TOWARDS REALISTIC REAL-TIME PHYSICS-BASED SIMULATION

by

Anka He Chen

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

Doctor of Philosophy

in

Computing

Kahlert School of Computing

The University of Utah

December 2024

The University of Utah Graduate School

STATEMENT OF DISSERTATION APPROVAL

The dissertation of **Anka He Chen**

has been approved by the following supervisory committee members:

Cem Yuksel,	Chair(s)	**10/15/2024** Date Approved
Yin Yang,	Member	**10/15/2024** Date Approved
Mike Kirby,	Member	**10/11/2024** Date Approved
Sheldon Andrews,	Member	**10/11/2024** Date Approved
Giljoo Nam,	Member	**10/25/2024** Date Approved

by **Mary Hall**, Director of

the Karlert School of Computing

and by **Darryl P. Butt**, Dean of The Graduate School.

ABSTRACT

Stability, realism (accuracy), and efficiency (parallelism) are the three most crucial properties of a physics-based simulator. Despite substantial research and significant advancements in physics solvers over the past decades, current methods still struggle to deliver all three. Convergent methods provide precise solutions to physics equations, but they are less stable and usually require solving a global system, leading to poor parallel performance. Conversely, GPU-based parallel solvers, such as Extended Position Based Dynamics (XPBD), simplify the physics equations and omit important system information, resulting in a failure to converge to the true physics equations, losing realism. Stability, on the other hand, is always a challenge for all those methods, particularly with strict computation budgets.

This dissertation introduces a novel solver named Vertex Block Descent (VBD), a highly parallelizable solver for the variational form of implicit Euler through vertex-level Gauss-Seidel iterations. VBD operates with local vertex position updates that reduce global variational energy while maximizing parallelism. This creates a physics solver that achieves numerical convergence with unconditional stability and exceptional computational performance.

My work also includes accurate and efficient solutions for handling collisions. This dissertation also proposes a formal definition of the shortest boundary paths for self-intersecting objects and develops a robust algorithm for computing these paths. This method provides a well-defined collision energy for self-colliding models, effectively solving collisions in simulations of deformable volumetric objects. Then this collision resolution is extended to codimensional objects such as cloth and strands, implementing a universal collision energy applicable to various geometries, named Offset Geometric Contact (OGC). OGC offers computationally-efficient physics simulations with guaranteed collision-free results. It is more than 100x faster than the alternative methods such as the Incremental Potential Contact (IPC) method and it entirely avoids the non-orthogonal contact forces that IPC can generate, which can lead to severe visual artifacts. Consequently, the resulting contact energy of OGC is much less stiffer than IPC's. Coupled with a novel trust region based optimization method, it can simulate challenging contact scenarios with unprecedented complexity and do so within a matter of milliseconds.

My work made the physics-based simulation orders of magnitude faster than previous methods, while providing unconditional stability for and penetration-free guarantee in highly stressful simulations with extremely large numbers of collisions.

Additionally, my work includes creating high-resolution deformation capture systems to collect real-world data on non-rigid objects for studying registration. This enables driving simulations with real data and facilitates the study of inverse physics problems.

CONTENTS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Physics-based simulation is a fundamental component of most computer graphics applications, and the demands for improved stability, faster computational performance, and enhanced visual realism continue to grow. Despite substantial research and significant advancements in physics solvers over the past decades, current methods still fall short, typically meeting only one or a select few of these requirements. Particularly in real-time graphics applications, the stability and performance requirements are so strict that accuracy can sometimes be begrudgingly considered of secondary importance.

However, emerging fields such as AR/VR, artificial intelligence, robotics, and autonomous driving are placing a heightened emphasis on the accuracy of real-time simulators, because those applications usually involve interacting with the real world. At the same time, the efficiency and stability of solvers remain uncompromisable in these applications. This is because these applications are inherently real-time or require handling massive datasets or facilitating online training with the simulator running on the fly.

In traditional computer graphics applications focused on animation, the accuracy of the simulator is often measured by visual plausibility, as the primary goal is to produce animations that appear realistic to human observers and meet the aesthetic preferences of artists. This typically requires the simulator to at least roughly approximate the physical properties of the simulated object and provide a solution using a time integration formula derived from those properties. Often, the simulator does not converge to the ground-truth solutions of the time integration formula, but retains some residuals due to the limitations of the computational budget. This is usually okay, because humans are not capable of recognizing those residuals.

However, in applications where the simulator interacts with the real world or is required to provide accurate results that resemble the real world, the demand for accuracy becomes much more stringent. In these scenarios, accuracy is a 2-layer measurement. Firstly, the simulator must efficiently converge to the ground truth solutions of the time integration formula. This ensures that the simulator can replicate the real-world scenario with the given parameters with a reasonable degree of fidelity based on the precision of the time integrator and the material model used. However, accurately replicating a scenario does not guarantee that the simulation results will precisely mirror the real world, as inaccuracies in input parameters—such as material properties, collision and friction parameters, and constraints—can skew results. This is often referred to as inverse physics problems, where the simulator needs to deduce the simulation parameters based on real-world observation. The reliability of these parameters heavily depends on the simulator's ability to reach convergence; without accurate convergence, even well-estimated parameters may fail to produce valid results. Therefore, the second layer of accuracy depends on the first layer of accuracy.

1.1 Zero-Compromised Simulator

My work focuses on improving both layers of those accuracy measures, without sacrificing efficiency and stability. The most fundamental step would be to propose a new simulator that meets all of those previously mentioned requirements. We make this step by proposing a novel solver named Vertex Block Descent (VBD) [1], which is specifically designed to offer all the desired properties: highly efficient and parallel computation, unconditional stability and exceptional convergence rate.

VBD can adhere to strict real-time computation constraints by adjusting the iteration count, without compromising its stability. It utilizes a second-order block coordinate descent method applied at the vertex level through Gauss-Seidel iterations to solve the variational form of the implicit time integrator. For the dynamics of elastic bodies, the method iteratively processes each mesh vertex independently, adjusting its position while keeping all other vertices fixed. This approach allows for significant parallelism, particularly when integrated with vertex-based mesh coloring. This coloring technique can reduce the number of necessary serialized workloads by an order of magnitude compared to element-based parallelization strategies. Our local position-based updates can ensure that the variational energy is always reduced. As a result, our method remains stable even with a single iteration per timestep and can handle large timesteps with solutions that contain significant unresolved residuals. With additional iterations, the VBD method converges more rapidly than alternative solutions, making it more feasible to fit within specified computational budgets while still ensuring stability and enhanced convergence rates.

VBD is a general solution that can be used for a variety of simulation or geometry problems, we present and evaluate it in the context of elastic body dynamics. We provide all essential components of using VBD for elastic body dynamics, including formulations for damping, constraints, collisions, and friction. We also introduce a simple initialization scheme to warm-start the optimization and improve convergence. In addition, we discuss momentum-based acceleration techniques and parallelization in the presence of dynamic collisions.

1.2 Collisions of Volumetric and Codimensional Objects

With the new solver VBD, we can efficiently and accurately solve the simulation equations. However, the equations themselves may sometimes be improperly defined. We found that collision energy, particularly self-collision energy, has significant issues that cause deviations from real-world physics. Addressing these issues is essential for achieving more accurate simulations.

1.2.1 Proper Collision Energy for Self-Intersecting Volumetric Models

Although self-intersections are often highly undesirable, they are commonplace in computer graphics. They can appear due to the limitations of modeling techniques, animation methods, or manual editing operations. Even physics-based simulations with self-collision handling are not immune to self-intersections, as most of them cannot guarantee an intersection-free state.

Notwithstanding the amount of work on self-intersection handling within physics-based simulations, it still remains a challenge in most cases. *Continuous collision detection* techniques [2] require starting with and maintaining an intersection-free state; therefore, they must be used with computationally-expensive methods that can always resolve all self-intersections and they fail when combined with cheaper techniques that are unable to do so. Methods that split an object into pieces [3] turn the self-intersection problem into intersections of these separate pieces, entirely avoiding the self-intersection problem, and they fail to resolve self-intersections within a piece. Methods that solve self-intersections using an intersection-free pose [4] not only require such a pose, but also become inaccurate as the objects deform and fail with sufficiently large deformations and deep penetrations. Therefore, none of these methods provides a robust and general solution for self-intersections.

we present a method that robustly and efficiently finds the *exact shortest internal path* of a point inside a mesh to its boundary, even in the presence of self-intersections and some inverted elements. We achieve this by introducing a precise definition of the shortest path to the mesh boundary, including points that are both on the boundary and inside the mesh at the same time, an unavoidable condition with self-intersections. Our approach works with tetrahedral meshes in 3D (with boundaries forming triangular meshes) and triangular meshes in 2D (with polyline boundaries). We demonstrate that one important application of our method is solving arbitrary self-intersections after they appear in deformable simulations, allowing the use of cheaper integration techniques that do not guarantee complete collision resolution.

Our method is based on the realizations that (1) the shortest path must be fully contained within the geodesic embedding of the mesh and (2) it must be a line segment under Euclidean metrics. Based on these, given a candidate boundary point, our method quickly checks if the line segment to this point is contained within the mesh. Combined with a spatial acceleration structure, we can efficiently find and test the candidate closest boundary points until the shortest path is determined. We also describe a fast and robust tetrahedral traversal algorithm that avoids infinite loops, needed for checking if a path is within the mesh. Furthermore, we propose an additional acceleration that can quickly eliminate candidate boundary points based on local geometry without the need to check their paths.

1.2.2 Accelerated Penetration-Free Simulation for Codimensional Objects

The work on the shortest path in self-intersecting objects provides an effective solution for handling intersections in simulations of deformable volumetric objects. However, this penetration-depth-based solution does not work for simulation of codimensional objects, where penetration is strictly prohibited. Consequently, penetration depth-based collision energy cannot be used in these simulations.

To address these challenges, we extended the shortest path algorithm to codimensional objects. The basic idea behind this is to inflate the codimensional body to a volumetric body, thus defining penetration depth. However, guaranteeing penetration-free simulations remains a significant challenge.

Despite significant advancements in penetration-free simulations, such as Incremental Potential Contact (IPC) [2], a critical problem remains: the computational cost. IPC-based simulators and their derivatives are usually orders of magnitude more expensive than alternative methods that do not provide such a guarantee. Moreover, the computational cost of those methods depends on each time step's state and is highly uneven. These issues prevent penetration-free simulations from being used in many applications, especially those requiring real-time performance.

The problem comes from two main factors: the collision energy is very stiff, making convergence difficult, and expensive procedures such as line search and collision detection must be incorporated into every iteration of the simulation to ensure penetration-free conditions.

The stiffness arises from the necessity of preventing penetration. To ensure the contact force is always strong enough to push objects apart, it must be able to become infinitely large as objects approach each other. This requires the contact force to transition from zero to infinite within the contact radius. This issue becomes particularly severe when the contact radius is very small.

