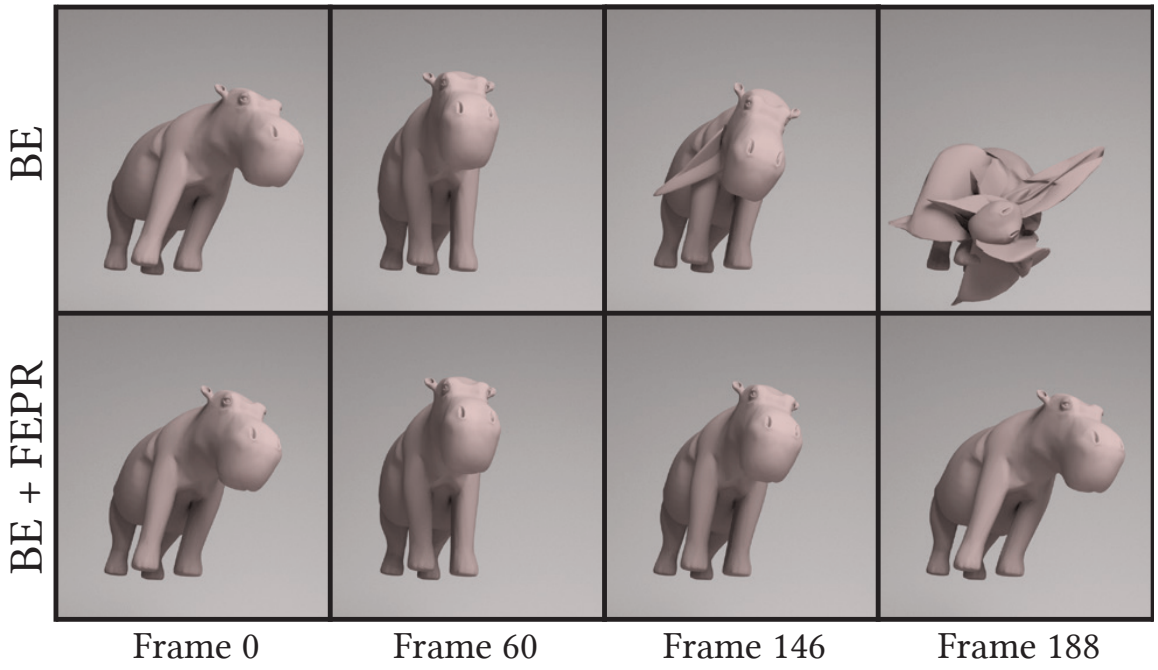

Figure 3.26: A swaying hippo simulated with linearized backward Euler explodes due to linearization errors. FEPR prevents the explosions and produces plausible motion.

momentum by construction. There are several variants of these methods; for simulations of deformable objects, the most common is the midpoint discrete gradient ([62, 63]). The midpoint discrete gradient has found use in mechanical engineering and can produce good results. Unfortunately, this is only the case with sufficiently small time steps, because the midpoint discrete gradient requires solving a root-finding problem $\mathbf{g}(\mathbf{x}, \mathbf{v}) = \mathbf{0}$. With larger time steps, the root finder can get stuck at a local minimum, failing to find a root. This can be observed even in a didactic one dimensional example similar to harmonic oscillator. Specifically, we ran a simulation using a quartic potential $E(x) = 128x^4$ using a time step of $h = 33$ ms and found a time step where the root solver failed. Figure 3.27 shows a contour plot of the merit function $\mathbf{g}(\mathbf{x}, \mathbf{v})^\mathsf{T}\mathbf{g}(\mathbf{x}, \mathbf{v})$. The goal of the root finder is to find zero values of the merit function, corresponding to roots $\mathbf{g}(\mathbf{x}, \mathbf{v}) = \mathbf{0}$. Unfortunately, depending on the initial guess, the root solver can get stuck at a local minimum of the merit function and fail. With the standard initial guess for implicit midpoint, $\mathbf{y}_n = \mathbf{x}_n + h\mathbf{v}_n$ for the position and $\mathbf{v}_n$ for the velocity, the root finder fails as illustrated in Figure 3.27 (left). This problem can be avoided by reducing the time step five times, to $h = 6.6$ ms, as shown in Figure 3.27 (right). Intuitively, this helps because with smaller time steps, the equations of the implicit

update rule are "less non-linear" and local minima problems are less likely to happen. In

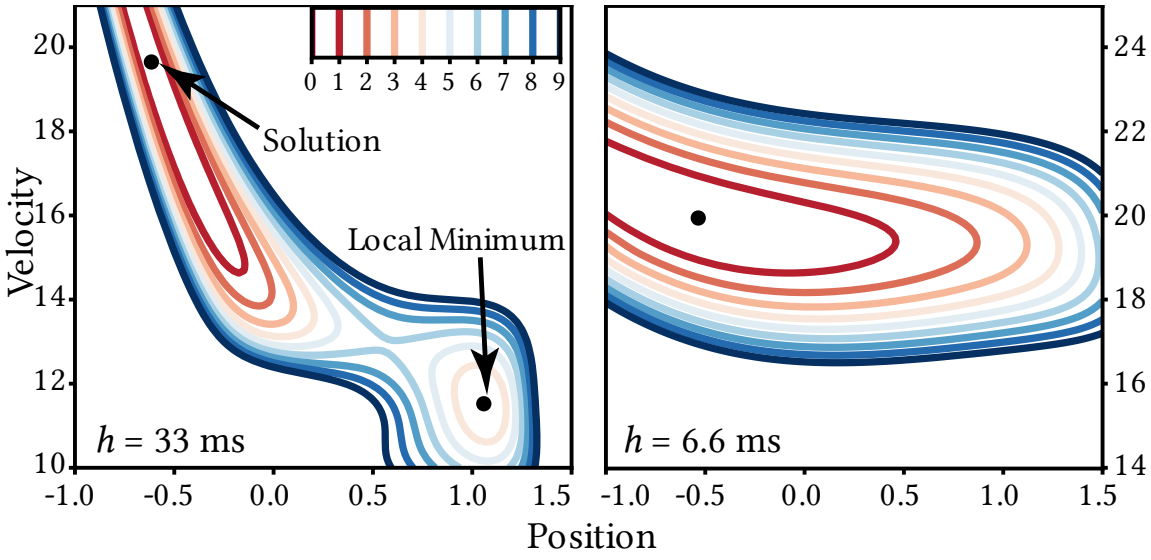

Figure 3.27: A contour plot of the merit function for the discrete gradient method for a time step of 33 ms (left), where the root solver got stuck at a local minimum. With time step reduced to 6.6 ms (right) the root solver succeeded.

practical simulations, this local minimum problem can produce significant visual artifacts. We demonstrate this in Figure 3.28, showing a ball falling under gravity, evaluated using our method and the midpoint discrete gradient. At a time step of $h = 33$ ms, our method results in the ball bouncing up and down without any issues. The midpoint discrete gradient at the same time step runs into local minimum issues early on in the simulation and produces implausible results. Decreasing the time step by a factor of ten to $h = 3.3$ ms helps, but the midpoint discrete gradient still runs into trouble towards the end of the simulation run, see Figure 3.28 (middle row).

The method proposed in Section 3.2 can introduce damping in the momentum. We demonstrate this on a spinning cube example in Figure 3.29. In this example, we can see that when implicit midpoint overshoots the total energy, explosion is prevented by blending with backward Euler, which removes the excessive energy due to its numerical damping properties. Unfortunately, as a side effect, the blending of backward Euler also reduces the angular momentum and slows down the global motion. Our method is an add-on on top of any integrator and avoids this problem by exactly projecting energy while also carefully taking into account both linear and angular momentum.

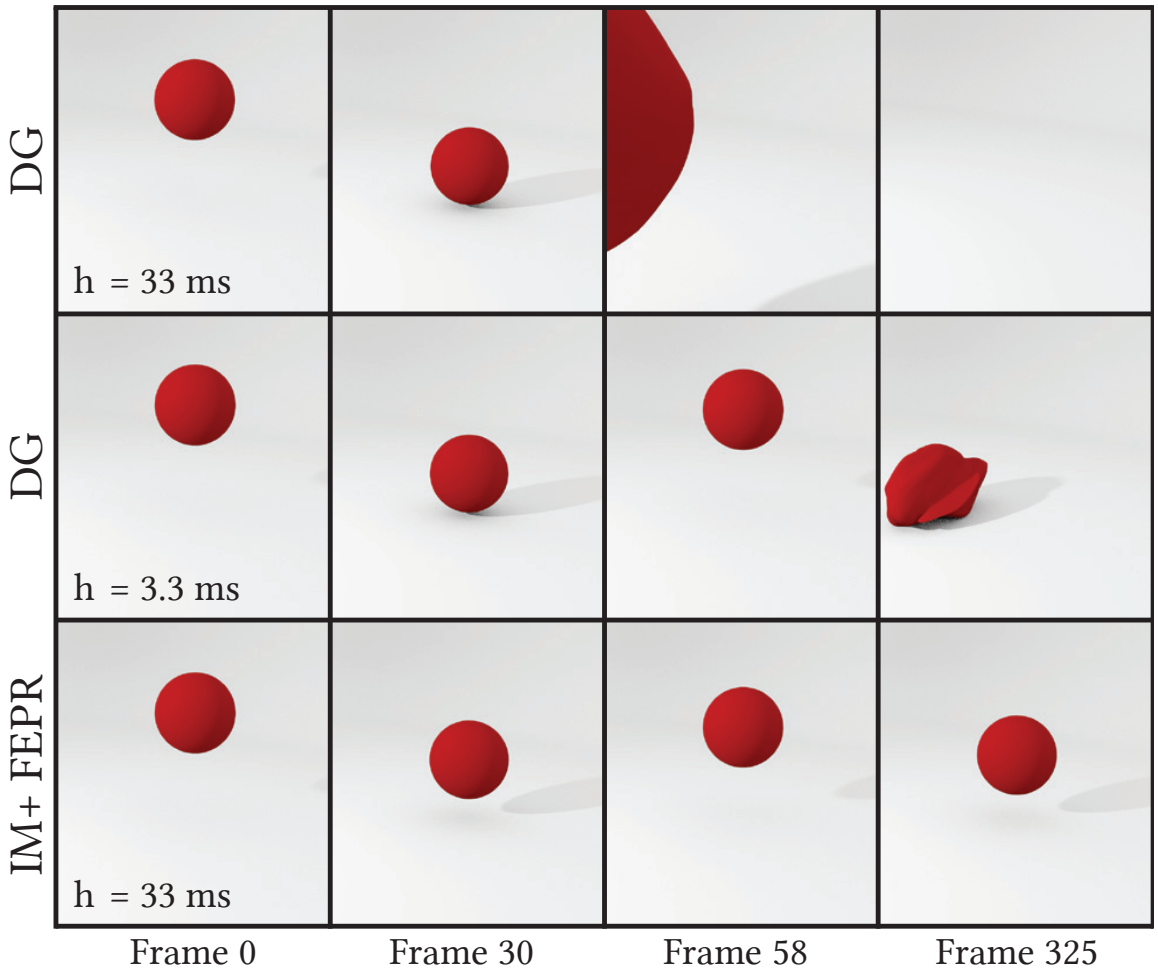

Figure 3.28: The root solver used to evaluate a midpoint discrete gradient update rule can get stuck at a local minimum and produce incorrect results. This eventually happens even with ten times smaller time steps (middle row). Our method works even with large time steps and produces plausible animation (bottom).

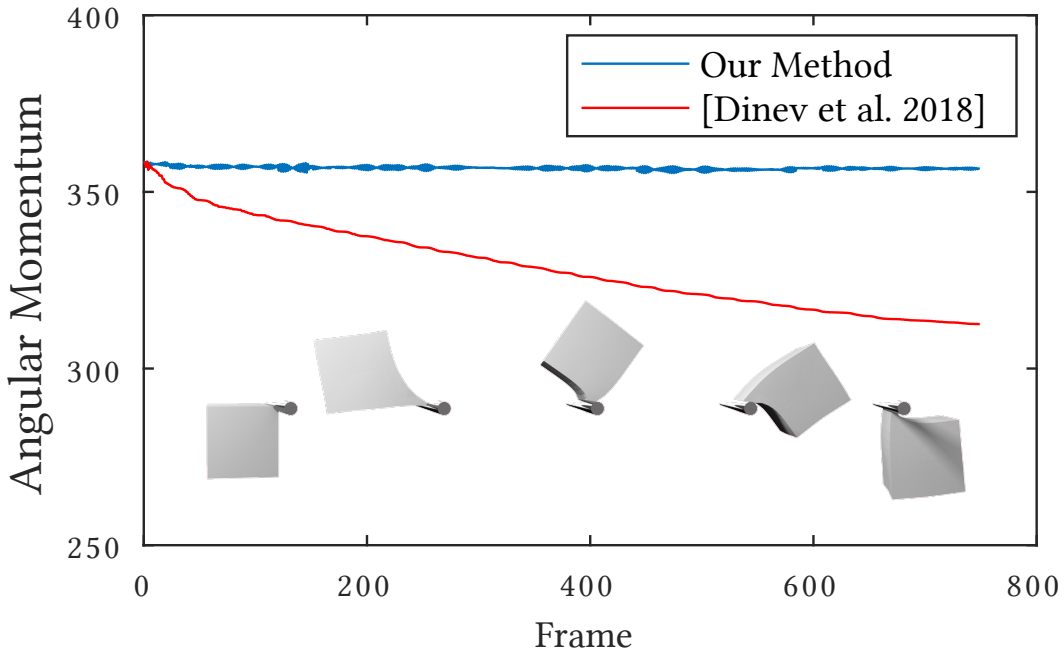

Figure 3.29: Angular momentum of a spinning cube around its spinning axis. Unlike Section 3.2, FEPR preserves the global rotational motion.

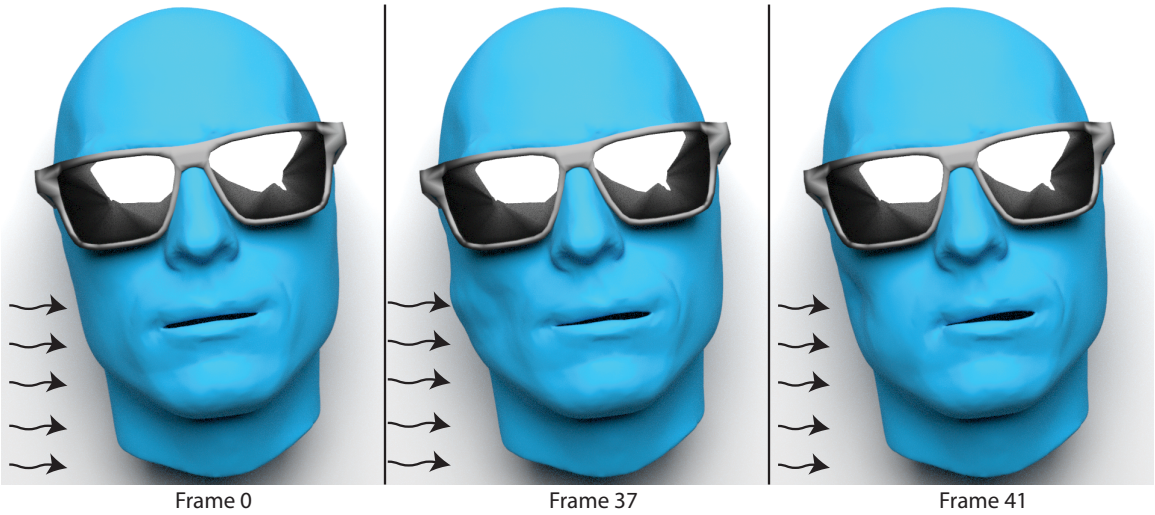

Figure 3.30: Applying a wind force to a face now produces the vivid motion and ripples we expect.

Evaluation Criteria. In graphics, we usually evaluate methods qualitatively based on their "visual plausibility". Instead, one could evaluate the accuracy of numerical integrators by comparing them to the ground-truth solutions computed by using a very small timestep or an adaptive timestepping scheme. The methods proposed in sections Section 3.2 and Section 3.3 do not claim to produce *accurate* solutions and are instead targeting visual plausibility in a real-time domain. Both methods do, however, converge: as the timestep $h \to 0$, the result of the underlying integrator will already produce a solution that satisfies the energy and momentum constraints, producing a solution to both projection steps.

Applying to a face. We can now apply this method on top of a backward Euler integrator to revisit our face-in-the-wind example. Figure 3.30 shows the result of applying a wind force (left to right) to a static face, using a backward Euler/Projective Dynamics integrator as a base and with our FEPR method applied to it. We have also included damping (otherwise the wind would infinitely insert energy into the system), which is the situation in the real world. We can see that this produces lively motion and big deformations, as we would expect from the flesh. Note that even though we have a damping model here, it still produces vivid motion as opposed to the numerical damping from backward Euler.

CHAPTER 4

FACIAL MUSCLE SIMULATION

The modeling of the human body is a very important problem in computer graphics, with wide applications ranging from the entertainment industry to medicine. In recent years, advances in facial tracking and capture technologies [12, 15, 14] have achieved impressive results and are becoming widely used in the film industry. However, anatomical modelling of the face is particularly difficult, due to the high levels of nonlinear deformations exhibited on the surface of the skin caused by complex interactions between the underlying facial muscles and the passive tissue (e.g. fat).

Traditional approaches to creating facial animations usually rely on direct deformation models which only take into account the surface of the face, such as blendshapes [91]. These techniques are favored for their ease of use and extensive support, but they are not without shortcomings. While they can create believable facial expressions, it is difficult to add physics-based effects to them (i.e., inertia, gravity, collisions). Furthermore, it is possible to produce unrealistic expressions, requiring correctives to fix artifacts.

Physics-based models attempt to remedy these artifacts by attempting to mimic the real-world biomechanics of facial expressions generation. Instead directly using the surface of the face, anatomical face models attempt to simulate the underlying muscle structure which causes the surface deformations [132]. This is a very challenging problem due to the fact that it is difficult to acquire such information for a given subject: all of the muscles and fat lie below the skin and require advanced medical imaging machines to observe, making it impractical for general use. Thus, physics-based models usually require a significant manual effort to create a muscle structure for a given subject. While this can produce good results which are used in high-budget feature films, it requires many man-hours on the part of technical artists to create a good model for just one subject.

Recent works have attempted to remedy this problem by attempting to infer the

underlying muscle structure [71, 70] using only surface scans. This is accomplished by setting up an inverse solve which tries and finds the correct muscle activations that are able to fit a given expression in a scan. This makes the method more practical for general use, but sacrifices the physical interpretations of the muscles by over-parameterizing them in order to fit the scans. More concretely, the number of parameters are a factor for the number of volumetric elements (i.e., tetrahedra); each element carries acts as an independent muscle.

In this work, we attempt to alleviate this issue by drastically reducing the number of parameters in the facial muscle model to more closely match the real-world muscles. We accomplish this by observing that any volumetric element (i.e., tetrahedra) in the face can contain several muscles as well as passive tissue. We extract this geometric information into a separate variable from the muscle activations, telling us how all of these elements are blended together. We then solve for the blending variable by simultaneously fitting several different target scans. By separating out this blending information, the muscle activations we solve for are closer to the real-world mechanics of the face; the number of unknowns is a function of number of muscles instead of number of elements. We then show that this blending variable can be re-used to successfully fit new, unseen expressions scans of the same subject.

This method retains the advantages of previous methods in that it is still fully automatic and does not require any manual muscle construction, but by extracting the subject-specific muscle geometry from the muscle activations we are able to greatly reduce the space of the activations. A bonus side-effect of this is that the activation-only solves are much faster due to the drastically lower number of unknowns. The muscle activations can also still used as keyframes to create "volumetric blendshapes" that are responsive to physics-based effects such as wind and changes to our subject's physical parameters (i.e., adding fat). We believe that solving for the muscle geometry is an important step towards having a full personalized physics-based model that is easily obtainable for any given subject, without requiring significant manual effort.

4.1 Related Work

Facial animation has been an important part of computer graphics research since the seminal works of [24, 146]. Human facial expressions are extremely important in

communication [47], and accurate representations of the human face is vital in both the entertainment industry and the medical field.

Data-Driven Methods. The traditional facial animation technique used by artists is geometry-based blendshapes [92, 91, 69, 126, 163, 94]. This allows for artists to create facial expressions geometrically blending several target expressions. This approach has the advantage of being simple to implement, intuitive for artists, and capable of creating a large variety of facial expressions. However, the linear blending of the surface can produce some non-physical shapes and has difficulty handing phenomena such as contact. Physics-based approaches have been used to augment geometric approaches. [100] augmented the blendshape framework with mass-spring systems, while [11] added a simulated surface, and [83, 81] added a volumetric layer on top of a blendshape model.

Blendshapes are usually created by employing capture techniques to get the target expressions. Facial capture systems have seen a significant increase in quality in recent years and are widely used in the entertainment industry. Marker-based techniques [21] and recently dense markerless techniques [30, 12, 15, 56, 86, 150, 1] can produce a very accurate reconstruction of a facial performance in multiview [60] and even monocular [129, 33] capture setups. Modern system can also accurately capture traditionally difficult areas such as the eyelids [20], eyes [19, 18], hair [13, 99, 67], teeth [158], lips [147, 57, 42], and skin microstructure [115, 64]. Anatomy-based techniques have also been used to improve the tracking, such as skull transformations [14], jaw tracking [167, 162] and deformation models[159].

Physics-based Approaches. Anatomical modeling of the body is an active area of research with many simulations platforms, such as ArtiSynth [98], SOFA [4], FEBio [101], and Abaqus [26]. Extracting the biomechanical parameters of the biological tissue is an ongoing problem in biomechanics, with many measurement [148] and models [156, 116] having been developed. Instead of excising the soft tissue and measuring the forces via torsion or suction tests, computer vision techniques can be applied to estimate material parameters [123, 124, 22, 23] by visually studying the deformation.

Many muscle models have been proposed for fully-body animations [117, 140, 142, 141, 90]. Various numerical strategies have been developed, such as Eulerian-on-Lagrangian-based [51, 52] and projective-dynamics-based [27, 127, 74] approaches. Facial muscles

are fundamentally different from skeletal muscles such as the biceps brachii [25], posing significant challenges due to their small, thin nature and complicated attachments. Muscle simulation for bodies usually studies the muscles' effects on the bone, while facial simulations try to explain the subtle deformations on the surface of the skin. Building person-specific muscle models has been an active field of research since the seminal works of [132], whose method requires significant manual sculpting of the muscles. Similarly, [160] used MRI scans to extract muscle information from a subject, again requiring significant manual tuning and editing to create a useful model. [39] introduced a model that allowed artists to manually sculpt expressions by editing muscle activations, and this approach was improved by [88, 8]. [136] introduced an automated method for constructing a person's geometry using CT data, which is not trivial to obtain. [38] introduced a method to geometrically adapt muscles from a template to a target using only the neutral scans, however they do not use target expressions scans to solve for muscle activations. [71] introduced a physics-based equivalent to blendshapes, which overcomes blendshapes' limitations in handling physics-based phenomena such as collisions and gravity. [70] extended this framework by adding a rotationally-invariant muscle activation model. This has been further extended by [75] to incorporate solving for heterogeneous material parameters and more complicated phenomena such as pre-strain. However, these activations lose their physical meaning, treating each tet as an independent muscle.

4.2 Method

4.2.1 Data preparation

We begin with a template face model, which includes the bones (skull and mandible), muscles, and a flesh mesh, shown in Figure 4.1 (top left). We then capture a series of registered scans of our subject (Figure 4.1, top right), including a neutral expression which we can use with the method of Anatomy Transfer[3] to transfer the anatomy of the template to our subject, adapting the meshes representing the bones, muscles, and flesh to fit our subject. Tetgen [130] is then used to create a tetrahedral mesh of the flesh (Figure 4.1, bottom). Similar to [70], we use the tetrahedral flesh mesh and the muscle meshes to find "active" tetrahedra. However, instead of storing a binary "active" or "passive" value, we store a numeric value representing which specific muscle a tetrahedron contains. Additionally,

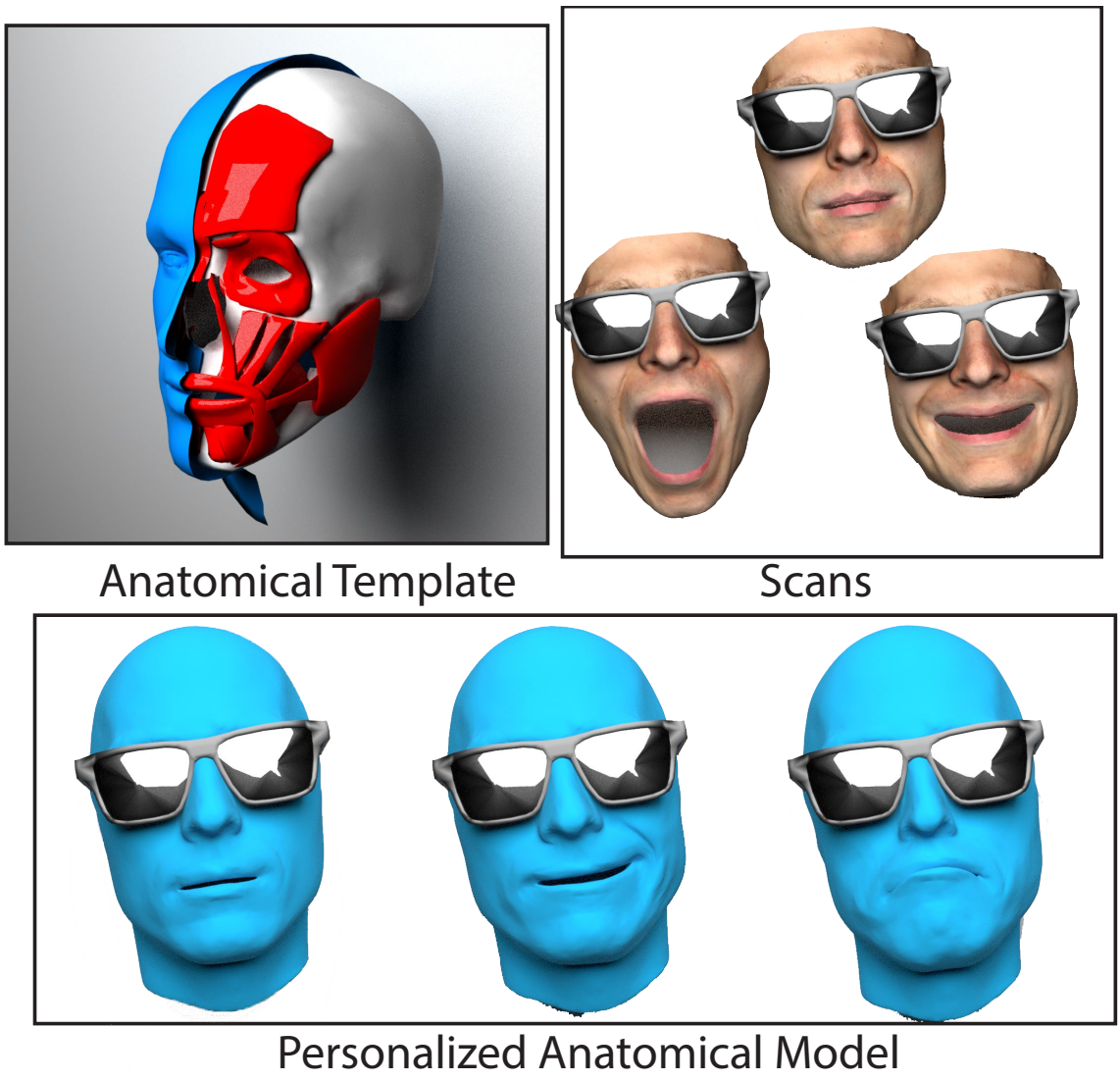

Figure 4.1: An animation may be produced by linearly interpolating the activations of two expressions.

since the muscles obtained from [3] are only an estimate and can differ from the subject's actual muscle geometry, we also store muscles whose meshes are located in a 1cm radius of the tetrahedron, to account for the potential of these muscles to shift around in a different subject compared to our template. This information is used as an initialization and will be valuable in Section 4.2.4.