In this work, we identify a geometric limitation of IPC: the resultant normal contact force may not always be orthogonal to the surface, potentially leading to artifacts. To address this, IPC employs a scheme that adaptively reduces the contact radius during the optimization process until it becomes extremely small. However, this further increases the stiffness of the contact energy.

IPC uses continuous collision detection (CCD) based technique to prevent collision. This technique applies a CCD on the optimization step the simulation solver provides, and culls it before the earliest intersection happens. For GPU implementation, this procedure is a global operation that requires synchronization and hinders parallelism. CCD is applied at every iteration, which is very expensive. Moreover, the CCD-based intersection filter halts the global optimization step at where the earliest intersection happens, meaning a local intersection stops the progress of all other points, even if those points are still far from intersecting. This could reduce the solver's efficiency, since each iteration can be computationally expensive, and the shortened optimization step induced by CCD results in more iterations.

1.3 Obtaining High-Quality Data of Deformation to Calibrate the Simulator

Now that we have a simulator that accurately converges to the physics equations describing objects' dynamics and includes a robust collision handling scheme, the simulation may still not perfectly resemble reality. This discrepancy arises because the simulation parameters are manually set and might not match the real ones.

Direct measurement can obtain these parameters, but this is often impractical when the simulated object is inaccessible. Instead, deducing parameters from recorded deformation data is more feasible. However, not all data types are suitable for optimizing simulations. Deformable body physics, which relies on "deformation"—a mapping between a 3D template shape and its deformed state—requires specific data types.

Consumer-grade depth sensors like Kinect or RealSense capture high-resolution shapes of deformed objects, producing point clouds with millions of points. However, this unstructured data lacks a consistent template mesh and frame-to-frame correspondence, making it unsuitable for analyzing deformation energy. The required data must be temporally consistent, maintaining correspondences across frames and between the template and deformed shapes. This data tracks motion trajectories of points on the 3D template, whose structure remains consistent throughout deformation, enabling efficient and robust deformation analysis.

Two main methods obtain temporally consistent data. One involves starting with unstructured input and using template model registration to establish temporal correspondences. This method is computationally intensive, primarily focusing on solving correspondences and relying on accurate initial guesses, which can lead to error accumulation. The other method involves directly capturing correspondence through a specially designed capture system. I explored this approach in my Siggraph 2021 paper, where we focused on capturing the human body, a critical deformable object. Our system uses a specially designed pattern to add essential textural information to the human body, enabling direct temporal-correspondence solving through a deep-learning/graphics-coupled capture framework.

This work increases the number of captured points by 100 times compared to optical motion capture, providing a robust solution that reduces or eliminates manual post-processing. To our knowledge, it is the first motion capture system to directly capture thousands of moving points from the human body in motion.

CHAPTER 2

BACKGROUND

2.1 Fundamental Problem of Physics-Based Simulation

Implicit time integrator are widely accepted as the primary methods for simulating elastic bodies in computer graphics due to their exceptional stability, especially when addressing stiff problems. Among these options, backward Euler [5, 6, 7, 8] is the most commonly utilized method.

Given a system with K vertices, we represent the state of our simulation at step t as $(\mathbf{x}^t, \mathbf{v}^t)$, where $\mathbf{x}^t \in \mathbb{R}^{3N}$ and $\mathbf{v}^t \in \mathbb{R}^{3N}$ are the concatenated position and velocity vectors, respectively. The basic form of backward Euler is:

$$\begin{cases} \mathbf{x}^{t+1} & = \mathbf{x}^t + h\mathbf{v}^{t+1} \\ \mathbf{v}^{t+1} & = \mathbf{v}^{t+1} + h(\mathbf{a}_{\text{ext}} + M^{-1}\mathbf{f}(\mathbf{x}^{t+1}, \mathbf{x}^{t+1})) \end{cases} \tag{2.1}$$

where M is the mass matrix, $\mathbf{a}_{\text{ext}}$ is the fixed external acceleration, h is the time step, $\mathbf{f}$ is the forces applied on objects that are a function of the future positions $\mathbf{x}^{t+1}$ and future velocities $\mathbf{v}^{t+1}$. For simplicity, we omit the time step and write $\mathbf{x}^{t+1}$ and $\mathbf{v}^{t+1}$ as $\mathbf{x}$ and $\mathbf{v}$.

We observe that $\mathbf{x}$ and $\mathbf{v}$ have a linear relation. Therefore we can substitute $\mathbf{v}$ using $\mathbf{x}$, and rewrite Equation 2.1 as:

$$\frac{1}{h^2}M(\mathbf{x} - \mathbf{y}) + \mathbf{f}(\mathbf{x}) = 0\,, \tag{2.2}$$

and

$$\mathbf{y} = \mathbf{x}^t + h\mathbf{v}^t + h^2\mathbf{a}_{\text{ext}} \tag{2.3}$$

is the inertia of the system. Equation 2.2 is the non-linear equation of $\mathbf{x}$, where the non-linearity comes from the formulation of $\mathbf{f}(\mathbf{x})$. Intuitively, this problem can be solve by linearizing $\mathbf{f}$ [5], and solve the corresponding linear equation:

$$\Delta\mathbf{x} = -(\nabla\mathbf{f})^{-1}(\frac{1}{h^2}M(\mathbf{x} - \mathbf{y}) + \mathbf{f})\,, \tag{2.4}$$

then $\mathbf{x}^{\text{new}} = \mathbf{x} + \Delta\mathbf{x}$ will be the position that satisfies Equation 2.2 assuming linearity of $\mathbf{f}$. However, in real applications, $\mathbf{f}$ can be very nonlinear. Therefore, directly taking the full step $\Delta\mathbf{x}$ may not be stable. To solve this problem, we can convert Equation 2.2 into a optimization problem:

$$\mathbf{x} = \operatorname*{argmin}_{\mathbf{x}} G(\mathbf{x}) \ , \tag{2.5}$$

where $G(\mathbf{x})$ is called the *variational energy*:

$$G(\mathbf{x}) = \frac{1}{2h^2}\|\mathbf{x} - \mathbf{y}\|_M^2 + E(\mathbf{x}) \ . \tag{2.6}$$

where

$$E(x) = E_e(x) + E_n(x) + \dots \tag{2.7}$$

is a potential energy such that $\mathbf{f} = \nabla E$. $E(x)$ can consist of multiple energies, such as elastic energy E_e and collision energy $E_n(x)$. Note that this requires $\mathbf{f}$ to contain only conservative forces. Furthermore, this optimization can contain constraints:

$$G(\mathbf{x}) = \frac{1}{2h^2}\|\mathbf{x} - \mathbf{y}\|_M^2 + E(\mathbf{x}) \tag{2.8}$$

$$\text{s.t. } \mathbf{a}(x) = 0 \tag{2.9}$$

$$\mathbf{b}(x) \leq 0 \tag{2.10}$$

$$\mathbf{c}(x) > 0 \ , \tag{2.11}$$

where $\mathbf{a}$ is bilateral constraints and $\mathbf{b}(x)$ and $\mathbf{c}(x)$ are unilateral constraints. The constraints puts special requirements to the trajectory of the vertices. A speicific case of the unilateral constraint is a penetration-free constraint:

$$d_{i,j}((1-\lambda)\mathbf{x}^t + \lambda\mathbf{x}) > 0), \forall i \neq j, i, j \in \mathcal{B}; \lambda \in [t_0, t_0 + h] \tag{2.12}$$

where $d_{i,j}$ is the distance between two primitives i, j, and $\mathcal{B}$ is the set of all the surface primitives of the object. This constraint ensures that the object does not have an intersection at any time between step t and step $t + 1$.

The optimization form allows a lot of optimization techniques to be applied to solving Equation 2.8 to ensure stability, such as doing a line search on $\Delta\mathbf{x}$, or applying a position-definite projection on $\nabla\mathbf{f}$ to make sure that $\Delta\mathbf{x}$ is actually a descent direction.

2.2 Solvers for Physics-Based Simulation

There is a large body of work on physics-based simulation in computer graphics. Here we only discuss the recent and the most relevant work.

Implicit time integrators are widely accepted as the primary methods for simulating elastic bodies in computer graphics due to their exceptional stability, especially when addressing stiff problems. Among these options, backward Euler [5, 6, 7, 8] is the most commonly utilized method, though other approaches like implicit Newmark [9, 10, 11], BDF2 [12, 13], and implicit-explicit [14, 15] have also been explored. Backward Euler is often approximated as a single Newton step, solving a linear system of equations [5]. Line search can be applied to improve robustness [6]. Preconditioning [16] or positive-definite projection [17] can be used to improve convergence. To circumvent a full linear solve for every Newton step, Cholesky factorization [18] emerges as a viable strategy. Techniques like multi-resolution [19, 20] or multigrid [21, 22, 23, 24, 25] solvers project finer details onto a coarser grid with fewer degrees of freedom, effectively reducing the computational cost of the linear system solver.

Additionally, stiffness warping [26] repurposes the stiffness matrix from the rest shape to handle significant rotational deformations. Using a quasi-Newton method [27] with an approximate Hessian can be far more cost-effective for computing its inverse with prefactoring than the actual Hessian. Example-based dynamics has also been explored [28, 29, 8], where the deformation energy is defined based on the nearest point in the example space. Projective Dynamics [30] represents deformation energy via a series of constraints that can be solved independently, then synchronized either through a prefactorized global step to accelerate convergence.

The relation between dynamics, energy, and minimization has been leveraged in variational integrators [9, 31, 32, 33, 34]. Reformulating backward Euler to an optimization problem combined with optimization techniques to allows the usage of large steps [8, 35]. Domain-decomposed optimization for solving the nonlinear problems of implicit numerical time integration can accelerate convergence [36]. Yet, the optimization formulation has its drawbacks, notably the varying problem formulation and initial positions in each step. Consequently, achieving consistent convergence within a fixed time budget becomes challenging. Real-time simulators compromise by accepting partially converged results, which visually resemble the final solution. Unfortunately, this compromise can lead to significant visual artifacts mainly due to retained residuals from earlier steps, which can accumulate across frames and can jeopardize the solver's stability. In contrast, our method VBD, demonstrates exceptional stability even when retaining a substantial amount of residual with a limited number of iterations.

Position-based dynamics methods (PBD [37] and XPBD [38]) convert forces into (soft) constraints and directly update the positions with Gauss-Seidel iterations operating on one constraint at a time. These position-based updates result in exceptional stability, which is often exploited by limiting the number of iterations to fix the computation cost, an effective strategy for real-time simulations. Akin to PBD, VBD also works with position updates, but operates directly using the force formulations without converting them to constraints. Parallelization with XPBD is achieved by graph coloring the constraints (i.e., the dual graph) [39, 40, 41]. However, this dual graph contains multiple times more connections (depending on the types of constraints) than the original graph of vertices, severely limiting the level of parallelism. In comparison, our method VBD is parallelized by coloring the original graph, which leads to much fewer colors (i.e., computation groups that must be processed sequentially) and thereby better parallelism. More importantly, the approximations in XPBD's formulation introduce errors that make it diverge from the solution of implicit Euler and can degrade realism, particularly with large time steps and limited iteration count, which are common in practice. In addition, XPBD particularly struggles with high mass ratios. Our method VBD, on the other hand, has none of these problems.

In recent years, a growing effort has been placed on accelerating simulations using GPUs [42, 43, 44, 45, 46, 47, 48]. Among them, the first-order descent methods [44, 46] have gained popularity due to their excellent parallelism. These methods employ a Jacobi-style preconditioned first-order descent on the backward Euler minimization formulation, enabling full vertex-level parallelization. However, Jacobi-style iterations typically converge substantially slower than Gauss-Seidel iterations. Also, such methods necessitate a line search to avoid overshooting and ensure stability.

Our method VBD can be categorized as a coordinate descent method for optimization [49]. In graphics, this technique has been used for geometric processing [50] and simulation with a barrier function [51]. Recently, [51] proposed a hybrid scheme where Gauss-Sediel and Jacobi iterations are combined at each parallel call. In comparison, VBD uses blocks of coordinates-based on vertices instead of blocks-based on elements, which results in much better parallelism and smaller local linear systems to solve, leading to faster convergence.

2.3 Collision Resolution

Collisions can be resolved by minimizing the penetration volume [52, 53] or by applying constraints [30, 37, 38, 54], penalty forces [55, 56, 57, 58], or impulses [59, 60, 61]. Here we emphasize self-collisions of deformable objects and methods that guarantee (self-)intersection-free results.

2.3.1 Self Collision

Self-intersecting meshes are a common yet often unwanted phenomenon in computer graphics, arising from the constraints of modeling tools, animation techniques, or manual modifications. Despite advancements in physics-based simulations equipped with self-collision handling capabilities, achieving an entirely intersection-free state remains challenging. Consequently, addressing self-intersections is essential for ensuring effective and reliable collision handling.

Starting with a self-intersecting shape, [62] proposed a method to separate the overlapping parts and create a bounding case mesh that represents the underlying geometry to allow ”un-glued” simulation.

Methods that solve self-intersections using an intersection-free pose [4] not only require such a pose, but also become inaccurate as the objects deform and fail with sufficiently large deformations and deep penetrations.

Methods that split an object into pieces [63, 64, 4, 3, 65] turn the self-intersection problem into intersections of these separate pieces, entirely avoiding the self-intersection problem, and they fail to resolve self-intersections within a piece. These methods circumvent the issue of self-intersections by focusing solely on the intersections between distinct pieces, overlooking any self-intersections within those pieces. This approach becomes particularly challenging for complex models or situations where it’s not clear how to effectively divide the model beforehand. Dividing or bifurcating the model tends to be pre-determined and costly to modify during runtime. Moreover, identifying the closest boundary point within a piece may not yield the true nearest boundary for the whole mesh if it resides in another piece. While sufficient accuracy may be achievable in certain instances, the use of Signed Distance Fields (SDFs) incurs substantial pre-computation and storage demands.

In comparison, we provide a solution [66] to find the *exact* penetration depth for models with arbitrary complexity and the accurate shortest path to the boundary regardless of the type or severity of self-intersections. In addition, we do not require costly pre-computations or volumetric storage. Our approach leverages two critical insights: (1) the shortest path must reside entirely within the mesh's geodesic embedding, and (2) it should form a straight line under Euclidean geometry. Utilizing these principles, our method swiftly verifies whether a straight line to a given boundary point remains inside the mesh. By integrating a spatial acceleration structure, we can effectively identify and evaluate potential boundary points until we pinpoint the shortest path. Additionally, we introduce an efficient and reliable tetrahedral traversal technique that precludes endless loops, essential for confirming path containment within the mesh. Moreover, we present a novel acceleration technique that rapidly dismisses unsuitable boundary points by considering the local geometry, thus obviating the need for path verification.

2.3.2 Penetration Free Deformable Body Simulation

Penetration-free simulation is a recent breakthrough in the physics-based simulation community. The simulation of deformable bodies is usually done by minimizing the implicit integration equation, with the collisions modeled as potential energies, or additionally, as constraints to the minimization problem. To strictly guarantee penetration-free, the collision must be formulated as non-compliable constraints. In the physics-based simulation community, those non-compliable constraints are usually enforced through two groups of methods: the line search based methods and the trust region methods.

2.3.2.1 Line Search Based Methods

The most representative work of the line search based method is the incremental potential contact [2]. It models collision energy as a logarithmic function on the distance between two primitives:

$$b(d,\hat{d}) = \begin{cases} -(d-\hat{d})^2 ln(\frac{d}{\hat{d}}) & 0 < d < \hat{d} \\ 0 & d \geq \hat{d} \end{cases} . \tag{2.13}$$

This energy (and the consequent contact force derived from it) goes to infinity as primitives draw nearer to each other, ensuring that the contact force will ultimately overpower any external or internal forces acting on those primitives, thus pushing them apart. However, this energy alone is not sufficient to guarantee penetration-free. This is because, in the process of solving the optimization, the steps are always taken discretely. The reason lies in the optimization process, which proceeds through discrete steps. It's possible that following an optimization step, primitives might end up penetrating each other. This occurs because the optimization often involves linearizing the problem, and the direction of descent determined through this linearization may not always effectively separate all approaching pairs of primitives.

To enforce the penetration-free condition, IPC requires optimization to halt before the earliest time of impact (TOI), determined via CCD-aware line search. The process involves iteratively recomputing contact relationships, descent directions, and CCD checks until convergence. [67] later extended this IPC collision model to codimensional objects, e.g., elastic rods and surfaces. This includes several novel improvements: modeling thickness, a generalized CCD that adapts to this thickness modeling, and another barrier function to limit the stretching.

CCD-aware line search requires linear motion at each optimization iteration, which is not satisfied for systems with rotational components like rigid body dynamics. [68] addressed this by dividing rotational motion into small linear segments for CCD, but this incurs more computation steps. [69] improved this by using affine transformations instead of SE(3) movements, turning rotational motion into linear affine motion. This approach eliminates the need for multiple segments, requiring only one CCD application per step, greatly enhancing simulation efficiency.

Various methods have been employed to enhance IPC simulation efficiency. [45] replaced IPC's Newtonian solver with projective dynamics, enabling penetration-free GPU simulations by reformulating IPC's barrier constraint with projected target positions. [70] introduced a block coordinate descent technique with element-based Gauss-Seidel iteration and local CCD to reduce computational costs. [71] developed a GPU-accelerated Gauss-Newton method to accelerate simulations using barrier contact energy. [72] simplified original geometry with standard shapes to reduce collision pairs and speed up simulations, sacrificing fine geometric details.

2.3.2.2 Trust-Region-Based Methods

In numerical optimization, a trust region defines a subset of the domain where the objective function is approximated, typically using a quadratic model [73]. The region adapts dynamically: expanding if the model proves accurate and contracting if it does not, enabling efficient optimization. Trust-region methods can be considered somewhat complementary to line-search methods: trust-region approaches initially determine a step size (the dimensions of the trust region) and subsequently select a step direction. In contrast, line-search methods start by choosing a step direction and then decide on the step size.

It is known that trust-region based methods are better suited for constrained optimization problems where constraint satisfaction is critical [74]. Constraints can be incorporated directly into the trust region formulation [75, 76], which are usually linear and convex [77], [78].

However, trust region methods for enforcing penetration-free constraints have not been extensively explored in the simulation community. Unlike IPC, which combines CCD with line search to enforce penetration constraints, trust-region methods use *discrete collision detection* (DCD) to define per-vertex (or per-rigid-body) trust regions, constraining movements to prevent penetration.

This kind of idea was initially explored in the context of rigid body dynamics, termed Conservative Advancement. [79] uses the extremal vertex query to find a directed motion bound for an object moving with constant translational and rotational velocities. [80] extends this kind of method to triangulated models and makes no assumption about the underlying geometry and topology.

This idea has also been investigated in the context of cloth simulation. [81] identified a necessary vertex displacement constraint to prevent cloth from self-intersecting, thus ensuring the avoidance of self-penetration at all times. [82] utilized this constraint within a step-and-project process to facilitate fast and realistic simulation.

However, the development of a trust region-based simulation system that incorporates barrier collision energy and ensures numerical convergence remains an open area of research.

2.3.2.3 Offset Geometry

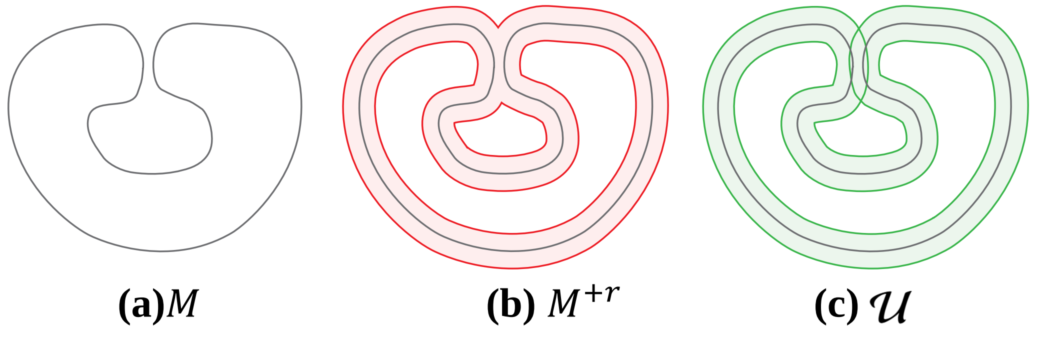


Figure 2.1: (a) original geometry M; (b) conventional offset geometry M^{+r}; (c) our intersection-aware manifold $\mathcal{U}$.

Offset geometries, also known as polygon offsets or Minkowski dilation, are geometric constructions where a polygon is expanded or contracted by a specified distance. The geometry M's offset geometry with distance r is defined as:

$$M^{+r} = \{\mathbf{x} \in \mathbb{R}^N | dis(\mathbf{x}, M) < r\} \tag{2.14}$$

The boundary of this offset geometry can be computed through various method, including winding number based methods [83], Voronoi Diagram-Based Methods [84], straight skeleton based methods [85, 86], polygonal annulus based methods [87].