4.2.2 Forward simulation of face models

We begin with a tetrahedral face mesh of a subject in the rest pose, with p vertices and n tetrahedra. We can describe the activation of a tetrahedron using an augmented

finite-element-like elastic potential, such as corotated linear elasticity [70]:

$$E_{act}(\mathbf{x}, \mathbf{A}) = \sum_{i=1}^{n} \min_{\mathbf{R}_i \in SO3} \frac{\mu_i}{v_i} ||\mathbf{F}_i(\mathbf{x}, \mathbf{x}_{rest}) - \mathbf{R}_i \mathcal{S}(\mathbf{a}_i))||_{\mathbf{F}}^2 + \frac{\lambda_i}{v_i} Tr^2(\mathbf{S}(\mathbf{F}_i(\mathbf{x}, \mathbf{x}_{rest})) - \mathbf{I}) \tag{4.1}$$

where $\mathbf{x} \in \mathbb{R}^{p \times 3}$ is a matrix of the mesh vertex positions in the deformed pose, $\mathbf{a}_i \in \mathbb{R}^6$ is a vector for the activation degrees of freedom for tetrahedron i (which we will be solving for in Section 4.2.3), $\mathcal{S} : \mathbb{R}^6 \rightarrow \mathbb{R}^{3 \times 3}$ is an operator which takes a 6 DoF vector and returns a symmetric matrix, $\mathbf{F}_i(\mathbf{x}, \mathbf{x}_{rest})$ is the deformation gradient for the i-th tetrahedron. Matrices $\mathbf{R}_i \in \mathbb{R}^{3 \times 3}$ and $\mathbf{S} \in \mathbb{R}^{3 \times 3}$ come from the polar decomposition of the deformation gradient $\mathbf{F}_i = \mathbf{R}_i \mathbf{S}_i$, and v_i is the rest volume of i-th tetrahedra, λ_i, μ_i are Lame's first and second parameter, respectively, and $|| \cdot ||_F$ denotes the Frobenius norm. We have added a linear elasticity-based *lambda* term to enforce some notion of volume conservation.

To determine vertex positions of the forward simulation result, we need to solve the following optimization problem, keeping $\mathbf{A}$ constant:

$$\min_{\mathbf{x}} \quad E_{act}(\mathbf{x}, \mathbf{A}) + E_{ext}(\mathbf{x}), \tag{4.2a}$$

$$\text{s.t.} \quad \mathbf{c}(\mathbf{x}, \mathbf{b}) = 0, \tag{4.2b}$$

where $E_{ext}(\mathbf{x})$ is external energy due to forces such as gravity. The constraints $\mathbf{c}(\mathbf{x}, \mathbf{b})$ correspond to physiological constraints like muscles attaching to the bones. Similar to [70], we model and solve for the jaw kinematics using a 5-DoF rigid transformation model which allows for rotation along only 2 axes and translation along 3 and stacks the five total degrees of freedom in a vector $\mathbf{b} \in \mathbb{R}^5$.

4.2.3 Inverse model

While the forward problem is simple to solve, obtaining the activation matrix $\mathbf{A}$ is difficult. We want to find an $\mathbf{A}$ matrix such that we are able to explain target scans that we have taken of a subject. We introduce a target energy:

$$E_{target}(\mathbf{x}, \mathbf{t}) = ||\mathbf{M}\mathbf{x} - \mathbf{N}\mathbf{t}||_{\mathbf{F}}^2 \tag{4.3}$$

where $\mathbf{x} \in \mathbb{R}^{p\times 3}$ is our tetrahedral mesh's vertices and $\mathbf{t} \in \mathbb{R}^{o\times 3}$ are vertex positions from our target scan with o vertices. The matrices $\mathbf{M} \in \mathbb{R}^{c\times p}$, $\mathbf{N} \in \mathbb{R}^{c\times o}$ are selection matrices that select vertices in $\mathbf{x}$ that correspond to vertices in $\mathbf{t}$, where c is the number of such correspondences. We then set up the following optimization problem to solve for the $\mathbf{A}$ matrix:

$$\min_{\mathbf{x},\mathbf{b},\mathbf{A}} \quad E_{target}(\mathbf{x},\mathbf{t}) + E_{reg}(\mathbf{x},\mathbf{A}) \tag{4.4a}$$

$$\text{s.t.} \quad \nabla_{\mathbf{x}} E_{act}(\mathbf{x},\mathbf{A}) + \nabla_{\mathbf{x}} E_{ext}(\mathbf{x}) = 0 \tag{4.4b}$$

$$c(\mathbf{x},\mathbf{b}) = 0 \tag{4.4c}$$

Constraint 4.4b requires quasi-static equilibrium and constraint 4.4c is the same physiological constraint as in Eq. 4.2. Since $p >> o$, the problem is highly overparameterized and regularization $E_{reg}(\mathbf{x},\mathbf{A})$ is necessary as we will discuss in Section 4.2.6. Solving this constrained minimization problem will produce a matrix $\mathbf{A}$ that when used to solve Eq. 4.2 will produce an $\mathbf{x}$ as close as possible to the target $\mathbf{t}$.

4.2.4 Muscle Geometry Matrix S

The key difference in our method is the parameterization of $\mathbf{A}$. Previous work parameterized the $\mathbf{A}$ matrix by allowing 9 [71] or 6 [70] degrees of freedom for every tetrahedron and constructing a matrix from those degrees of freedom. This gives the model great flexibility and allows it to explain almost any given target at the cost of losing the physiological meaning of the matrix $\mathbf{A}$: each tetrahedron acts as its own independent "muscle" and each "muscle" can be transformed in any symmetric way. This is far from the real-world situation, where facial expressions are created using only a few muscles.

To reduce the parameter space, first we will change the semantic meaning of $\mathbf{A}$: instead of having an $\mathbf{A}_i \in \mathbb{R}^{3\times 3}$ matrix per tetrahedron, we will have an $\mathbf{A}_i \in \mathbb{R}^{3\times 3}$ matrix per facial muscle. Since $\mathbf{A}_i$ is a symmetric matrix constructed from 6 variables, we will instead store it as a 6×1 vector $\mathbf{a}_i$. The global activation vector is $\mathbf{a} \in \mathbb{R}^{6m}$, where m is the number of muscles.

These activations are now independent of the tetrahedra. Intuitively, we want tetrahedra to be influenced by nearby muscles. Since facial muscles are close to each other and

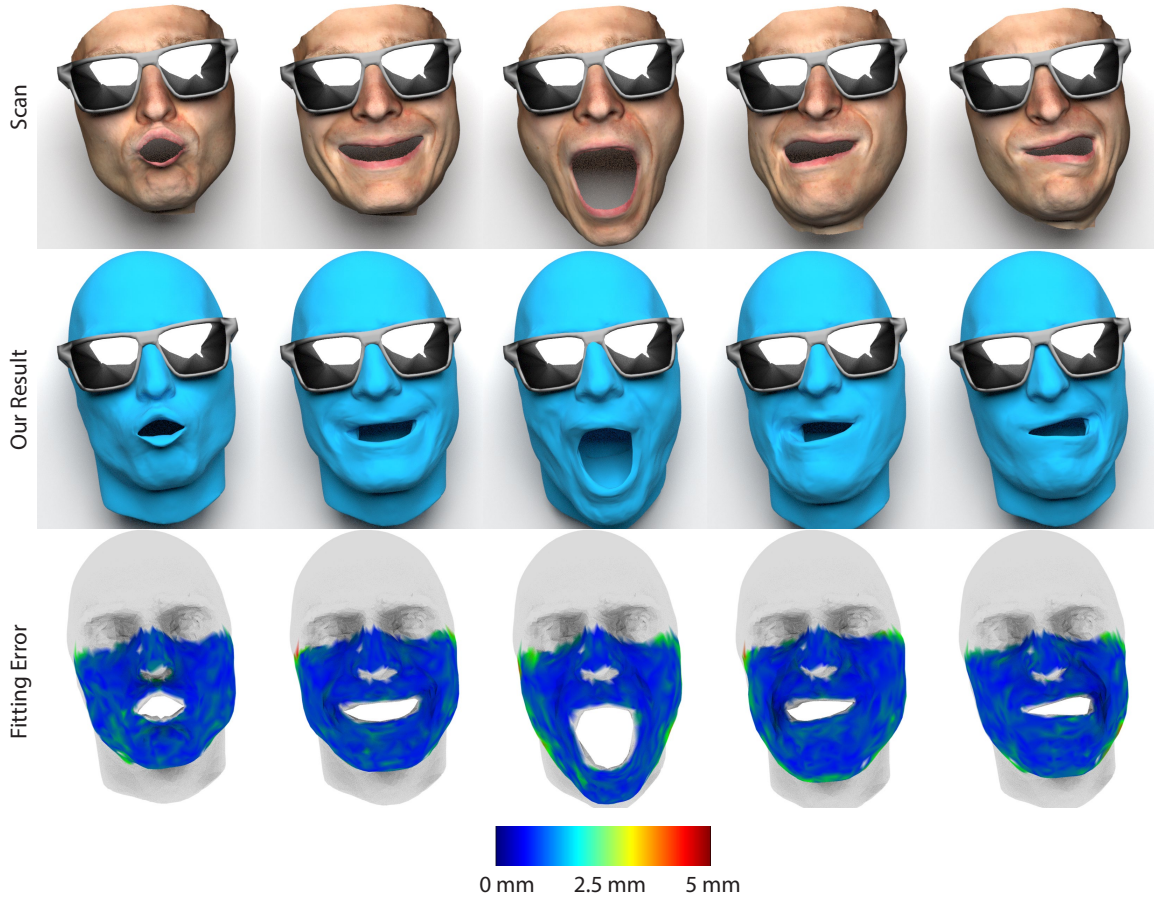

Figure 4.2: The input scans used for the multi-target fitting (top), the mesh produced by our method (middle), and a visualization of the point-to-point fitting error (bottom).

even overlap, tetrahedra need to be able to combine nearby activations. We introduce a per-tetrahedron vector $\mathbf{s}_i \in \mathbb{R}^m$, where every value s_j represents a weight for how much this tetrahedron is influenced by muscle j. A tetrahedron's total activation $\mathbf{a}_i$ is then composed of

$$\mathbf{a}_i = \sum_j s_{ij} \mathbf{a}_j \tag{4.5}$$

where s_{ij} is tetrahedron i's weight for muscle j and $\mathbf{a}_j$ is the activation for this same muscle. This makes a tetrahedron's individual activation a linear combination of the muscle activations. In order for this weighted sum to be sensible, we must impose the following constraints on $\mathbf{s}_i$: $0 \leq s_{ij} \leq 1$, and $\sum_j^m s_{ij} = 1$. Furthermore, we will also enforce spatial sparsity on this vector: we only allow muscles that are within 1cm of this tetrahedron

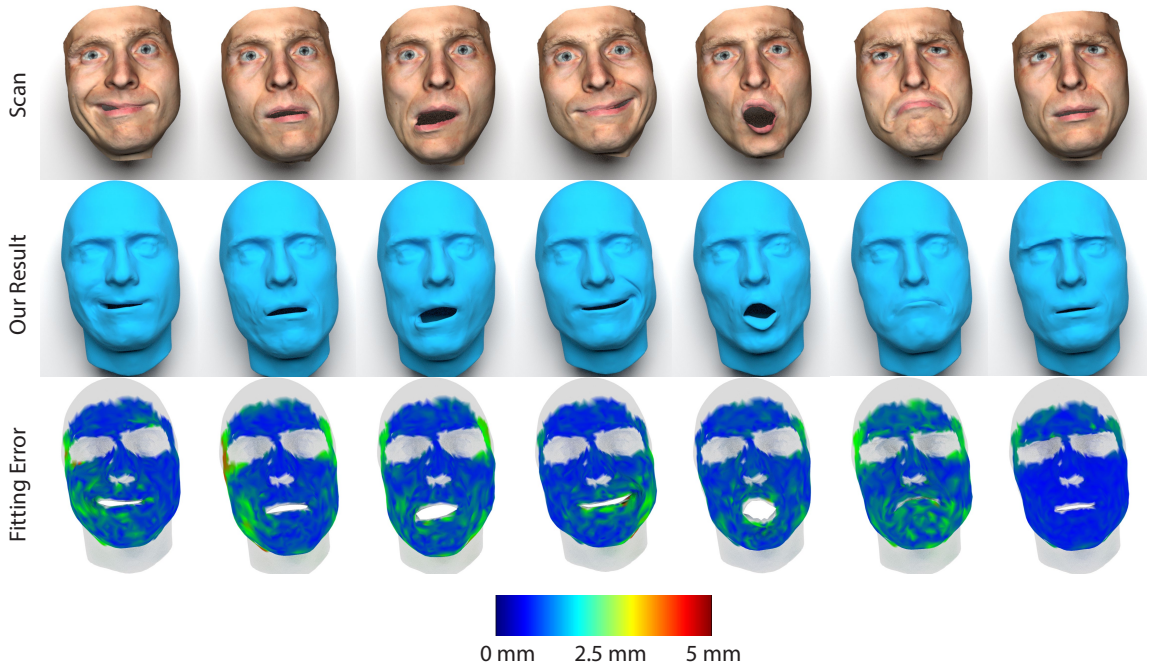

Figure 4.3: A set of previously unseen scans scans (top), the mesh produced by our method using only an activation solve (middle), and a visualization of the point-to-point fitting error (bottom).