However, those methods mainly focus on 2D polygons instead of 3D polyhedral meshes. Moreover, The conventional concept of offset geometry presents significant challenges when applied to contact modeling. Specifically, traditional offset methods often fail to accurately represent self-intersections and overlapping regions within the geometry, as illustrated in Figure 2.1b. The offset geometry given by Equation 2.14 will "merge" parts that are separated in the original manifold. This occurs when a point's distance to two separate parts of M is both less than r. This limitation can lead to inaccuracies in simulating contact interactions, as the model may not correctly account for multiple layers of contact and self contacts.

Ideally, the offset geometry should be aware that there are 2 overlapping layers, as illustrated in Figure 2.1c, and the point in the overlapping area should be subject to contact forces from the opposite side. The dimensionality of the offset geometry in Figure 2.1c has been lifted to N (the dimensionality of the immersion space), and therefore is not codimensional anymore. Intuitively, we can determine the layers of intersection, and compute the penetration depth using a method akin to [66] to compute contact energy.

2.3.2.4 Gauss Map

[88, 89] extended the Gauss-Bonnet theorem to polyhedral surfaces by introducing a method to compute curvature at vertices using the Gauss map. [90] further expressed curvature measures in terms of the number of critical points. [91] introduced the concept of Extended Gaussian Images (EGI) for object recognition by projecting the normal vectors of a polyhedron's faces onto a sphere, assigning densities proportional to the corresponding face areas. [92] proposed a variation of EGI where normal vector lengths are proportional to face areas, investigating its uniqueness for convex polyhedra and its application in reconstructing objects using the Minkowski theorem. This approach requires defining face orientations, known as combinatorial types, and an iterative process for 3D reconstruction. [93] estimated curvature for polygonal surfaces using normal cycles at vertices, edges, and triangles, providing error bounds for discrete surfaces derived from restricted Delaunay triangulation. Building on these, [94] propose a novel approach to curvature measurement that distinguishes positive and negative components, enabling accurate vertex characterization. Their Polyhedral Gauss Map directly correlates normal vectors from the polygonal mesh, reflecting vertex geometry and their local neighborhoods more precisely.

2.4 Capturing Deformation of Non-Rigid Objects

Optical-systems-based on reflective markers [95] are the most widely used approaches to capture the human body. While typically only sparse marker sets are used, [96, 97] pushed the resolution of reflective markers-based system up to 350 markers to capture the detailed skin deformation. However, difficulties in marker labeling [98] complicate further increases of resolution by adding even more markers. Recent work utilizes self-similarity analysis [99] and deep learning [100, 101] to reduce the expensive manual clean-up in marker labeling procedure. An alternative to the classical reflective markers is the use of colored cloth, enabling the capture of certain types of garments [102, 103] or hand tracking using colored gloves [104].

Early work in markerless motion capture [105] and [106] inferred human poses directly from 2D images or videos. [107] integrates multiple image cues such as edges, color information and volumetric reconstruction to achieve higher accuracy. [108] tracks a 3D human body on 2D image by combining image segments, optical flows and SIFT features [109]. [110] deforms laser-scan of the tracked subject under the constraints of multi-view videos to capture spatio-temporally coherent body deformation and textural surface appearance of the actors. Silhouettes [111, 112, 113, 114] or visual hulls [115] can be used to obtain more detailed human body deformations. [116] introduce a multi-layer framework that combines stochastic optimization, filtering, and local optimization to tack 3D human poses. [117] model the human body via a sums of Gaussians, representing both shape and appearance of the captured actors.

Deep learning enabled estimation of 2D or 3D human poses from multi-view [118] or monocular multi-person images [119, 120, 121, 122], more recently also with hands and faces [123, 124, 125]. 3D pose or even dense 3D surface of the human body can also be predicted from a single image [126, 127, 128, 129, 130]. Morphable human models can be learned from multi-person datasets [131, 132]. Models such as SCAPE [133], SMPL [134] and STAR [135] focus on the body, while models such as Adam [136] and SMPL-X [137] also include the face and the hands. Focusing on high-quality rendering rather than geometry, [138, 139] proposed methods for photo-realistic relighting of moving humans, including clothing and accessories such as backpacks.

The idea of a motion capture suit with special texture is related to fiducial markers used e.g., in robotics or augmented reality, such as ARTag [140], AprilTag [141, 142], ArUco [143] and many others, but these fiducial markers are typically assumed to be non-deforming. They are also not easy to read for humans, which would complicate their annotations. The localization of our fiducial marker is related to corner detection. Many corner detection methods have been developed to meet different use-case scenarios. For localizing the corners, there are methods that are designed to detect general corners features that naturally exists in nature, like [144, 145], Another class of corner detectors focuses on rigid calibration checkerboards [146, 147, 148, 149], particularly useful in camera calibration. The code recognition component of our method is related to text recognition. As discussed in [150], the text recognizer generally performs poorly on text with large spatial transformations. One possible solution is based on generating region proposals [151, 152] to rectify the spatial transformation.

High-resolution temporal correspondences can be obtained by registering a template mesh to RGB-D images or 3D scans. The registration can be based on solely geometric information [153, 154], or combined with RGB images to align [155] to reduce the tangential sliding. Model-less approaches are also possible [156, 157, 158]. Those methods focus on registering sequential motions frame to frame, with the assumption of small displacements between subsequent frames. Therefore, they can suffer from error accumulation, resulting in drift over time [159]. Aligning non-sequential motions is also possible [160, 161], but it is challenging to establish correspondence between very different poses [162, 163]. Deformation models can be trained from 3D scans [164, 133], with non-rigid scan registration being the technical challenge [165].

Similarly to our new motion capture suit, the FAUST [166] and DFAUST [167] methods paint high-frequency colored patterns directly to the skin. We chose to work with a suit because putting it on and off is easy and fast compared to applying colored stamps and washing them off after the capture session. Our capture system is significantly simpler and less expensive: we use only on 16 standard (RGB) cameras with passive uniform lights, while [166, 167] used 22 pairs of stereo cameras, 22 RGB cameras and 34 speckle projectors (active light). Perhaps more important are the technical differences between our approach and DFAUST, in particular the fact that our codes are unique as opposed to the self-repetitive patterns used in FAUST and DFAUST. Rather than creating a dataset, our goal was to create a universal and practical method to enable future research on advanced human body modeling and its applications in areas ranging from graphics to sports medicine.

CHAPTER 3

VERTEX BLOCK DESCENT

This section proposes an efficient, stable, and parallel solver for the fundamental problem in physics-based simulation: Equation 2.8. The solver is called vertex block descent (VBD), which is essentially a solver for optimization problems or non-linear equations. Therefore, it can be applied to various simulation problems, particularly if they can formulated as an optimization problem. In section Section 3.1, we explain our method in the context of elastic body dynamics for objects represented by a set of vertices that carry mass and a set of force elements and constraints that act on them. Generalization of our method to other example simulation problems is discussed later in Section 3.3.

3.1 Vertex Block Descent for Elastic Bodies

We begin with deriving our global optimization method that splits the simulated system into vertex-level local systems (Section 3.1.1). Then, we discuss the methods we use for solving the local systems (Section 3.1.2) and describe how we incorporate damping (Section 3.1.3) and constraints (Section 3.1.4). We present our collision formulation (Section 3.1.5) and friction formulation (Section 3.1.6). After explaining our methods for warm-starting our optimization to improve convergence (Section 3.1.7), we describe how to incorporate momentum-based acceleration to improve convergence (Section 3.1.8). Finally, we discuss the improved parallelization that our method provides along with methods for efficiently parallelizing dynamically-introduced force elements due to varying collision events (Section 3.1.9).

3.1.1 Global Optimization

We propose an optimization technique that falls under the category of coordinate descent methods to efficiently minimize this energy G in Equation 2.8. If we only modify a single vertex at a time, fixing all other vertices, the part of the energy term $E(\mathbf{x})$ that is affected only includes the set of force elements $\mathcal{F}_i$ that are acting on (or using the position of) vertex i. Thus, we define the *local variational energy* G_i around vertex i as

$$G_i(\mathbf{x}) = \frac{m_i}{2h^2}\|\mathbf{x}_i - \mathbf{y}_i\|^2 + \sum_{j\in\mathcal{F}_i} E_j(\mathbf{x}) \ , \tag{3.1}$$

where m_i is the mass of the vertex and E_j is the energy of force element j in $\mathcal{F}_i$.

Note that G is not equal to the sum of local variational energies, i.e., $G(\mathbf{x}) \neq \sum_i G_i(\mathbf{x})$, simply because the force elements appear multiple times in this sum (once for each of its vertices). However, when we modify the position of a single vertex only, the reduction in G_i is equal to the resulting reduction in G.

Based on this observation, our method operates on a single vertex at a time and updates its position by minimizing the local energy

$$\mathbf{x}_i \leftarrow \underset{\mathbf{x}_i}{\operatorname{argmin}}\ G_i(\mathbf{x}) \tag{3.2}$$

and solves the global system using Gauss-Seidel iterations. Each local minimization for a vertex effectively finds a descent step for G using the degrees of freedom (DoF) for the vertex as a block of coordinates, hence the name *Vertex Block Descent* (VBD). After each iteration, the total reduction in G is equal to the sum of all reductions in G_i. In other words, the energy change of each vertex position adjustment is accumulated to the system energy. Consequently, if we can make sure that each local energy drops G_i when we are adjusting vertex i, we can guarantee that we are descending the system energy G.

Thus, our system directly operates on vertex positions and the resulting velocities are calculated following the implicit Euler formulation

$$\mathbf{v}^{t+1} = \frac{1}{h}\left(\mathbf{x}^{t+1} - \mathbf{x}^t\right) \ . \tag{3.3}$$

3.1.2 Local System Solver

The position updates per vertex (in Equation 3.2) involve minimizing a local energy that only depends on the position change of a single vertex, represented by G_i. Note that Equation 3.1 only has 3 DoF, so the cost of evaluating and inverting its Hessian is much cheaper compared to the global problem in Equation 2.6. Therefore, we can fully utilize the second-order information and use Newton's method to minimize the localized energy G_i, which involves solving the 3D linear system

$$\mathbf{H}_i \, \Delta\mathbf{x}_i = \mathbf{f}_i \,, \tag{3.4}$$

where $\Delta\mathbf{x}_i$ is the change in position, $\mathbf{f}_i$ is the total force acting on the vertex, calculated using

$$\mathbf{f}_i = -\frac{\partial G_i(\mathbf{x})}{\partial \mathbf{x}_i} = -\frac{m_i}{h^2}(\mathbf{x}_i - \mathbf{y}_i) - \sum_{j \in \mathcal{F}_i} \frac{\partial E_j(\mathbf{x})}{\partial \mathbf{x}_i} \,, \tag{3.5}$$

and $\mathbf{H}_i \in \mathbb{R}^{3\times 3}$ is the Hessian of G_i with respect to the DoF of vertex i, such that

$$\mathbf{H}_i = \frac{\partial^2 G_i(\mathbf{x})}{\partial \mathbf{x}_i \partial \mathbf{x}_i} = \frac{m_i}{h^2}\mathbf{I} + \sum_{j \in \mathcal{F}_i} \frac{\partial^2 E_j(\mathbf{x})}{\partial \mathbf{x}_i \partial \mathbf{x}_i} \,. \tag{3.6}$$

Here, the first term is a diagonal matrix and the second one is the sum of Hessians of the force elements with respect to vertex i. Intuitively, the solution of this linear system is the extreme point of the quadratic approximation for the localized energy G_i. By reducing G_i with each iteration, we can guarantee a reduction in G, thereby iteratively solving the global system in Equation 2.6.

We can analytically solve this system using $\Delta\mathbf{x}_i = \mathbf{H}_i^{-1}\mathbf{f}_i$. For such a small system, the analytical solver is very efficient and stable. We found it to be faster than solvers based on Conjugate Gradient or LU/QR decomposition. Another advantage of the analytical solver is that it does not require the Hessian to be positive-definite. Of course, when the Hessian is not positive-definite, the direction given by Equation 3.4 may not be a descent direction. Nevertheless, we opt for this direction regardless, recognizing that even when $\mathbf{H}_i$ is not positive-definite, solving Equation 3.4 still brings us towards the extremum of the quadratic approximation for the localized energy G_i. This solution is close to where the gradient of the inertia and the potential terms balance out and it is usually a stable state of the system. Also, the motivation of the variational form of implicit Euler (Equation 2.6), is to find a point where $dG(\mathbf{x})/d\mathbf{x} = 0$. Therefore, any extreme point is a valid solution of implicit Euler, and it does not have to be a local minimum. In all our experiments, including those specifically designed stress tests (see Figure 3.1 and Figure 3.2), we have consistently observed that this scheme does not pose any issues concerning system stability or convergence.

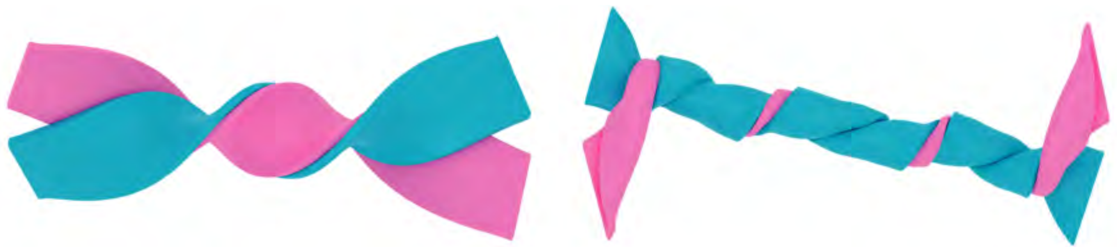


Figure 3.1: Twisting two beams together, totaling 97 thousand vertices and 266 thousand tetrahedra, demonstrating complex frictional contact and buckling.

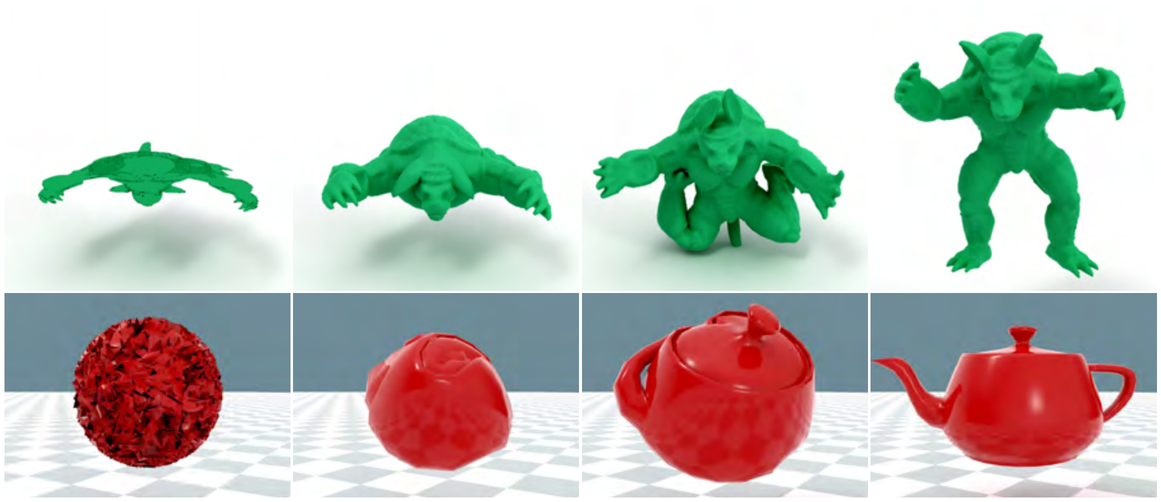


Figure 3.2: Stress tests that begin simulations under extreme deformations: (top row) a perfectly flattened armadillo model with 15 thousand vertices and 50 thousand tetrahedra, and (bottom row) a Utah teapot model with 2 thousand vertices and 8.5 thousand tetrahedra, deformed by randomly placing its vertices onto the surface of a sphere. Both models quickly recover to their original shape shortly after the simulation starts. Both simulations use accelerated iterations with $\rho = 0.95$.

An alternative solution to this is the PSD Hessian projection [17]. However, it is exceptionally rare for the Hessian to not be positive-definite, and the PSD projection process is notably costly due to multiple SVD decompositions. Engaging in this costly operation to prevent such rare events seems unjustified, especially considering that these occurrences do not jeopardize system stability or convergence significantly.

Another challenge with the analytical solver arises when encountering a (nearly) rank-deficient Hessian. To address this, we propose a simple solution: if $|\det(\mathbf{H}_i)| \leq \epsilon$ for some small threshold ϵ, we opt to bypass adjusting this particular vertex for that iteration. Given that its neighboring vertices are likely to undergo adjustments before the next iteration, it is improbable that its Hessian will remain rank-deficient in subsequent iterations. With this simple solution, in the extreme scenario where all vertices possess a rank-deficient Hessian, the system could potentially become frozen. Yet, it is crucial to note that such a case is highly improbable, since the Hessian of the inertia potential is always full-rank. One potential remedy for this would be switching to the modified Conjugate Gradient solver [51] when such a case happens. However, doing this will add additional branching to the code and can slow down the solver. Thus, we have not included it in our implementation, but, depending on specific use cases, there is always the flexibility to opt for the Conjugate Gradient solver as a backup solution.

The linear system in Equation 3.4 corresponds to a single Newton step, so it does not necessarily provide the optimal solution for Equation 3.2. In fact, since it is just a second-order approximation of G_i, it does not even guarantee a reduction in G_i. To ensure the descent of energy with this single step, we can incorporate a backtracking line search along the descent direction. Note that, unlike global line searches in descent-based simulation methods (e.g., [44]), this line search operates locally. It specifically verifies the descent of G_i, which in turn guarantees a descent in G without the need for evaluating the global system.

Line search avoids over-shooting and, thereby, ensures stability. In practice, however, the additional computation cost of line search may not be justified. In our experiments, we found that line search costs an extra 40% computation time, while not providing any measurable benefits. This is because VBD can maintain stability even without line search. Therefore, the results we present in this paper do not include line search, though it is an option available.

3.1.3 Damping

Damping plays a crucial role in simulations. It prevents excessive oscillations and also enhances system stability. Despite the inherent numerical damping introduced by the implicit Euler method, providing users with manual control over damping is highly desirable. To address this, we have integrated a simplified Rayleigh damping model into our solver [168]. This process is both straightforward and efficient, as it also operates locally within the 3×3 system and utilizes the precomputed stiffness matrix.

Since we are relying on implicit Euler, we can represent the velocity as position change, using $\mathbf{v}_i = (\mathbf{x}_i - \mathbf{x}_i^t)/h$. Then, we can add the damping term to the Hessian in Equation 3.6, resulting

$$\mathbf{H}_i = \frac{m_i}{h^2}\mathbf{I} + \sum_{j \in \mathcal{F}_i} \frac{\partial^2 E_j(\mathbf{x})}{\partial \mathbf{x}_i \partial \mathbf{x}_i} + \left(\sum_{j \in \mathcal{F}_i} \frac{k_d}{h} \frac{\partial^2 E_j(\mathbf{x})}{\partial \mathbf{x}_i \partial \mathbf{x}_i} \right) , \tag{3.7}$$

where k_d is the damping coefficient. Finally, we add the damping force to Equation 3.5 using the same damping term, such that

$$\mathbf{f}_i = -\frac{m_i}{h^2}(\mathbf{x}_i - \mathbf{y}_i) - \sum_{j \in \mathcal{F}_i} \frac{\partial E_j(\mathbf{x})}{\partial \mathbf{x}_i} - \left(\sum_{j \in \mathcal{F}_i} \frac{k_d}{h} \frac{\partial^2 E_j(\mathbf{x})}{\partial \mathbf{x}_i \partial \mathbf{x}_i} \right) (\mathbf{x}_i - \mathbf{x}_i^t) . \tag{3.8}$$

3.1.4 Constraints

Since our method directly manipulates the position of each vertex, managing a constraint on a vertex becomes straightforward. Constraints generally fall into two categories: unilateral ($C(\mathbf{x}) \leq 0$) or bilateral ($C(\mathbf{x}) = 0$). With bilateral constraints, if a vertex position is directly set to a specific value, we simply skip updating its position. Otherwise, it is constrained to a (usually linear) subspace. Our approach involves representing the constrained vertex position using the subspace basis. This transforms both the vertex position and gradient into an L-dimensional vector, where L is the subspace dimension. Consequently, handling local steps for constrained vertices involves solving an $L \times L$ system. Regarding unilateral constraints, we allow compromises and define potential energy to be solved alongside other potentials. This method handles world box constraints in our simulations.

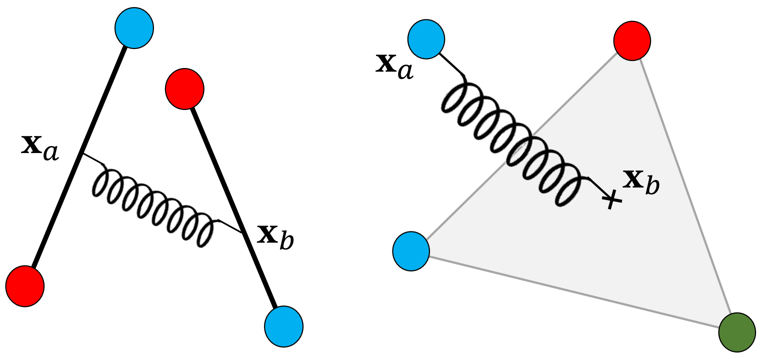


Figure 3.3: Two collision types: (a) edge-edge can have at most two pairs and (b) vertex-face can have at most one pair with the same color, since vertices on the same side of a collision must have different colors.