to take non-zero values. This prevents faraway muscles from influencing a tetrahedron (i.e., a muscle on the right side of the face can not directly influence a tetrahedron located on the left side). Finally, using the same operator $\mathcal{S}$ from Eq. 4.1, we can express our active muscle model as:

$$E_{act}(\mathbf{x}, \mathbf{A}) = \sum_{i=1}^{n} \min_{\mathbf{R}_i \in SO3} \frac{\mu_i}{v_i} ||\mathbf{F}_i(\mathbf{x}, \mathbf{x}_{rest}) - \mathbf{R}_i \mathcal{S}\left(\sum_{j}^{m} s_{ij}\mathbf{a}_j\right)||_{\mathbf{F}}^2 + \frac{\lambda_i}{v_i} Tr^2(\mathbf{S}(\mathbf{F}_i(\mathbf{x}, \mathbf{x}_{rest})) - \mathbf{I}) \tag{4.6}$$

where i indexes tetrahedra and j indexes muscles (s_{ij} denotes coefficient j of vector $\mathbf{s}_i$). What we have accomplished here is moving many of the degrees of freedom into a blending matrix $\mathbf{S} \in \mathbb{R}^{n \times m}$ (where n is the number of tetrahedra and m is the number of muscles), which encodes the geometric blending of the muscles. For a single target, this does not accomplish our goal of reducing the degrees of freedom. However, this $\mathbf{S}$ matrix is a *subject-specific* parameter, akin to the stiffnesses λ, μ: in contrast to *expression-specific* parameters. Once an $\mathbf{S}$ matrix for a particular subject is found, it can be re-used for all expressions generated

by the same subject. For new expressions, the only expression-specific parameters are the positions $\mathbf{x}$ and the muscle activations $\mathbf{a}$.

Passive Tissue. To handle passive tissues, we need to make only a small modification to the $\mathbf{S}$ and $\mathbf{a}$ variables. Passive tissue can be modeled by ensuring:

$$\mathcal{S}\left(\sum_{j}^{m} s_{ij}\mathbf{a}_j\right) = \mathbf{I}_3 \tag{4.7}$$

If we append a constant identity vector $\mathbf{a_I} = [100101]^T$ to the beginning of the global activation vector $\mathbf{a}$ and add a column to the beginning of $\mathbf{S}$ representing the weight corresponding to passive tissue, then a fully passive tetrahedron would have a value of 1 for this weight (and, respecting the constraints, a value of 0 everywhere else) in its corresponding row in the $\mathbf{S}$ matrix. This also allows our model to contain tetrahedra that are partially muscle and partially passive tissue. This expands the dimensions of our $\mathbf{S}$ matrix to $\mathbf{S} \in \mathbb{R}^{n\times(m+1)}$ and gives our model the ability to include in a single tetrahedron multiple muscles as well as passive tissue, similar to what a volume slice in the face would contain.

4.2.5 Multi-target fitting

In order to properly optimize for a geometry matrix $\mathbf{S}$, we need to use several expressions simultaneously to prevent over-fitting. Each expression has its own $\mathbf{x}$ and $\mathbf{a}$ variables, but all expressions share the same $\mathbf{S}$ variables. This leads to our multi-target inverse optimization problem:

$$\min_{\mathbf{x},\mathbf{b},\mathbf{a},\mathbf{S}} \quad \sum_{k}^{q} E_{target}(\mathbf{x}_k, \mathbf{t}_k) + E_{reg}(\mathbf{a}_k, \mathbf{S}) \tag{4.8a}$$

$$\text{s.t.} \quad \nabla_{\mathbf{x}} E_{act}(\mathbf{x}_0, \mathbf{a}_0, \mathbf{S}) + \nabla_{\mathbf{x}} E_{ext}(\mathbf{x}_0) = 0 \tag{4.8b}$$

$$c(\mathbf{x}_0, \mathbf{b}_0) = 0 \tag{4.8c}$$

$$\vdots$$

$$\nabla_{\mathbf{x}} E_{act}(\mathbf{x}_q, \mathbf{a}_q, \mathbf{S}) + \nabla_{\mathbf{x}} E_{ext}(\mathbf{x}_q) = 0 \tag{4.8d}$$

$$c(\mathbf{x}_q, \mathbf{b}_q) = 0 \tag{4.8e}$$

$$0 \leq \mathbf{S} \leq 1 \tag{4.8f}$$

$$\mathbf{S}\mathbf{1}_n = \mathbf{1}_m \tag{4.8g}$$

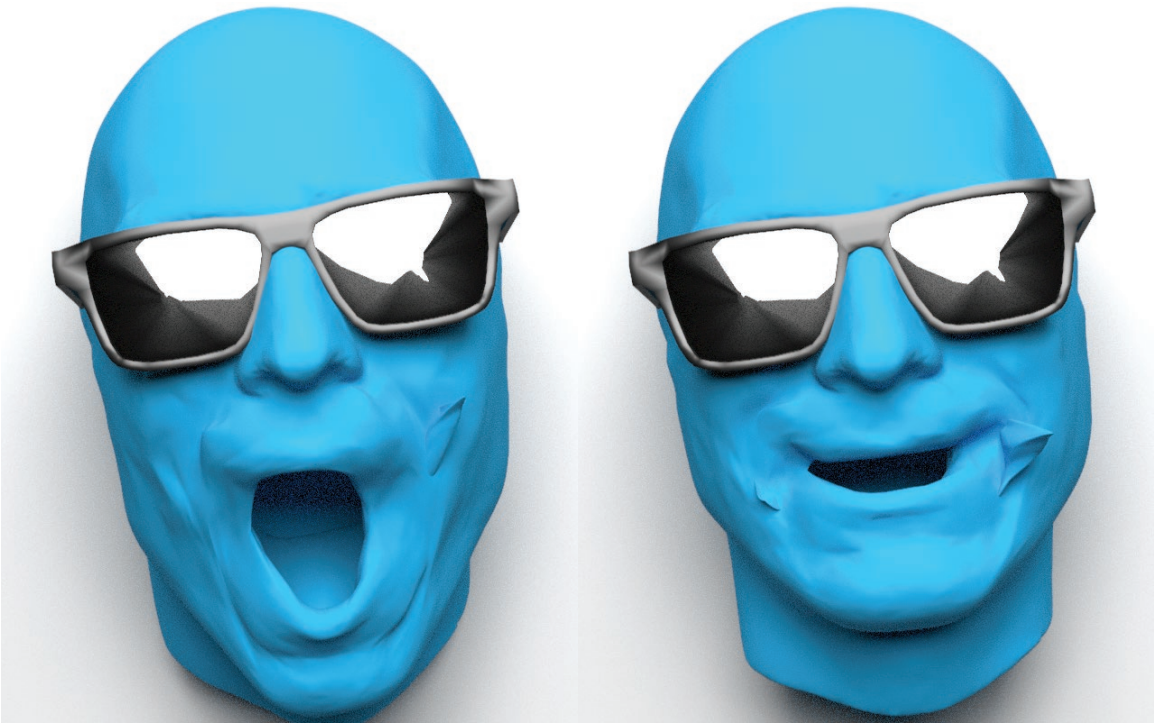

Figure 4.4: Optimizing for only the activations without accounting for the muscle geometry produces artifacts and results in a poor fitting of the input scan.

where q is the number of targets, p is the number of vertices per target, m is the number of muscles, $\mathbf{x} \in \mathbb{R}^{3p \times q}$ represents the positions, $\mathbf{a} \in \mathbb{R}^{6m \times q}$ represents the activations with the different columns representing different targets (i.e., $\mathbf{x}_k, \mathbf{a}_k$ are the positions and activations for target k), and $\mathbf{S} \in \mathbb{R}^{n \times m}$ is the geometry blending matrix described above (recall that n is the number of tetrahedra). The objective term (Eq. 4.8a) contains all of the target energies and the regularization terms (discussed in Section 4.2.6). The constraints in Eq. 4.8b and Eq. 4.8c are the steady-state and physiological constraints (including jaw kinematics) for target 0, and each target has such a pair of constraints. Finally, the **S**-matrix constraints discussed in Section 4.2.4 are represented in Eq. 4.8f and Eq. 4.8g, where $\mathbf{1}_n \in \mathbb{R}^n$ and $\mathbf{1}_m \in \mathbb{R}^m$ represents all-ones vectors.

4.2.6 Regularization

Eq. 4.8 is an under-determined problem due to the fact that the target scans and constraints exclude many of the internal vertices. To help solve the optimization problem, we use simple regularizations on the **A** and **S** matrices that push them towards their initial

values $\mathbf{A}_i, \mathbf{S}_i$:

$$\frac{w_a}{2}||\mathbf{A} - \mathbf{A}_i||_{\mathbf{F}}^2 + \frac{w_s}{2}||\mathbf{S} - \mathbf{S}_i||_{\mathbf{F}}^2 \tag{4.9}$$

The weights w_a, w_s represent the strength of this regularization and are discussed more in Section 4.2.7. Additionally, we want the activations to be volume-preserving, i.e. having a determinant of one. We add the following regularization to try to enforce this property:

$$\frac{w_v}{2}\sum_{j=1}^{m}(det(\mathcal{S}(\mathbf{a}_j) - 1)^2 \tag{4.10}$$

where $\mathbf{a}_j \in R^6$ represents the block from $\mathbf{a}$ corresponding to muscle j. Adding Eq. 4.9 and Eq. 4.10 gives us the E_{reg} term in Eq. 4.8.

4.2.7 Numerical Solution

We solved the optimization problem in Eq. 4.8 using the IPOPT [151] package and employing a block coordinate descent strategy, alternating between solves for $\mathbf{x}, \mathbf{A}$ and $\mathbf{x}, \mathbf{S}$. Additionally, we use high regularization weights w_a, w_s in Eq. 4.9 at the beginning and decrease them in subsequent solves. In our results, the $\mathbf{S}$ matrix was obtained by performing two or three alternating $\mathbf{A}$ and $\mathbf{S}$ iterations before decreasing the weights.

Once an $\mathbf{S}$ matrix is obtained for a subject, we can then solve for new expressions using only the $\mathbf{A}$ solve. For this solve, we do not need to use the strategy of starting with a high regularization weight w_a and then iteratively decrease it; we can use a low weight from the start. This combined with the low dimensionality of the unknown activations makes this solve much faster.

4.3 Results

4.3.1 Evaluation

To optimize for the $\mathbf{S}$ matrix, we used five scans to solve the multi-target optimization problem in Eq. 4.8, shown in Figure 4.2 (top). The resulting meshes are shown in Figure 4.2 (middle) and the fitting error is visualized in Figure 4.2 (bottom) as the point-to-point distance between our mesh vertices and the corresponding vertices in the scans. We alternated between solving for the global $\mathbf{S}$ matrix and the target-specific $\mathbf{A}_i$ matrices as

described in section Section 4.2.7. Our method is able to fit the target scans quite well, with an average fitting error of 0.85 mm.

After this **S** optimization phase, we can now use the **S** to solve for only the muscle activations **A** for new expressions, which were not included in the multi-target fitting optimization. We tried fitting new scans, show in Figure 4.3 (top) by solving only for the activations **A**. Figure 4.3 (middle) shows the resulting mesh while Figure 4.3 (bottom) shows the point-to-point error of the fitting. We can see that we are able to fit these expressions, with an average error of 0.90 mm.