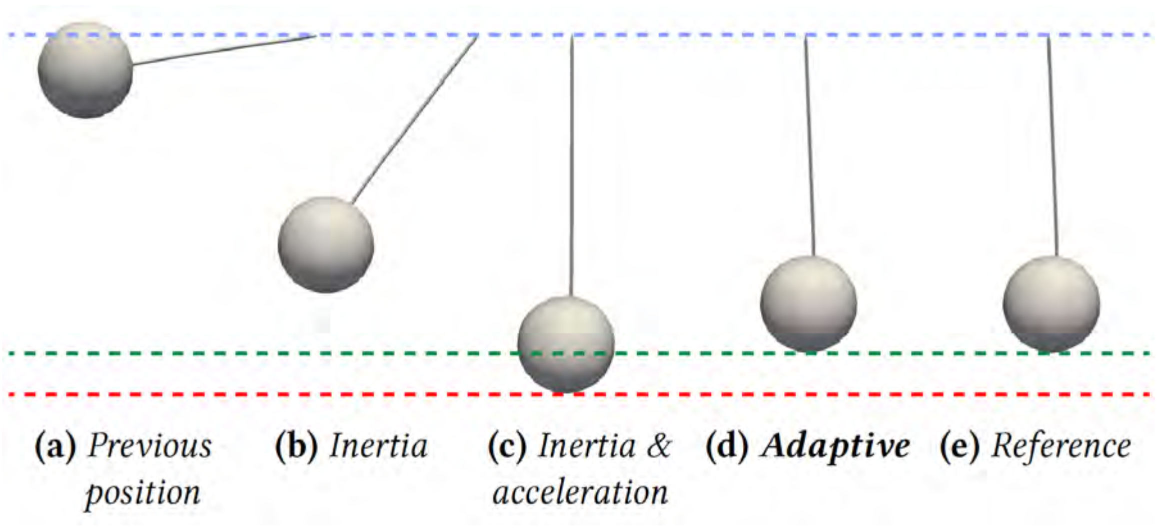


Figure 3.4: Different initialization options for a swinging elastic pendulum dropped from the same height (blue line) simulated with our method using only 20 iterations per frame, showing the same frame of the simulation. Notice that initializing using (a) previous position and (b) inertia fail to properly move under gravity, while (c) inertia and acceleration leads to accessive stretching (red line) when VBD does not run to convergence. (d) Our adaptive solution closely matches (e) the reference generated by fully converged Newton's method (green line).

3.1.5 Collisions

Collisions can be handled by simply introducing a quadratic collision energy per vertex, based on the penetration depth d, such that

$$E_c(\mathbf{x}) = \frac{1}{2} k_c \, d^2 \qquad \text{with} \qquad d = \max\left(0, (\mathbf{x}_b - \mathbf{x}_a) \cdot \hat{\mathbf{n}}\right) , \tag{3.9}$$

where k_c is the collision stiffness parameter, $\mathbf{x}_a$ and $\mathbf{x}_b$ are the two *contact points* on either side of the collision, and $\hat{\mathbf{n}}$ is the contact normal.

There are two collision types for triangle meshes (Figure 3.3):

- Edge-edge collisions use continuous collision detection (CCD). $\mathbf{x}_a$ and $\mathbf{x}_b$ correspond to the intersection points on either edge and the contact normal is the direction between them, i.e., $\hat{\mathbf{n}} = \mathbf{n}/\|\mathbf{n}\|$, where $\mathbf{n} = \mathbf{x}_b - \mathbf{x}_a$.
- Vertex-triangle collisions are detected either by CCD or discrete collision detection (DCD). In this case, $\mathbf{x}_a$ is the colliding vertex and $\mathbf{x}_b$ is the corresponding point on either the collision point (for CCD) or the closest point (for DCD) on the triangle [169]. $\hat{\mathbf{n}}$ is the surface normal at $\mathbf{x}_b$.

In our implementation, we perform a DCD at the beginning of the time step using $\mathbf{x}^t$ to identify vertices that have already penetrated, and the rest of the collisions use CCD. We simplify the computation of the gradient and the Hessian of the collision energy by not differentiating through $\hat{\mathbf{n}}$, i.e., assuming that $\hat{\mathbf{n}}$ is constant.

Performing collision detection at every iteration using CCD can be expensive and can easily become the bottleneck. Therefore, in our implementation, we perform CCD once every n_{col} iterations. While this has the risk of missing some collision events, they are likely to be captured via DCD in the next time step. All detected collisions remain as force elements until the next collision detection. For vertex-triangle collisions detected with DCD, it is important to recompute the closest point ($\mathbf{x}_b$) before computing the gradient and Hessian of E_c.

3.1.6 Friction

To compute friction for collision c, we must consider the relative motion of the contact points defined as

$$\delta\mathbf{x}_c = \left(\mathbf{x}_a - \mathbf{x}_a^t\right) - \left(\mathbf{x}_b - \mathbf{x}_b^t\right) , \tag{3.10}$$

where $\mathbf{x}_a^t$ and $\mathbf{x}_b^t$ are the positions of $\mathbf{x}_a$ and $\mathbf{x}_b$ at the beginning of the time step.

With this $\delta\mathbf{x}_c$, we can use the friction model of *incremental potential contact* (IPC) [2]. First, we project $\delta\mathbf{x}_c$ to the 2D contact tangential space, using a transformation matrix $\mathbf{T}_c \in \mathbb{R}^{3\times 2}$, to evaluate the tangential relative translation $\mathbf{u}_c = \mathbf{T}_c^T \delta\mathbf{x}_c$, where $\mathbf{T}_c = [\hat{\mathbf{t}}\ \hat{\mathbf{b}}]$ is formed by a tangent $\hat{\mathbf{t}}$ and binormal $\hat{\mathbf{b}}$ vectors orthogonal to $\hat{\mathbf{n}}$. The signed magnitude of the collision force applied on vertex i is $\lambda_{c,i} = \frac{\partial E_c}{\partial \mathbf{x}_i} \cdot \hat{\mathbf{n}}$. Note that the sign of $\lambda_{c,i}$ is different for vertices on different sides of the collision. Let μ_c be the friction coefficient. We can then calculate the friction force using

$$\mathbf{f}_{c,i} = -\mu_c\, \lambda_{c,i}\, \frac{\partial \delta\mathbf{x}_c}{\partial \mathbf{x}_i}\, \mathbf{T}_c\, f_1(\|\mathbf{u}_c\|)\, \frac{\mathbf{u}_c}{\|\mathbf{u}_c\|}\ \text{, where} \tag{3.11}$$

$$f_1(u) = \begin{cases} 2\left(\frac{u}{\epsilon_v h}\right) - \left(\frac{u}{\epsilon_v h}\right)^2, & u \in (0, h\epsilon_v) \\ 1, & u \geq h\epsilon_v \end{cases} . \tag{3.12}$$

Here, f_1 serves as a smooth transition function between static and dynamic friction. When the relative velocity exceeds a small threshold ϵ_v, dynamic friction is applied. Conversely, if the relative velocity is below this threshold, static friction is applied, scaling between the range of $[0, 1]$.

In our formulation, we need the Hessian of the friction term, which is the derivative of this function. We approximate the derivative by not differentiating through $\|\mathbf{u}_c\|$ for a more stable friction force formulation [46], such that

$$\frac{\partial \mathbf{f}_{c,i}}{\partial \mathbf{x}_i} \approx -\mu_c\, \lambda_{c,i}\, \frac{\partial \delta\mathbf{x}_c}{\partial \mathbf{x}_i}\, \mathbf{T}_c\, \frac{f_1(\|\mathbf{u}_c\|)}{\|\mathbf{u}_c\|} \mathbf{T}_c^T \left(\frac{\partial \delta\mathbf{x}_c}{\partial \mathbf{x}_i}\right)^T . \tag{3.13}$$

Without this approximation, PSD projection and line search might be needed to ensure stability. Finally, all friction forces $\mathbf{f}_{c,i}$ and their derivatives $\partial\mathbf{f}_{c,i}/\partial\mathbf{x}_i$ are added to $\mathbf{f}_i$ and $\mathbf{H}_i$, respectively.

3.1.7 Initialization

Since our method is an iterative solver, we begin with an *initial guess* for $\mathbf{x}$. The closer this initial guess is to the resulting $\mathbf{x}^{t+1}$, the fewer number of iterations we would need to converge. Typically, if the simulation converges, different initializations should not significantly affect the final results, although they may influence the number of iterations required to achieve convergence.

Providing a good initial guess (one that is close to $\mathbf{x}^{t+1}$) is particularly important for applications with a limited computation budget, e.g., using a fixed number of iterations.

In such applications, the initial guess can strongly impact the remaining residual at the end of the final iteration.

We begin with considering three simple options for initialization:

(a) **Previous position:** $\mathbf{x} = \mathbf{x}^t$

(b) **Inertia:** $\mathbf{x} = \mathbf{x}^t + h\mathbf{v}^t$

(c) **Inertia and acceleration:** $\mathbf{x} = \mathbf{x}^t + h\mathbf{v}^t + h^2\mathbf{a}_{\text{ext}} = \mathbf{y}$

Option (a) struggles with substantially stiff materials. As we adjust each vertex separately, assuming the others remain fixed, our method must lean on the inertia potential's gradient (i.e., $m_i(\mathbf{x}_i - \mathbf{y}_i)/2h^2$) to march toward the local minimum. With stiff materials, this method results in notably slower convergence rates due to the inertia potential being considerably less stiff. Consequently, it encounters challenges in simulating scenarios that resemble free fall, like a swinging elastic pendulum at its maximum height, as shown in Figure 3.4a.

Option (b) allows the system to start with the inertia of the previous step, which helps, but terminating the iterations prior to convergence can again result in local material stiffness overpowering external acceleration, as shown in Figure 3.4b.

Option (c) is similar to the initialization of position-based dynamics, and performs notably better as it effectively preserves inertia and properly reacts to external acceleration. However, with a limited number of iterations, materials behave softer than they should, often resulting in overstretching or collapsing under gravity. An example of this can be seen in Figure 3.4c, where the pendulum extends more than it should. Most notably, it struggles with steady objects at rest in contact, consistently initializing them into a penetrating state, as if they are in free fall. In such cases, the contact forces must entirely undo the position change of initialization during the iterations. This not only creates unnecessary computation, but also places considerable strain on the accuracy of the collision detection and handling methods (including friction). Therefore, properly simulating objects that are stacked on top of each other becomes a major challenge with this initialization option.