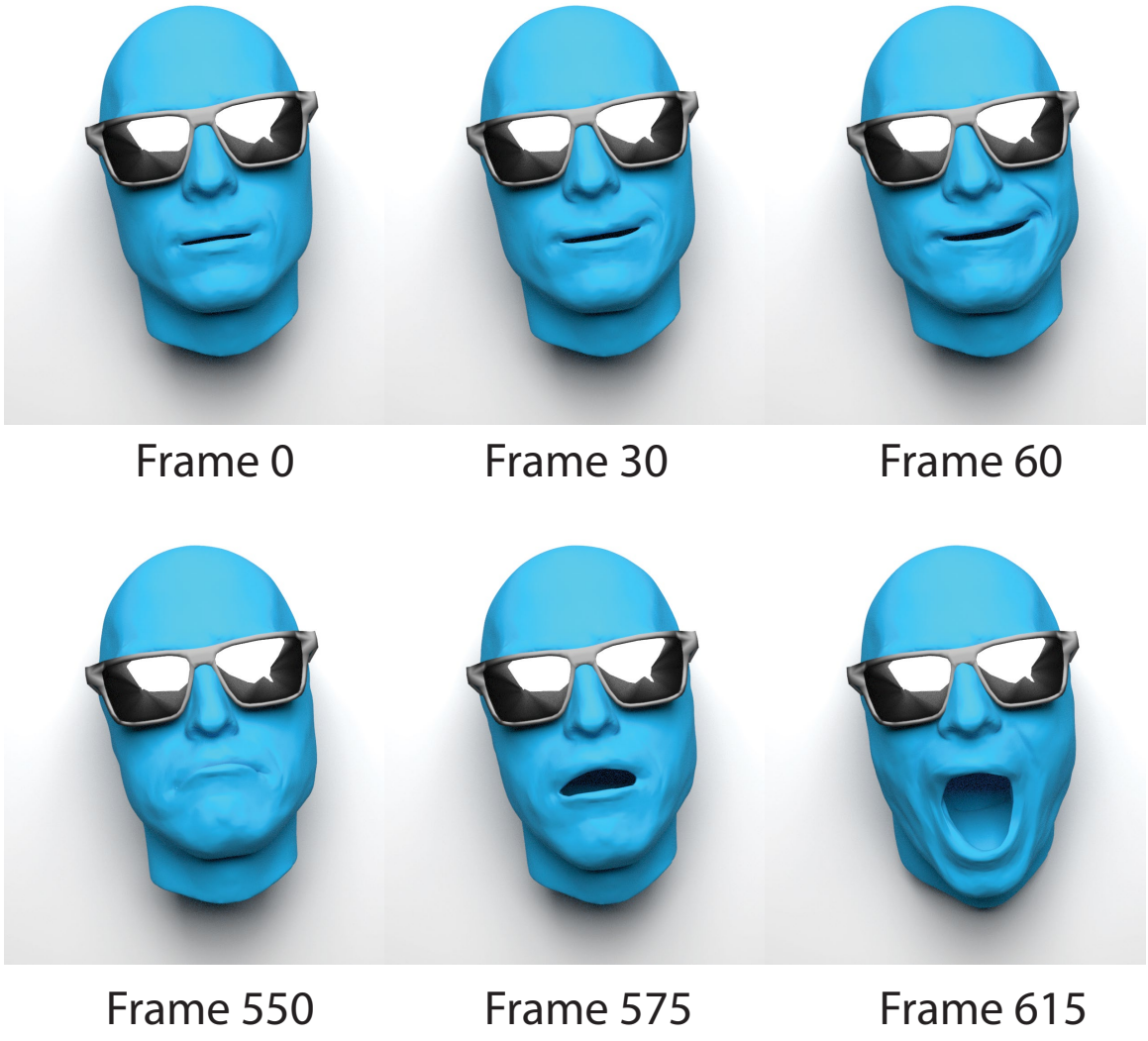

Figure 4.5: An animation may be produced by linearly interpolating the activations of two expressions.

Instead of doing the multi-target optimization for **S**, we can just use the initial values and only solve for the activations. This results in significant artifacts, shown in Figure 4.4, caused by an inability of the underlying reduced-space activations to replicate the target expression, sometimes resulting in significant tetrahedral inversions. Our method with the **S** optimization produces a much better result, showing that we cannot just reduce the space of the muscle activations without accounting for the geometry.

Introducing the **S** matrix does not affect the ability of our activations to be used as volumetric blendshapes [71]. Figure 4.5 shows an animation where we use two expressions as keyframes and linearly interpolate the activations and jaw kinematics of each expression for the in-between frames. Please refer to the accompanying video for a more clear demonstration.

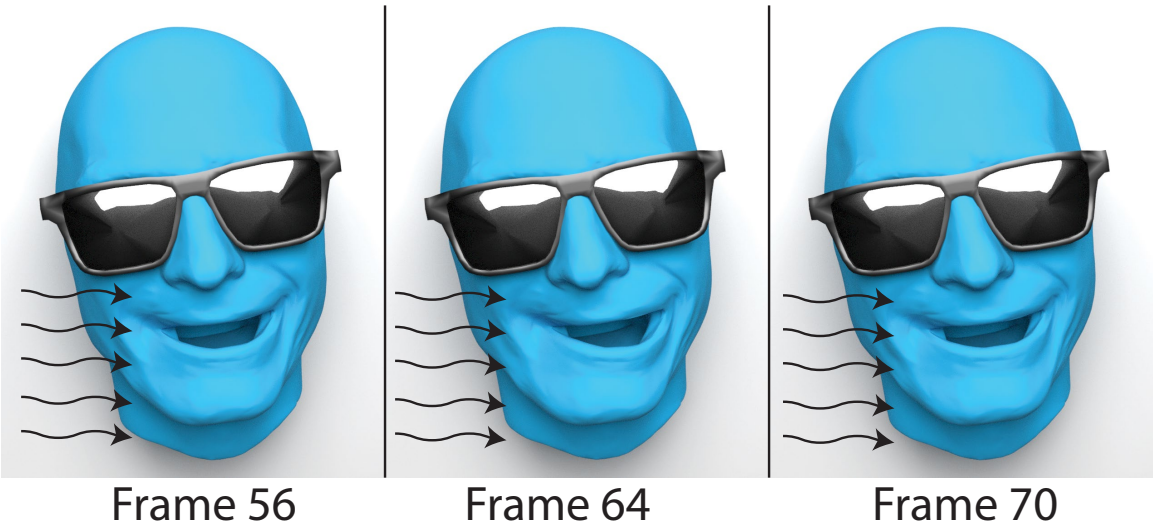

Figure 4.6: Applying external forces such as wind causes the flesh to deform accordingly.

Since we are using a physics-based model, we can also add in physical effects. We can change the physical properties of our subject: in Figure 4.7 we increase the "fatness" of our subject. We accomplish this in a similar vein to the work of [70]: we model fat as a plastic deformation, augmenting $\mathbf{F}(\mathbf{x}, \mathbf{x}_{rest})$ in Eq. 4.1 with $\mathbf{F}(\mathbf{x}, \mathbf{x}_{rest})\mathbf{F}_p^{-}1$, where $\mathbf{F}_p = f_i\mathbf{I}$ is a scaled identity matrix, with $f_i > 0$ representing the fat multiplier for this tetrahedron. Unlike [70], we do not require a manually made "fat map" as we have optimized for the active/passive ratio during our **S** optimization. First, we choose a scalar value for f indicating how much "fatter" ($f > 1$) or "thinner" ($f < 1$) we want our person (e.g., Figure 4.7 uses a range of

0.9 to 1.2). For any given tetrahedron, the first weight in its $\mathbf{s}_i$ vector is the ratio of passive tissue, so we can use this weight to grow or reduce fat as desired: $f_i = f(1 - s_{i0}) + f s_{i0}$.

We can also incorporate external forces: in Figure 4.6 we added a wind force to our keyframed animations in the $+\mathbf{x}$ direction, causing the flesh to deform and jiggle while still making the correct expressions.

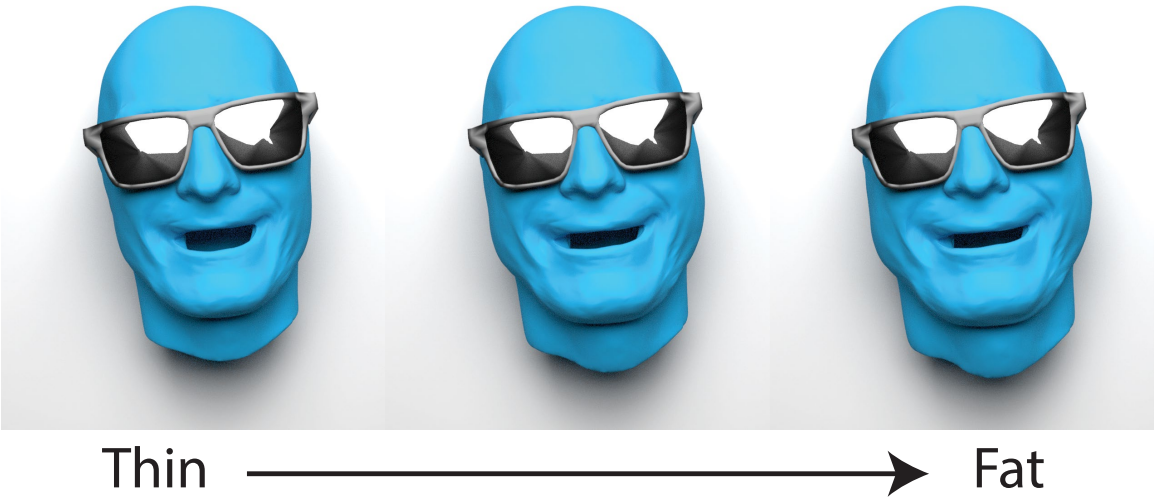

Figure 4.7: We can use our model to simulate the accumulation of fat on our subject. Fat is only accumulated in passive tissue, not in muscle.

4.3.2 Comparison to Previous Work

Ichim et al.[71] introduced a muscle model that enabled fitting to target scans by allowing every muscle tetrahedron to move independently. While this makes the fitting very close to the target scans, it does so by constructing a model that intentionally *overfits* to the scan. One drawback of this approach is that it allows non-physical deformations that can arise from artifacts in the capture process. To illustrate this point, we applied some non-physical deformation to one of our scans and test both the Phace method [70] and our method, shown in Figure 4.8. We can see that our method is not able to fit this deformity due to our more restricted muscle model. Phace[70] is able to (incorrectly) fit this scan artifact. While this scenario is admittedly exaggerated, it illustrates the drawbacks of overfitting to the data. An ideal muscle model is capable of producing *only* the various expressions produced by the facial muscles and is not able to explain non-physical input data.

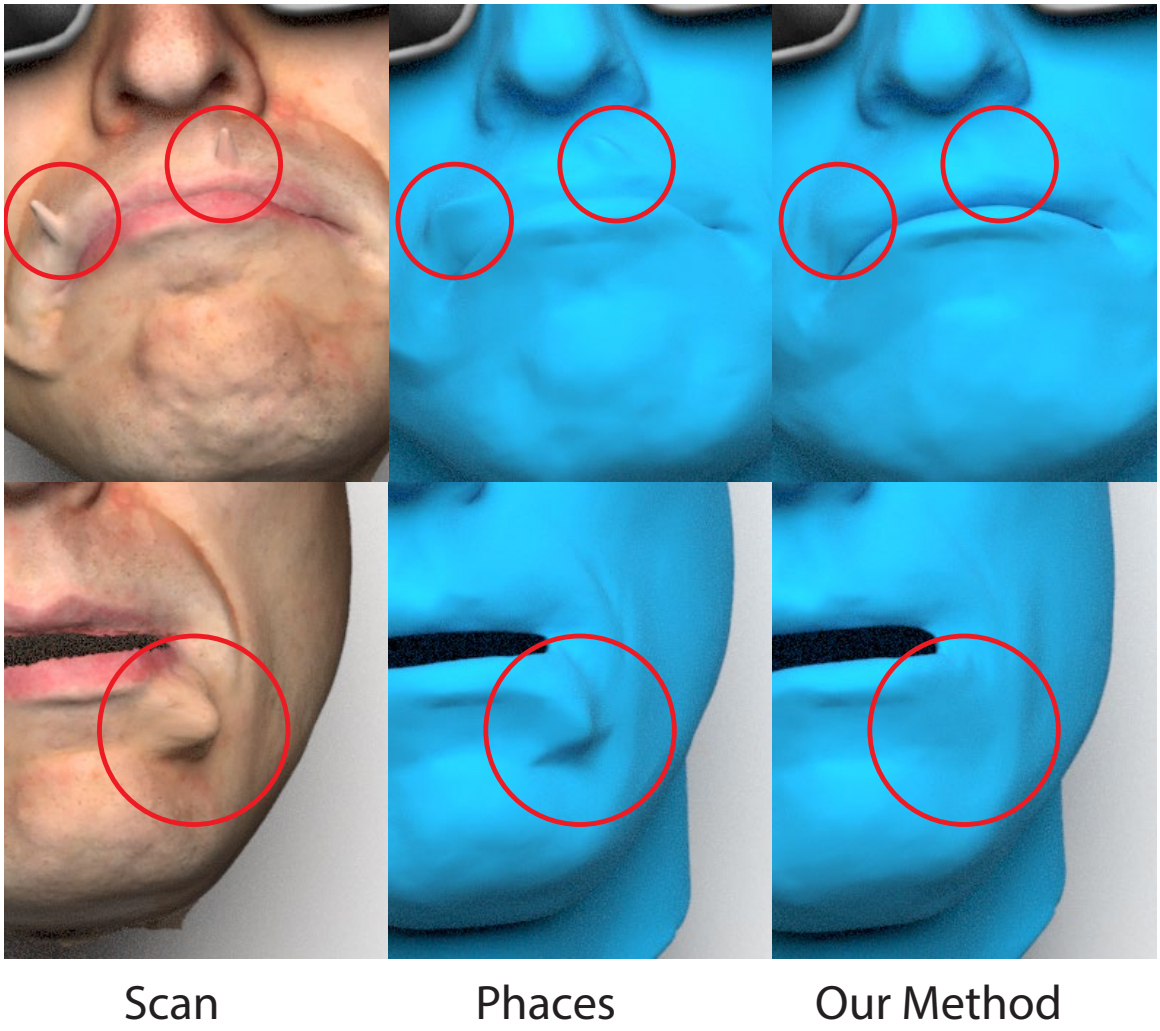

Figure 4.8: Previous work allows for fitting of non-physical shapes. We add some artifacts to a scan (left) and demonstrate how the method of [70] is able to fit this (middle). Our muscle model is correctly unable to fit this artifact (right).

Another side effect of our reduced parameter space for the activations is that we are able to achieve significant speedup when solving for new muscle activations when compared with Phace[70]. This is important because ultimately these activations are used in a volumetric blendshape system, which can require many different expressions to be solved for. For Figure 4.8 the activation solve for Phace[70] (top) took 5 iterations and 734 ms, (bottom) took 7 iterations and 1100 ms. Our activation solve (top) took 10 iterations and 188 ms, (bottom) took 8 iterations and 148 ms with a mesh resolution of 7,046 vertices and 25,227 tetrahedra on an Intel Core i7-8750 processor. For all of the expressions in Figure 4.2 and Figure 4.3, our activation solve took an average of 8.58 iterations and 157.42 ms, while

Phace[70] took an average of 6.5 iterations and 998.08 ms.

CHAPTER 5

CONCLUSION AND FUTURE WORK

In this work, we discussed many different techniques for simulating the human face, beginning with a geometry-based technique for improving capture pipelines and then proceeding to physics-based techniques for simulating the passive and active tissue. We showed the advantages that physics-based models have over geometric approaches, and discussed the limited of current techniques while introducing new ones. However, the human face is an extremely complicated system and we still have a long ways to go before we have video-realistic human simulations.

Geometry-based techniques remain an attractive option due to how well they cooperate with capture technologies. The method we introduced in Chapter 2 (published in [42]) is a post-processing method, that takes advantage of the fact that the inner lip shape and outer mouth shape are influenced by the same underlying muscle structures (e.g., *Obicularis Oris* and other mouth muscles), which allows to us to use the outer mouth shape to infer the inner lip shape. We accomplish this by first transforming the regions we want into gradient space by constructing virtual tets, and using several corrected example meshes to train a linear regression for the mapping between the outer mouth region and the inner lip region. Finally, we use principle component analysis (PCA) to greatly reduce the space of our regression problem. We demonstrate that this method produces good results on several different actors while correcting several different common lip problems (e.g., loss of fleshiness, tracking errors, sticky lips).

This method is made specifically for the lips, but does not cover all problemtic regions of the face. One important area it would likely not be very helpful for is correcting the eyelids. This is because the eyelids are entirely composed of their own flesh and muscle structure (*Levator Palpebrae Superioris* for the upper eyelid and *Capsulopalpebral* for the lower eyelid). This means that we cannot infer the shape of the eyelid just from the surrounding

eye. In the future, deep learning approaches might be used to enhance this.

In Chapter 3, we first discussed the challenges of simulating passive tissue, particularly in a real-time domain with large timesteps. Numerical time integrators are either too slow (adaptive timestepping methods), unstable (explicit integrators, symplectic integrators), or introduce significant uncontrollable numerical damping (BDF-1 or backward Euler and BDF-2). We first begin with an in-depth discussion of various numerical time integrators and discuss their properties. We give a toy-case proof for backward Euler stability that sheds some light as to why it is so stable, even for non-convex potential energies.Then, to remedy the problems of current numerical time integrators, we further explore the concept of energy projecting and apply it to the real-time domain. First, in Section 3.2 (published in [43]) we introduce a simple energy projection that leverages the fact that backward Euler removes energy to the some to find a projection along the linear interpolation between implicit midpoint and backward Euler. We show that even this simple projection can achieve good results, and can be accelerated using techniques such as Projective Dynamics [27]. However, one very important limitation of this method is that it does not preserve the momentum structure of the integrator, meaning it is not momentum-conserving when used with implicit midpoint. Do remedy this issue, we developed the method introduced in Section 3.3 (published in [44]). Here, we keep the idea of projecting to a constant-energy manifold, but instead of relying on the results of other integrators (such as backward Euler) we express this projection as a constrained optimization problem, requiring that we find the closest point on the constant energy manifold while preserving the momentum structure of the integrator. In order to handle situations where momentum is not conserved (e.g., collisions, damping) we introduce slack variables that allow the constraints to be relaxed when needed. We employ a similar energy-tracking scheme as the previous work to keep track of dissipation and forcing. One benefit of formulating the projection in this way is that it can be used with any arbitrary integrator as a starting point, no longer restricted just to implicit midpoint. We show that we are able to improve a variety of integrators, including backward Euler and implicit midpoint, and show that such conservation produces more vivid motion, nicer wrinkles in cloth, and enhanced stability.

One limitation of these methods is that they are only designed for deformable solids and rely on the fact that elastic potential energies are bounded from below by zero, which

is the reason energy conservation implies stability for such potentials. This is not true for every type of potential, and these methods would need some extra steps in order to be applicable to other physics-based systems, such as fluids. Furthermore, while we replicate some first integrals of the real-world system, we importantly do not replicate all of them. Most importantly, our methods are symplectic, which can cause some artifacts to be observed. This reveals that our methods do not necessarily give an *accurate* solution to the underlying ODE's, instead relying on the "visual plausibility" of the solutions rather than numerical accuracy. This manifests itself as high-frequency oscillations when using implicit midpoint as the base integrator. A projection method that can control these high-frequency oscillations would be a very powerful tool for physics-based simulation systems.

The final piece of our human face simulation is to account for the active tissue that drives the face: the muscle. In Chapter 4 we discuss the challenges of building anatomical muscle models, especially personalized ones using only easily-obtainable data such as surface scans. We discuss the frameworks of "volumetric blendshapes" [71, 70] and their issues with over-parameterization. In Section 4.2 (submitted to Computers and Graphics) we introduce a method that builds on this framework, but greatly reduces the parameterization by making a few key observations. The human face has two main classes of parameters: subject-specific parameters and pose-specific parameters. Subject-specific parameters are parameters that vary from person to person, but are constant for any given person, regardless of the expression. These would include properties like the geometric shape of the person's bones, flesh, and muscle as well as various stiffness parameters. Pose-specific parameters ideally live in a very small space: they would be the muscle activation level for each muscle. We separate out the muscle blending geometry (which is a subject-specific parameter) from the muscle activations, resulting in a much lower-dimensional activation space that is a function of the number of muscles rather than the number of volumetric elements. While this helps the parameterization a lot, it is still far from an ideal parameterization; it has 6 degrees of freedom per muscle rather than the ideal number of 1 degree of freedom per muscle. Furthermore, muscle activations are linearly blended together, which is also not what happens in real-world human anatomy.

We hope that this thesis provides important steps to advance the state-of-the-art of human facial simulation. There are many different aspects that play into creating a realistic

human facial model, especially one that is personalized to a specific actor. We covered geometry-based techniques, simulation of the passive tissues, and muscle models. Each of these areas still needs a lot of development before realistic human facial models can be fully utilized in the entertainment and medical industries. I look forward to seeing the development of these areas in the future.

REFERENCES

[1] O. ALEXANDER, M. ROGERS, W. LAMBETH, J.-Y. CHIANG, W.-C. MA, C.-C. WANG, AND P. DEBEVEC, *The digital emily project: Achieving a photorealistic digital actor*, IEEE Computer Graphics and Applications, 30 (2010), pp. 20–31.

[2] O. ALEXANDER, M. ROGERS, W. LAMBETH, M. CHIANG, AND P. DEBEVEC, *The digital emily project: photoreal facial modeling and animation*, in Acm siggraph 2009 courses, ACM, 2009, p. 12.

[3] D. ALI-HAMADI, T. LIU, B. GILLES, L. KAVAN, F. FAURE, O. PALOMBI, AND M.-P. CANI, *Anatomy transfer*, ACM Transactions on Graphics (TOG), 32 (2013), pp. 1–8.

[4] J. ALLARD, S. COTIN, F. FAURE, P.-J. BENSOUSSAN, F. POYER, C. DURIEZ, H. DELINGETTE, AND L. GRISONI, *Sofa-an open source framework for medical simulation*, 2007.

[5] R. ANDERSON, B. STENGER, AND R. CIPOLLA, *Lip tracking for 3d face registration.*, in MVA, 2013, pp. 145–148.

[6] S. ANDREWS, K. ERLEBEN, P. G. KRY, AND M. TEICHMANN, *Constraint reordering for iterative multi-body simulation with contact*, in ECCOMAS Thematic Conference on Multibody Dynamic, 2017.

[7] U. M. ASCHER AND S. REICH, *The midpoint scheme and variants for hamiltonian systems: advantages and pitfalls*, SIAM Journal on Scientific Computing, 21 (1999), pp. 1045–1065.

[8] M. BAO, M. CONG, S. GRABLI, AND R. FEDKIW, *High-quality face capture using anatomical muscles*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 10802–10811.

[9] D. BARAFF AND A. WITKIN, *Large steps in cloth simulation*, in Proc. of ACM SIGGRAPH, 1998, pp. 43–54.

[10] M. BARNARD, E.-J. HOLDEN, AND R. OWENS, *Lip tracking using pattern matching snakes*, in Proc. of the Fifth Asian Conference on Computer Vision, vol. 1, 2002.

[11] V. BARRIELLE, N. STOIBER, AND C. CAGNIART, *Blendforces: A dynamic framework for facial animation*, in Computer Graphics Forum, vol. 35, Wiley Online Library, 2016, pp. 341–352.

[12] T. BEELER, B. BICKEL, P. BEARDSLEY, B. SUMNER, AND M. GROSS, *High-quality single-shot capture of facial geometry*, in ACM Transactions on Graphics (ToG), vol. 29, ACM, 2010, p. 40.

[13] T. BEELER, B. BICKEL, G. NORIS, P. BEARDSLEY, S. MARSCHNER, R. W. SUMNER, AND M. GROSS, *Coupled 3d reconstruction of sparse facial hair and skin*, ACM Transactions on Graphics (ToG), 31 (2012), p. 117.

[14] T. BEELER AND D. BRADLEY, *Rigid stabilization of facial expressions*, ACM Transactions on Graphics (TOG), 33 (2014), p. 44.

[15] T. BEELER, F. HAHN, D. BRADLEY, B. BICKEL, P. BEARDSLEY, C. GOTSMAN, R. W. SUMNER, AND M. GROSS, *High-quality passive facial performance capture using anchor frames*, in ACM Transactions on Graphics (TOG), vol. 30, ACM, 2011, p. 75.

[16] J. BENDER, D. KOSCHIER, P. CHARRIER, AND D. WEBER, *Position-based simulation of continuous materials*, Computers & Graphics, 44 (2014), pp. 1–10.

[17] J. BENDER, M. MÜLLER, M. A. OTADUY, M. TESCHNER, AND M. MACKLIN, *A survey on position-based simulation methods in computer graphics*, in Comput. Graph. Forum, vol. 33, 2014, pp. 228–251.

[18] P. BÉRARD, D. BRADLEY, M. GROSS, AND T. BEELER, *Lightweight eye capture using a parametric model*, ACM Transactions on Graphics (TOG), 35 (2016), p. 117.

[19] P. BÉRARD, D. BRADLEY, M. NITTI, T. BEELER, AND M. H. GROSS, *High-quality capture of eyes.*, (2014).

[20] A. BERMANO, T. BEELER, Y. KOZLOV, D. BRADLEY, B. BICKEL, AND M. GROSS, *Detailed spatio-temporal reconstruction of eyelids*, ACM Transactions on Graphics (TOG), 34 (2015), p. 44.

[21] K. S. BHAT, R. GOLDENTHAL, Y. YE, R. MALLET, AND M. KOPERWAS, *High fidelity facial animation capture and retargeting with contours*, in Proceedings of the 12th ACM SIGGRAPH/eurographics symposium on computer animation, ACM, 2013, pp. 7–14.

[22] B. BICKEL, M. BÄCHER, M. A. OTADUY, W. MATUSIK, H. PFISTER, AND M. GROSS, *Capture and modeling of non-linear heterogeneous soft tissue*, ACM Transactions on Graphics (TOG), 28 (2009), pp. 1–9.

[23] B. BICKEL, P. KAUFMANN, M. SKOURAS, B. THOMASZEWSKI, D. BRADLEY, T. BEELER, P. JACKSON, S. MARSCHNER, W. MATUSIK, AND M. GROSS, *Physical face cloning*, ACM Transactions on Graphics (TOG), 31 (2012), pp. 1–10.

[24] V. BLANZ AND T. VETTER, *A morphable model for the synthesis of 3d faces*, in Proceedings of the 26th annual conference on Computer graphics and interactive techniques, ACM Press/Addison-Wesley Publishing Co., 1999, pp. 187–194.

[25] S. S. BLEMKER, P. M. PINSKY, AND S. L. DELP, *A 3d model of muscle reveals the causes of nonuniform strains in the biceps brachii*, Journal of biomechanics, 38 (2005), pp. 657–665.

[26] L. BÖRGESSON, *Abaqus*, in Developments in geotechnical engineering, vol. 79, Elsevier, 1996, pp. 565–570.

[27] S. BOUAZIZ, S. MARTIN, T. LIU, L. KAVAN, AND M. PAULY, *Projective dynamics: fusing constraint projections for fast simulation*, ACM Transactions on Graphics (TOG), 33 (2014), pp. 1–11.

[28] S. BOYD, N. PARIKH, E. CHU, B. PELEATO, AND J. ECKSTEIN, *Distributed optimization and statistical learning via the alternating direction method of multipliers*, Foundations and Trends® in Machine Learning, 3 (2011), pp. 1–122.

[29] S. BOYD AND L. VANDENBERGHE, *Convex Optimization*, Cambridge University Press, New York, NY, USA, 2004.

[30] D. BRADLEY, W. HEIDRICH, T. POPA, AND A. SHEFFER, *High resolution passive facial performance capture*, ACM transactions on graphics (TOG), 29 (2010), p. 41.

[31] R. BRIDSON, R. FEDKIW, AND J. ANDERSON, *Robust treatment of collisions, contact and friction for cloth animation*, ACM Trans. Graph., 21 (2002), pp. 594–603.

[32] R. BRIDSON, S. MARINO, AND R. FEDKIW, *Simulation of clothing with folds and wrinkles*, 2003, pp. 28–36.

[33] C. CAO, D. BRADLEY, K. ZHOU, AND T. BEELER, *Real-time high-fidelity facial performance capture*, ACM Transactions on Graphics (ToG), 34 (2015), p. 46.