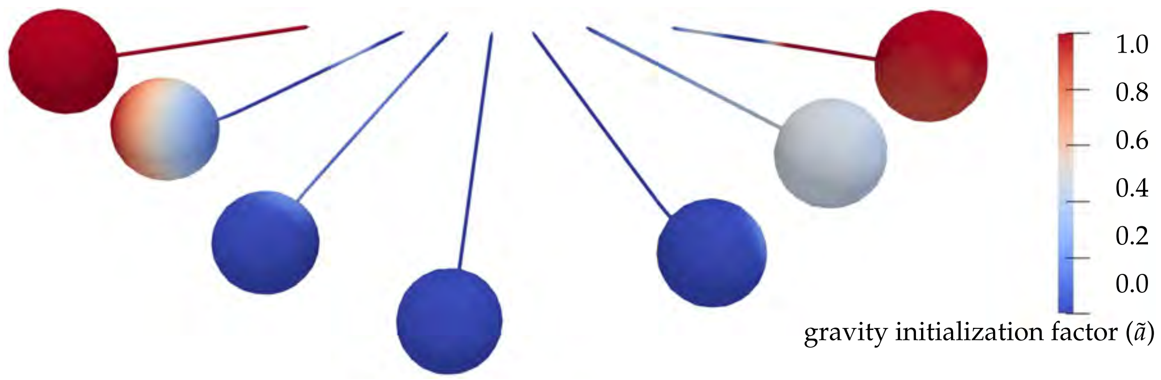


Figure 3.5: The ratio of gravity $\tilde{a}$ used with adaptive initialization during the swinging of an elastic pendulum. The model is a single-piece tetrahedral mesh. It automatically distinguishes vertices in approximate free-fall (red) and those where elasticity counteracts gravity (blue).

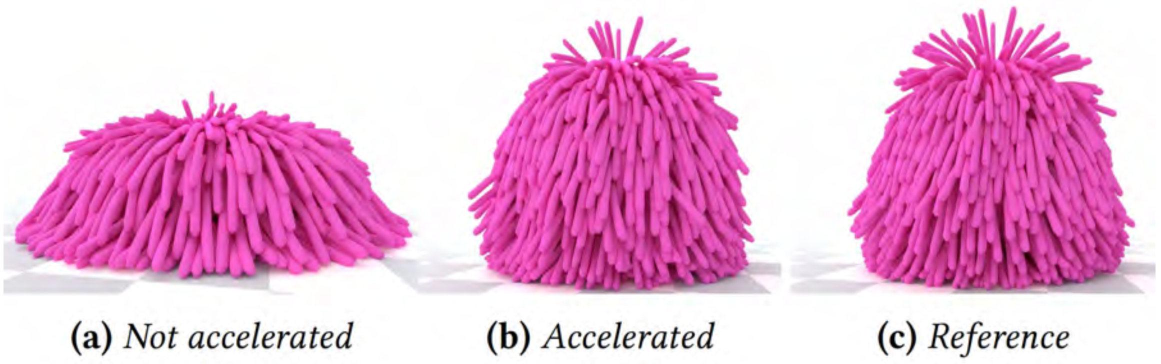


Figure 3.6: Demonstrating the accelerator's impact in a collision-intensive scene: a squishy ball (230K vertices, 700K tetrahedra) dropping and bouncing. Both (a) and (b) use $h = 1/120$ seconds with a constant number of 120 iterations per time step, taking 0.11 seconds of average computation time per frame. (a) Without acceleration 120 iterations appear to be insufficient. (b) Our acceleration scheme ($\rho = 0.95$), skipping colliding vertices, manages to resolve complex collisions, notably enhancing elasticity convergence for much stiffer outcomes, closely matching (c) the reference computed using 2000 iterations.

We propose an adaptive initialization scheme that combines options (b) and (c), taking advantage of the freedom that VBD provides in the choice of initialization. This scheme uses

(d) **Adaptive:** $\mathbf{x} = \mathbf{x}^t + h\mathbf{v}^t + h^2\tilde{\mathbf{a}}$

replacing the external acceleration $\mathbf{a}_{\text{ext}}$ in (c) with an estimated acceleration term $\tilde{\mathbf{a}}$, determined by exploiting the typical similarity between two consecutive time steps. We begin with the acceleration of the previous frame $\mathbf{a}^t = (\mathbf{v}^t - \mathbf{v}^{t-1})/h$ and compute its component a^t_{ext} along the external acceleration direction $\hat{\mathbf{a}}_{\text{ext}} = \mathbf{a}_{\text{ext}}/\|\mathbf{a}_{\text{ext}}\|$, such that $a^t_{\text{ext}} = \mathbf{a}^t \cdot \hat{\mathbf{a}}_{\text{ext}}$. Then, we simply make sure that the estimated acceleration does not exceed the external acceleration, using

$$\tilde{\mathbf{a}} = \tilde{a}\,\mathbf{a}_{\text{ext}}\,, \qquad \text{where} \qquad \tilde{a} = \begin{cases} 1\,, & \text{if } a^t_{\text{ext}} > \|\mathbf{a}_{\text{ext}}\| \\ 0\,, & \text{if } a^t_{\text{ext}} < 0 \\ a^t_{\text{ext}}/\|\mathbf{a}_{\text{ext}}\|\,, & \text{otherwise.} \end{cases} \tag{3.14}$$

This adaptive approach includes external acceleration in the initialization when the motion of a vertex resembles free fall. When an object is stationary, however, as in rest-in-contact, it maintains the previous position in the initialization, preventing undesired penetrations before the first iteration. It also successfully avoids excessive stretching, as can be seen in Figure 3.4d. Visualizations of different $\tilde{a}$ values in this simulation are shown in Figure 3.5. In short, our adaptive initialization is a simple but effective strategy and it is possible because VBD does not dictate a particular initialization (unlike XPBD, for example).

3.1.8 Accelerated Iterations

We use the Chebyshev semi-iterative approach [42] to improve the convergence of our method, though other momentum-based acceleration techniques, such as the Nesterov's method [170] can be applied as well. The Chebyshev method iteratively computes an acceleration ratio based on the approximation of the system's spectral radius. Instead of directly using the output vertex positions of Gauss-Seidel $\bar{\mathbf{x}}^{(n)}$ after iteration n, it recomputes the positions at the end of the iteration using

$$\mathbf{x}^{(n)} = \omega_n(\bar{\mathbf{x}}^{(n)} - \mathbf{x}^{(n-2)}) + \mathbf{x}^{(n-2)}\,, \tag{3.15}$$

where ω_n is the acceleration ratio that changes at each iteration as

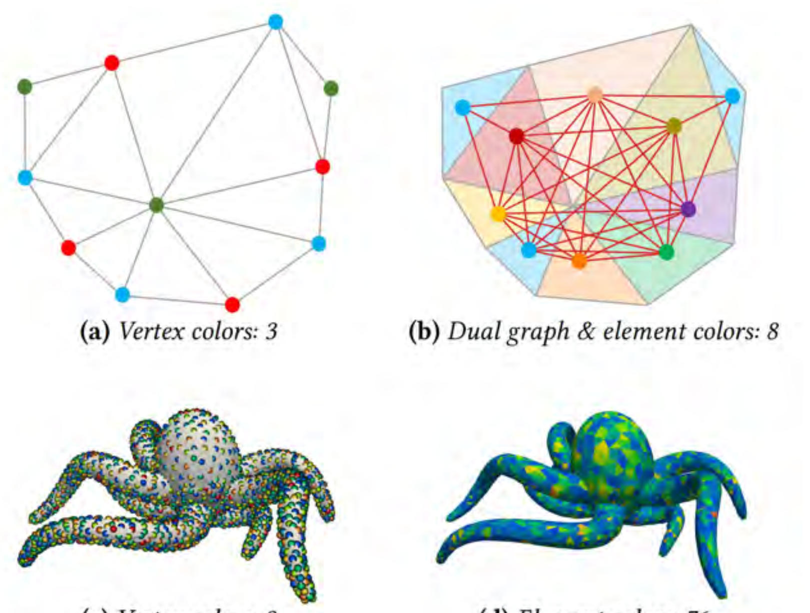
$$\omega_n = \frac{4}{4 - \rho^2\omega_{n-1}} \qquad \text{with} \qquad \omega_1 = 1 \qquad \text{and} \qquad \omega_2 = \frac{2}{2-\rho^2}\,, \tag{3.16}$$

where $\rho \in (0, 1)$ is the estimated spectral radius, which can be set manually, or automatically tuned using the technique introduced in [44]. Note that this position recomputation procedure is performed globally after each Gauss-Seidel iteration is completed, not after each local solver step.

This accelerator was originally developed for solving linear systems, assuming the energy to be smooth and (nearly) quadratic. Elastic energy generally fulfills these criteria. However, collision energy tends to be discontinuous and highly stiff, making the use of an accelerator in collision-intensive scenes prone to overshot and compromise the system's stability. To address this, we propose a simple yet highly effective solution for accelerating scenes with collisions: skipping the accelerations for actively colliding vertices. Note that the acceleration must be a continuous process. If a vertex is detected colliding at a certain iteration, we will skip the acceleration for it in all the following iterations in the same step, regardless of whether the collision has been resolved. This approach has minimal impact on the convergence of elasticity, since typically only a small fraction of vertices are in collision. Also, for those colliding vertices, the elasticity is usually overpowered by the collision forces. Thus, this solution maintains the stability of the system, while effectively accelerating the convergence of elasticity, as shown in Figure 3.6.

3.1.9 Parallelization

Gauss-Seidel-type iterative methods are often parallelized using graph coloring by determining groups (i.e. colors) that can be handled in parallel without impacting the sequential nature of the Gauss-Seidel loop. Obviously, the same can be applied to VBD by simply coloring vertices such that no force element uses multiple vertices of the same color.



(a) *Vertex colors: 3* **(b)** *Dual graph & element colors: 8*

(c) *Vertex colors: 8* **(d)** *Element colors: 76*

Figure 3.7: Coloring vertices versus elements: (a) vertex coloring needs 3 colors for 10 vertices while (b) element coloring (i.e., coloring the vertices of the dual graph) needs 8 colors for 10 triangles in this example. The difference is more pronounced for tet-meshes: (c) our vertex coloring uses only 8 colors for 3,891 vertices while (d) our element coloring implementation needs 76 colors for 14,802 tetrahedra in this example.