[34] I. CHAO, U. PINKALL, P. SANAN, AND P. SCHRÖDER, *A simple geometric model for elastic deformations*, ACM Trans. Graph., 29 (2010), p. 38.

[35] Y. J. CHEN, U. M. ASCHER, AND D. K. PAI, *Exponential rosenbrock-euler integrators for elastodynamic simulation*, IEEE transactions on visualization and computer graphics, 24 (2018), pp. 2702–2713.

[36] K.-J. CHOI AND H.-S. KO, *Stable but responsive cloth*, ACM Trans. Graph., 21 (2002), pp. 604–611.

[37] J. CHUNG AND G. HULBERT, *A time integration algorithm for structural dynamics with improved numerical dissipation: the generalized-α method*, Journal of applied mechanics, 60 (1993), pp. 371–375.

[38] M. CONG, M. BAO, K. S. BHAT, R. FEDKIW, ET AL., *Fully automatic generation of anatomical face simulation models*, in Proceedings of the 14th ACM SIGGRAPH/Eurographics Symposium on Computer Animation, ACM, 2015, pp. 175–183.

[39] M. CONG, K. S. BHAT, AND R. FEDKIW, *Art-directed muscle simulation for high-end facial animation*, 2016.

[40] G. DEBUNNE, M. DESBRUN, M.-P. CANI, AND A. H. BARR, *Dynamic real-time deformations using space & time adaptive sampling*, in Proc. of Computer graphics and interactive techniques, 2001, pp. 31–36.

[41] K. DEKKER AND J. G. VERWER, *Stability of runge-kutta methods for stiff nonlinear differential equations*, ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik, 67 (1987), pp. 68–68.

[42] D. DINEV, T. BEELER, D. BRADLEY, M. BÄCHER, H. XU, AND L. KAVAN, *User-guided lip correction for facial performance capture*, in Computer Graphics Forum, vol. 37, Wiley Online Library, 2018, pp. 93–101.

[43] D. DINEV, T. LIU, AND L. KAVAN, *Stabilizing integrators for real-time physics*, ACM Transactions on Graphics (TOG), 37 (2018), pp. 1–19.

[44] D. DINEV, T. LIU, J. LI, B. THOMASZEWSKI, AND L. KAVAN, *Fepr: fast energy projection for real-time simulation of deformable objects*, ACM Transactions on Graphics (TOG), 37 (2018), pp. 1–12.

[45] R. W. EASTON, *Geometric methods for discrete dynamical systems*, 1998.

[46] B. EBERHARDT, O. ETZMUSS, AND M. HAUTH, *Implicit-explicit schemes for fast animation with particle systems*, Springer, 2000.

[47] P. EKMAN AND E. L. ROSENBERG, *What the face reveals: Basic and applied studies of spontaneous expression using the Facial Action Coding System (FACS)*, Oxford University Press, USA, 1997.

[48] R. D. ENGLE, R. D. SKEEL, AND M. DREES, *Monitoring energy drift with shadow hamiltonians*, Journal of Computational Physics, 206 (2005), pp. 432–452.

[49] E. ENGLISH AND R. BRIDSON, *Animating developable surfaces using nonconforming elements*, vol. 27, ACM, 2008, p. 66.

[50] N. EVENO, A. CAPLIER, AND P.-Y. COULON, *Accurate and quasi-automatic lip tracking*, IEEE Transactions on circuits and systems for video technology, 14 (2004), pp. 706–715.

[51] Y. FAN, J. LITVEN, D. I. LEVIN, AND D. K. PAI, *Eulerian-on-lagrangian simulation*, ACM Transactions on Graphics (TOG), 32 (2013), pp. 1–9.

[52] Y. FAN, J. LITVEN, AND D. K. PAI, *Active volumetric musculoskeletal systems*, ACM Transactions on Graphics (TOG), 33 (2014), pp. 1–9.

[53] B. FIERZ, J. SPILLMANN, AND M. HARDERS, *Element-wise mixed implicit-explicit integration for stable dynamic simulation of deformable objects*, 2011, pp. 257–266.

[54] M. FRÂNCU AND F. MOLDOVEANU, *Position based simulation of solids with accurate contact handling*, Computers & Graphics, 69 (2017), pp. 12–23.

[55] M. FRATARCANGELI, V. TIBALDO, AND F. PELLACINI, *Vivace: a practical gauss-seidel method for stable soft body dynamics*, ACM Transactions on Graphics (TOG), 35 (2016), p. 214.

[56] G. FYFFE, A. JONES, O. ALEXANDER, R. ICHIKARI, AND P. DEBEVEC, *Driving high-resolution facial scans with video performance capture*, ACM Transactions on Graphics (TOG), 34 (2014), p. 8.

[57] P. GARRIDO, M. ZOLLHÖFER, C. WU, D. BRADLEY, P. PÉREZ, T. BEELER, AND C. THEOBALT, *Corrective 3d reconstruction of lips from monocular video.*, (2016).

[58] T. F. GAST, C. SCHROEDER, A. STOMAKHIN, C. JIANG, AND J. M. TERAN, *Optimization integrator for large time steps*, IEEE transactions on visualization and computer graphics, 21 (2015), pp. 1103–1115.

[59] Z. GE AND J. E. MARSDEN, *Lie-poisson hamilton-jacobi theory and lie-poisson integrators*, Physics Letters A, 133 (1988), pp. 134–139.

[60] A. GHOSH, G. FYFFE, B. TUNWATTANAPONG, J. BUSCH, X. YU, AND P. DEBEVEC, *Multiview face capture using polarized spherical gradient illumination*, in ACM Transactions on Graphics (TOG), vol. 30, ACM, 2011, p. 129.

[61] R. GOLDENTHAL, D. HARMON, R. FATTAL, M. BERCOVIER, AND E. GRINSPUN, *Efficient simulation of inextensible cloth*, ACM Transactions on Graphics (TOG), 26 (2007), p. 49.

[62] O. GONZALEZ, *Time integration and discrete hamiltonian systems*, Journal of Nonlinear Science, 6 (1996), pp. 449–467.

[63] ——, *Exact energy and momentum conserving algorithms for general models in nonlinear elasticity*, Computer Methods in Applied Mechanics and Engineering, 190 (2000), pp. 1763–1783.

[64] P. GRAHAM, B. TUNWATTANAPONG, J. BUSCH, X. YU, A. JONES, P. DEBEVEC, AND A. GHOSH, *Measurement-based synthesis of facial microgeometry*, in Computer Graphics Forum, vol. 32, Wiley Online Library, 2013, pp. 335–344.

[65] E. HAIRER, *Long-time energy conservation of numerical integrators*, Foundations of computational mathematics, Santander 2005, 2006, pp. 162–180.

[66] E. HAIRER, C. LUBICH, AND G. WANNER, *Geometric numerical integration: structure-preserving algorithms for ordinary differential equations*, vol. 31, 2006.

[67] L. HU, C. MA, L. LUO, AND H. LI, *Single-view hair modeling using a hairstyle database*, ACM Transactions on Graphics (TOG), 34 (2015), pp. 1–9.

[68] T. HUGHES, T. CAUGHEY, AND W. LIU, *Finite-element methods for nonlinear elastodynamics which conserve energy*, Journal of Applied Mechanics, 45 (1978), pp. 366–370.

[69] A. E. ICHIM, S. BOUAZIZ, AND M. PAULY, *Dynamic 3d avatar creation from hand-held video input*, ACM Transactions on Graphics (ToG), 34 (2015), pp. 1–14.

[70] A.-E. ICHIM, P. KADLEČEK, L. KAVAN, AND M. PAULY, *Phace: physics-based face modeling and animation*, ACM Transactions on Graphics (TOG), 36 (2017), p. 153.

[71] A. E. ICHIM, L. KAVAN, M. NIMIER-DAVID, AND M. PAULY, *Building and animating user-specific volumetric face rigs.*, 2016.

[72] T. INSIDER, *What happens when 185 mph wind is blown straight at your face*.

[73] N. JIN, W. LU, Z. GENG, AND R. P. FEDKIW, *Inequality cloth*, in Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation, ACM, 2017, p. 16.

[74] P. KADLEČEK, A.-E. ICHIM, T. LIU, J. KŘIVÁNEK, AND L. KAVAN, *Reconstructing personalized anatomical models for physics-based body animation*, ACM Transactions on Graphics (TOG), 35 (2016), p. 213.

[75] P. KADLECEK AND L. KAVAN, *Building accurate physics-based face models from data*, in Symposium on Computer Animation, 2019.

[76] C. KANE, *Variational integrators and the Newmark algorithm for conservative and dissipative mechanical systems*, PhD thesis, caltech, 1999.

[77] C. KANE, J. E. MARSDEN, AND M. ORTIZ, *Symplectic-energy-momentum preserving variational integrators*, Journal of mathematical physics, 40 (1999), pp. 3353–3371.

[78] R. KAUCIC AND A. BLAKE, *Accurate, real-time, unadorned lip tracking*, in Sixth International Conference on Computer Vision (IEEE Cat. No. 98CH36271), IEEE, 1998, pp. 370–375.

[79] M. KAWAI, T. IWAO, A. MAEJIMA, AND S. MORISHIMA, *Automatic photorealistic 3d inner mouth restoration from frontal images*, in International Symposium on Visual Computing, Springer, 2014, pp. 51–62.

[80] L. KHAREVYCH, W. YANG, Y. TONG, E. KANSO, J. E. MARSDEN, P. SCHRÖDER, AND M. DESBRUN, *Geometric, variational integrators for computer animation*, 2006, pp. 43–51.

[81] M. KIM, G. PONS-MOLL, S. PUJADES, S. BANG, J. KIM, M. J. BLACK, AND S.-H. LEE, *Data-driven physics for human soft tissue animation*, ACM Transactions on Graphics (TOG), 36 (2017), pp. 1–12.

[82] T.-Y. KIM, N. CHENTANEZ, AND M. MÜLLER-FISCHER, *Long range attachments-a method to simulate inextensible clothing in computer games*, 2012, pp. 305–310.

[83] Y. KOZLOV, D. BRADLEY, M. BÄCHER, B. THOMASZEWSKI, T. BEELER, AND M. GROSS, *Enriching facial blendshape rigs with physical simulation*, in Computer Graphics Forum, vol. 36, Wiley Online Library, 2017, pp. 75–84.

[84] D. KUHL AND M. CRISFIELD, *Energy-conserving and decaying algorithms in non-linear structural dynamics*, International journal for numerical methods in engineering, 45 (1999), pp. 569–599.

[85] R. A. LABUDDE AND D. GREENSPAN, *Energy and momentum conserving methods of arbitrary order for the numerical integration of equations of motion*, Numerische Mathematik, 25 (1975), pp. 323–346.

[86] S. LAINE, T. KARRAS, T. AILA, A. HERVA, S. SAITO, R. YU, H. LI, AND J. LEHTINEN, *Production-level facial performance capture using deep convolutional neural networks*, in Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation, ACM, 2017, p. 10.

[87] J. D. LAMBERT, *Numerical Methods for Ordinary Differential Systems: The Initial Value Problem*, John Wiley & Sons, Inc., New York, NY, USA, 1991.

[88] L. LAN, M. CONG, AND R. FEDKIW, *Lessons from the evolution of an anatomical facial muscle model*, in Proceedings of the ACM SIGGRAPH Digital Production Symposium, ACM, 2017, p. 11.

[89] M. LAU, J. CHAI, Y.-Q. XU, AND H.-Y. SHUM, *Face poser: Interactive modeling of 3d facial expressions using facial priors*, ACM Transactions on Graphics (TOG), 29 (2009), pp. 1–17.

[90] S.-H. Lee, E. Sifakis, and D. Terzopoulos, *Comprehensive biomechanical modeling and simulation of the upper body*, ACM Transactions on Graphics (TOG), 28 (2009), p. 99.

[91] J. P. Lewis, K. Anjyo, T. Rhee, M. Zhang, F. H. Pighin, and Z. Deng, *Practice and theory of blendshape facial models.*, (2014).

[92] H. Li, T. Weise, and M. Pauly, *Example-based facial rigging*, Acm transactions on graphics (tog), 29 (2010), pp. 1–6.

[93] H. Li, J. Yu, Y. Ye, and C. Bregler, *Realtime facial animation with on-the-fly correctives.*, ACM Trans. Graph., 32 (2013), pp. 42–1.

[94] T. Li, T. Bolkart, M. J. Black, H. Li, and J. Romero, *Learning a model of facial shape and expression from 4d scans*, ACM Transactions on Graphics (ToG), 36 (2017), p. 194.

[95] T. Liu, A. W. Bargteil, J. F. O'Brien, and L. Kavan, *Fast simulation of mass-spring systems*, ACM Trans. Graph., 32 (2013), pp. 209:1–7.

[96] T. Liu, S. Bouaziz, and L. Kavan, *Quasi-newton methods for real-time simulation of hyperelastic materials*, ACM Transactions on Graphics (TOG), 36 (2017), pp. 1–16.

[97] Y. Liu, F. Xu, J. Chai, X. Tong, L. Wang, and Q. Huo, *Video-audio driven real-time facial animation*, ACM Transactions on Graphics (TOG), 34 (2015), pp. 1–10.

[98] J. E. Lloyd, I. Stavness, and S. Fels, *Artisynth: A fast interactive biomechanical modeling toolkit combining multibody and finite element simulation*, in Soft tissue biomechanical modeling for computer assisted surgery, Springer, 2012, pp. 355–394.

[99] L. Luo, H. Li, S. Paris, T. Weise, M. Pauly, and S. Rusinkiewicz, *Multi-view hair capture using orientation fields*, in Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on, IEEE, 2012, pp. 1490–1497.

[100] W.-C. Ma, Y.-H. Wang, G. Fyffe, B.-Y. Chen, and P. Debevec, *A blendshape model that incorporates physical interaction*, Computer Animation and Virtual Worlds, 23 (2012), pp. 235–243.

[101] S. A. Maas, B. J. Ellis, G. A. Ateshian, and J. A. Weiss, *Febio: finite elements for biomechanics*, Journal of biomechanical engineering, 134 (2012).

[102] M. Macklin and M. Müller, *Position based fluids*, ACM Trans. Graph., 32 (2013), p. 104.

[103] M. Macklin, M. Müller, and N. Chentanez, *Xpbd: position-based simulation of compliant constrained dynamics*, in Proc. of Motion in Games, 2016, pp. 49–54.

[104] M. Macklin, M. Müller, N. Chentanez, and T.-Y. Kim, *Unified particle physics for real-time applications*, ACM Trans. Graph., 33 (2014), p. 153.

[105] J. E. Marsden and M. West, *Discrete mechanics and variational integrators*, Acta Numerica 2001, 10 (2001), pp. 357–514.

[106] S. Martin, B. Thomaszewski, E. Grinspun, and M. Gross, *Example-based elastic materials*, in ACM Transactions on Graphics (TOG), vol. 30, ACM, 2011, p. 72.

[107] A. McAdams, Y. Zhu, A. Selle, M. Empey, R. Tamstorf, J. Teran, and E. Sifakis, *Efficient elasticity for character skinning with contact and collisions*, in ACM Trans. Graph., vol. 30, 2011, p. 37.

[108] D. Michels, V. T. Luan, and M. Tokman, *A stiffly accurate integrator for elastodynamic problems*, SIGGRAPH, (2017).

[109] D. L. Michels and M. Desbrun, *A semi-analytical approach to molecular dynamics*, Journal of Computational Physics, 303 (2015), pp. 336–354.

[110] D. L. Michels, G. A. Sobottka, and A. G. Weber, *Exponential integrators for stiff elastodynamic problems*, ACM Trans. Graph., 33 (2014), p. 7.

[111] J. C. Miranda, X. Alvarez, J. Orvalho, D. Gutierrez, A. A. Sousa, and V. Orvalho, *Sketch express: facial expressions made easy*, in Proceedings of the Eighth Eurographics Symposium on Sketch-Based Interfaces and Modeling, 2011, pp. 87–94.

[112] M. Müller, *Hierarchical Position Based Dynamics*, in Workshop in Virtual Reality Interactions and Physical Simulation "VRIPHYS" (2008), 2008.

[113] M. Müller, N. Chentanez, T.-Y. Kim, and M. Macklin, *Strain based dynamics*, vol. 2, 2014.

[114] M. Müller, B. Heidelberger, M. Hennix, and J. Ratcliff, *Position based dynamics*, Journal of Visual Communication and Image Representation, 18 (2007), pp. 109–118.

[115] K. Nagano, G. Fyffe, O. Alexander, J. Barbič, H. Li, A. Ghosh, and P. E. Debevec, *Skin microstructure deformation with displacement map convolution.*, (2015).

[116] M. A. Nazari, P. Perrier, M. Chabanas, and Y. Payan, *Simulation of dynamic orofacial movements using a constitutive law varying with muscle activation*, Computer methods in biomechanics and biomedical engineering, 13 (2010), pp. 469–482.

[117] L. P. Nedel and D. Thalmann, *Real time muscle deformations using mass-spring systems*, in Computer Graphics International, 1998. Proceedings, IEEE, 1998, pp. 156–165.

[118] N. M. Newmark, *A method of computation for structural dynamics*, in Proc. ASCE, vol. 85, 1959, pp. 67–94.

[119] Q. D. Nguyen and M. Milgram, *Semi adaptive appearance models for lip tracking*, in Image Processing (ICIP), 2009 16th IEEE International Conference on, IEEE, 2009, pp. 2437–2440.

[120] J. Nocedal and S. Wright, *Numerical optimization*, 2006.

[121] K. Olszewski, J. J. Lim, S. Saito, and H. Li, *High-fidelity facial and speech animation for vr hmds*, ACM Transactions on Graphics (TOG), 35 (2016), pp. 1–14.

[122] M. Overby, G. E. Brown, J. Li, and R. Narain, *Admm $\supseteq$ projective dynamics: Fast simulation of hyperelastic models with dynamic constraints*, IEEE transactions on visualization and computer graphics, 23 (2017), pp. 2222–2234.

[123] D. K. Pai, K. v. d. Doel, D. L. James, J. Lang, J. E. Lloyd, J. L. Richmond, and S. H. Yau, *Scanning physical interaction behavior of 3d objects*, in Proceedings of the 28th annual conference on Computer graphics and interactive techniques, 2001, pp. 87–96.

[124] D. K. Pai, A. Rothwell, P. Wyder-Hodge, A. Wick, Y. Fan, E. Larionov, D. Harrison, D. R. Neog, and C. Shing, *The human touch: measuring contact with real human soft tissues*, ACM Transactions on Graphics (TOG), 37 (2018), pp. 1–12.

[125] E. G. Parker and J. F. O'Brien, *Real-time deformation and fracture in a game environment*, 2009, pp. 165–175.

[126] R. B. i. Ribera, E. Zell, J. Lewis, J. Noh, and M. Botsch, *Facial retargeting with automatic range of motion alignment*, ACM Transactions on Graphics (TOG), 36 (2017), pp. 1–12.

[127] S. Saito, Z.-Y. Zhou, and L. Kavan, *Computational bodybuilding: Anatomically-based modeling of human bodies*, ACM Transactions on Graphics (TOG), 34 (2015), p. 41.

[128] Y. Seol, J. P. Lewis, J. Seo, B. Choi, K. Anjyo, and J. Noh, *Spacetime expression cloning for blendshapes*, ACM Transactions on Graphics (TOG), 31 (2012), pp. 1–12.

[129] F. Shi, H.-T. Wu, X. Tong, and J. Chai, *Automatic acquisition of high-fidelity facial performances using monocular videos*, ACM Transactions on Graphics (TOG), 33 (2014), p. 222.

[130] H. Si, *Tetgen, a delaunay-based quality tetrahedral mesh generator*, ACM Transactions on Mathematical Software (TOMS), 41 (2015), pp. 1–36.

[131] E. Sifakis and J. Barbič, *Fem simulation of 3d deformable solids: a practitioner's guide to theory, discretization and model reduction*, in ACM SIGGRAPH Courses, 2012, p. 20.

[132] E. Sifakis, I. Neverov, and R. Fedkiw, *Automatic determination of facial muscle activations from sparse motion capture marker data*, Acm transactions on graphics (tog), 24 (2005), pp. 417–425.

[133] J. C. Simo, N. Tarnow, and K. Wong, *Exact energy-momentum conserving algorithms and symplectic schemes for nonlinear dynamics*, Computer methods in applied mechanics and engineering, 100 (1992), pp. 63–116.

[134] G. Sobottka, T. Lay, and A. Weber, *Stable integration of the dynamic cosserat equations with application to hair modeling*, (2008).

[135] J. Stam, *Nucleus: towards a unified dynamics solver for computer graphics*, in IEEE Int. Conf. on CAD and Comput. Graph., 2009, pp. 1–11.

[136] I. Stavness, M. A. Nazari, C. Flynn, P. Perrier, Y. Payan, J. E. Lloyd, and S. Fels, *Coupled biomechanical modeling of the face, jaw, skull, tongue, and hyoid bone*, in 3D multiscale physiological human, Springer, 2014, pp. 253–274.

[137] A. Stern and M. Desbrun, *Discrete geometric mechanics for variational time integrators*, in ACM SIGGRAPH Courses, ACM, 2006, pp. 75–80.

[138] A. Stern and E. Grinspun, *Implicit-explicit variational integration of highly oscillatory problems*, Multiscale Modeling & Simulation, 7 (2009), pp. 1779–1794.

[139] J. SU, R. SHETH, AND R. FEDKIW, *Energy conservation for the simulation of deformable bodies*, TVCG, 19 (2013), pp. 189–200.

[140] J. TERAN, S. BLEMKER, V. 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, Eurographics Association, 2003, pp. 68–74.

[141] J. TERAN, E. SIFAKIS, S. S. BLEMKER, V. NG-THOW-HING, C. LAU, AND R. FEDKIW, *Creating and simulating skeletal muscle from the visible human data set*, IEEE Transactions on Visualization and Computer Graphics, 11 (2005), pp. 317–328.

[142] J. TERAN, E. SIFAKIS, G. IRVING, AND R. FEDKIW, *Robust quasistatic finite elements and flesh simulation*, in Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation, ACM, 2005, pp. 181–190.

[143] D. TERZOPOULOS AND K. FLEISCHER, *Deformable models*, The Visual Computer, 4 (1988), pp. 306–331.

[144] ——, *Modeling inelastic deformation: viscolelasticity, plasticity, fracture*, in ACM Siggraph Computer Graphics, vol. 22, 1988, pp. 269–278.

[145] D. TERZOPOULOS, J. PLATT, A. BARR, AND K. FLEISCHER, *Elastically deformable models*, in Computer Graphics (Proceedings of SIGGRAPH), vol. 21, 1987, pp. 205–214.

[146] D. TERZOPOULOS AND K. WATERS, *Physically-based facial modelling, analysis, and animation*, The journal of visualization and computer animation, 1 (1990), pp. 73–80.

[147] Y.-L. TIAN, T. KANADE, AND J. COHN, *Robust lip tracking by combining shape, color and motion*, in Proceedings of the 4th Asian Conference on Computer Vision, 2000, pp. 1040–1045.

[148] H. TRAN, F. CHARLEUX, M. RACHIK, A. EHRLACHER, AND M. HO BA THO, *In vivo characterization of the mechanical properties of human skin derived from mri and indentation techniques*, Computer methods in biomechanics and biomedical engineering, 10 (2007), pp. 401–407.

[149] P. VOLINO AND N. MAGNENAT-THALMANN, *Implicit midpoint integration and adaptive damping for efficient cloth simulation*, Computer Animation and Virtual Worlds, 16 (2005), pp. 163–175.

[150] J. VON DER PAHLEN, J. JIMENEZ, E. DANVOYE, P. DEBEVEC, G. FYFFE, AND O. ALEXANDER, *Digital ira and beyond: creating real-time photoreal digital actors*, in ACM SIGGRAPH 2014 Courses, 2014, pp. 1–384.

[151] A. WÄCHTER AND L. T. BIEGLER, *On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming*, Mathematical programming, 106 (2006), pp. 25–57.

[152] H. WANG, *A chebyshev semi-iterative approach for accelerating projective and position-based dynamics*, ACM Trans. Graph., 34 (2015), p. 246.

[153] H. WANG, J. O'BRIEN, AND R. RAMAMOORTHI, *Multi-resolution isotropic strain limiting*, Proc. of ACM SIGGRAPH Asia, 29 (2010), pp. 156:1–156:10.

[154] H. WANG AND Y. YANG, *Descent methods for elastic body simulation on the gpu*, ACM Transactions on Graphics (TOG), 35 (2016), p. 212.

[155] S.-L. WANG, W. H. LAU, AND S. H. LEUNG, *Automatic lip contour extraction from color images*, Pattern Recognition, 37 (2004), pp. 2375–2387.

[156] J. A. WEISS, B. N. MAKER, AND S. GOVINDJEE, *Finite element implementation of incompressible, transversely isotropic hyperelasticity*, Computer methods in applied mechanics and engineering, 135 (1996), pp. 107–128.

[157] M. WEST, *Variational integrators*, PhD thesis, California Institute of Technology, 2004.

[158] C. WU, D. BRADLEY, P. GARRIDO, M. ZOLLHÖFER, C. THEOBALT, M. GROSS, AND T. BEELER, *Model-based teeth reconstruction*, ACM Transactions on Graphics (TOG), 35 (2016), p. 220.

[159] C. WU, D. BRADLEY, M. GROSS, AND T. BEELER, *An anatomically-constrained local deformation model for monocular face capture*, ACM Transactions on Graphics (TOG), 35 (2016), p. 115.

[160] T. WU, *A computational framework for modelling the biomechanics of human facial expressions*, PhD thesis, ResearchSpace@ Auckland, 2013.

[161] F. XU, J. CHAI, Y. LIU, AND X. TONG, *Controllable high-fidelity facial performance transfer*, ACM Transactions on Graphics (TOG), 33 (2014), pp. 1–11.

[162] W. YANG, N. MARSHAK, D. SỲKORA, S. RAMALINGAM, AND L. KAVAN, *Building anatomically realistic jaw kinematics model from data*, The Visual Computer, 35 (2019), pp. 1105–1118.

[163] S.-H. YOON, J. LEWIS, AND T. RHEE, *Blending face details: Synthesizing a face using multiscale face models*, IEEE computer graphics and applications, 37 (2017), pp. 65–75.

[164] D. ZHAO, Y. L. LI, AND J. BARBIČ, *Asynchronous implicit backward euler integration*, Proc. EG/ACM Symp. Computer Animation, (2016).

[165] G. ZHONG AND J. E. MARSDEN, *Lie-poisson hamilton-jacobi theory and lie-poisson integrators*, Physics Letters A, 133 (1988), pp. 134–139.

[166] J. ZIMMERMANN, A. NEALEN, AND M. ALEXA, *Silsketch: automated sketch-based editing of surface meshes*, in Proceedings of the 4th Eurographics workshop on Sketch-based interfaces and modeling, 2007, pp. 23–30.

[167] G. ZOSS, D. BRADLEY, P. BÉRARD, AND T. BEELER, *An empirical rig for jaw animation*, ACM Transactions on Graphics (TOG), 37 (2018), pp. 1–12.