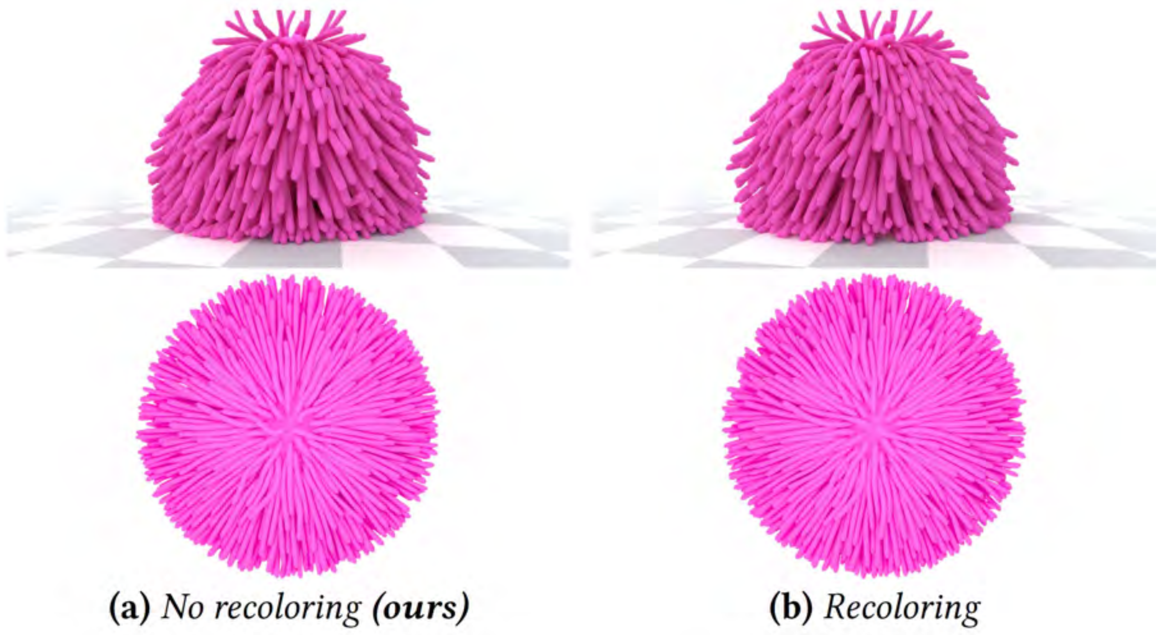


Figure 3.8: Handling collisions using (a) our scheme without recoloring and (b) recoloring to achieve perfect Gauss-Seidel iterations, both simulated using friction forces and accelerated iterations with $\rho = 0.95$. Notice that the results are highly similar, though not identical.

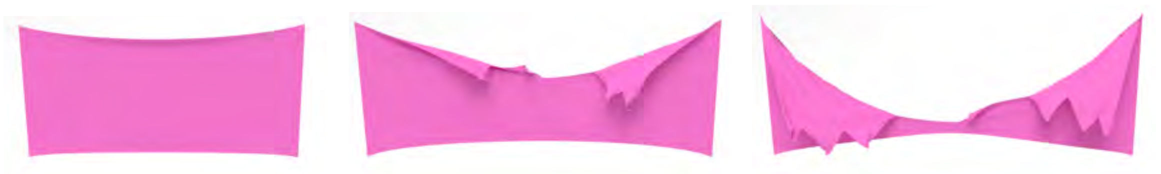


Figure 3.9: Tearing a piece of cloth with 2500 vertices and 4800 triangles.

The advantage of VBD here is that, because it colors vertices, it typically results in much fewer colors as compared to techniques that color constraints/force elements, such as PBD. This is because coloring these elements is equivalent to coloring the nodes of the dual graph, which not only has more nodes, but, more importantly, also has a lot more connections per vertex in general. Examples of this are shown in Figure 3.7. Since different colors must be handled sequentially, fewer colors means better parallelism.

When all force elements are known ahead of the simulation, graph coloring can be performed as a preprocess. However, dynamically generated constraints/force elements, such as ones due to collisions, cannot be known ahead of time, requiring dynamic recoloring.

In our implementation for elastic body dynamics, we avoid the cost of recoloring by precomputing coloring only for material forces, ignoring dynamically generated force elements due to collisions. This means that these collision forces may use multiple vertices of the same color. Therefore, we cannot simply run a parallel loop over all vertices of the same color and update them, because handling a vertex with a dynamically generated force element may run into race conditions (with partially updated vertex positions) when accessing other vertices of the same color.

We resolve this by having an auxiliary vertex position buffer ($\mathbf{x}^{\text{new}}$) that stores the updated vertex position. When executing each local VBD position update, we write the updated vertex positions to the auxiliary buffer, instead of directly overwriting the original vertex position buffer. Then, we copy the updated positions from the auxiliary buffer to the original vertex position buffer after each color pass. This prevents the race conditions that arise from simultaneous read and write operations on vertex positions.

With this process, dynamically generated force elements using multiple vertices of the same color result in (partially) Jacobi-style iterations for those vertices, because they rely on the positions from the previous iteration of those vertices. For vertices with different colors, it is equivalent to Gauss-Seidel iterations, considering the updated positions of the vertices with different colors. Note that our algorithm does not explicitly switch between Jacobi and Gauss-Seidel iterations, but the resulting iteration we describe above corresponds to either (partially) Jacobi or Gauss-seidel, depending on the colors of the colliding vertices.

One might expect this solution to negatively impact the convergence of VBD. Fortunately, however, such Jacobi-style information exchanges are relatively rare. This is because, as shown in Figure 3.3, with vertex-face collisions at most one pair and with edge-edge collisions at most two pairs of vertices can have the same color. Also, vertices with the same colors must be located on different sides of the collision; therefore, their elastic energies are usually decoupled. This ensures that the majority of the information exchange follows the Gauss-Seidel order and thereby the impact of our solution on convergence is minimal. Figure 3.8 presents an example with a large number of collisions, comparing our solution of skipping recoloring to proper Gauss-Seidel iterations with recoloring, showing that the differences are minor.

Our solution also works with other types of topological changes, such as tearing and fracturing. Deleting force elements does not require any changes to vertex coloring. When an object is split by duplicating vertices, as in the case of tearing a piece of cloth along some edges (see Figure 3.9), duplicated vertices can safely inherit the colors of their original vertices.

3.2 GPU Implementation

In this section, we describe a GPU implementation specifically designed to leverage the inherent parallelization mechanism of modern GPUs, which consists of two hierarchical levels: block-level and thread-level parallelism. Block-level parallelism facilitates large-scale parallel operations, assuming that each block operates independently. On the other hand, thread-level parallelism provides finer, single-instruction-multiple-thread (SIMT) style parallelism, allowing for inter-thread communication and synchronization within the same block.

Reflecting on VBD, we observed that it naturally aligns with this hierarchical architecture. We have thousands of vertices within a single color category that operate independently, and each vertex is associated with multiple force elements, which can be processed concurrently. algorithm 1 shows the pseudocode of our implementation. It uses block-level parallelism for processing each vertex. The threads within each block are used for computing the forces and Hessians, storing them in local shared memory, and computing the sums via reduction. We use a fixed number of threads for each block. When the total number of adjacent force elements exceeds the number of threads for each

Algorithm 1: VBD simulation for one time step.

```
Input: $\mathbf{x}^t$: the positions of the previous step; $\mathbf{v}^t$: the velocities of the previous step;
       $\mathbf{a}_{\text{ext}}$: the external acceleration
Output: This step's position $\mathbf{x}^{t+1}$ and velocity $\mathbf{v}^{t+1}$.
1  $\mathbf{y} \leftarrow \mathbf{x}^t + h\mathbf{v}^t + h^2\mathbf{a}_{\text{ext}}$
2  Initial DCD using $\mathbf{x}^t$
3  $\mathbf{x} \leftarrow$ initial guess with adaptive initialization
4  for each iteration $n \leq n_{\max}$ do
5      if $n$ mod $n_{\text{col}} = 1$ then CCD using $\mathbf{x}$
6      for each color $c$ do
           // Block-level parallelization
7          parallel for each vertex $i$ in color $c$ do
               // Thread-level parallelization
8              parallel for each $j \in \mathcal{F}_i$ do
                   // Variables in shared memory
9                  $\mathbf{f}_{i,j} = -\frac{\partial E_j}{\partial \mathbf{x}_i}$
10                 $\mathbf{H}_{i,j} = \frac{\partial^2 E_j}{\partial \mathbf{x}_i \partial \mathbf{x}_i}$
11             end
               // Local reduction sums
12             $\mathbf{f}_i = \sum_{j \in \mathcal{F}_i} \mathbf{f}_{i,j}$
13             $\mathbf{H}_i = \sum_{j \in \mathcal{F}_i} \mathbf{H}_{i,j}$
14             $\Delta\mathbf{x}_i \leftarrow \mathbf{H}_i^{-1}\mathbf{f}_i$
15             $\Delta\mathbf{x}_i \leftarrow$ optional line search from $\mathbf{x}_i + \Delta\mathbf{x}_i$ to $\mathbf{x}_i$
16             $\mathbf{x}_i^{\text{new}} \leftarrow \mathbf{x}_i + \Delta\mathbf{x}_i$
17         end
           // Copy updated positions back to the vertex buffer
18         parallel for each vertex $i$ in color $c$ do
19             $\mathbf{x}_i = \mathbf{x}_i^{\text{new}}$
20         end
21     end
       // Optional: accelerated iteration
22     parallel for each vertex $i$ do
23         Update $\mathbf{x}_i$ using Equation 3.15.
24     end
25 end
26 $\mathbf{v} = (\mathbf{x} - \mathbf{x}^t)/h$
27 return $\mathbf{x}$, $\mathbf{v}$
```

block, individual threads will loop over their assigned elements. During this process, they calculate the forces and Hessians for these force elements and then sum them to their assigned shared memory. At last, the forces and Hessians of all the threads are combined using a local reduction sum method (lines 12,13).

In our experiments, we observed nearly *an order of magnitude performance improvement*, as compared to processing each vertex with a single thread. The primary advantage lies in the optimization of the memory access pattern, a common bottleneck in GPU programs. This implementation reduces memory divergence within blocks. Because the neighboring force elements of a vertex often share multiple vertices, the threads within the same block can share a global memory access to those shared vertices. Furthermore, this strategy improved the parallelism of the algorithm by allowing parallel evaluation of the force and Hessian of adjacent force elements, which are then written to the significantly faster shared memory. This bypasses the slower global memory and enables the parallel aggregation of force and Hessian values, enhancing the overall efficiency of the process.

3.3 VBD for Other Simulation Systems

We have described our method in Section 3.1 in the context of elastic body dynamics. Yet, VBD is not limited to such simulations and can be used to solve various optimization problems. Here, we consider some other example simulation systems and briefly discuss how our method can be applied. This is not intended as an exhaustive list but merely as examples that could guide the reader to discern how their specific simulation problem could utilize VBD.

3.3.1 Particle-Based Simulations

Particle-based simulations can easily use VBD by simply replacing the vertices in our description above with particles. Since VBD needs the Hessian of the force element energies, implementations would require computing the derivatives of all forces acting on a particle wrt. its position.

Parallelizing particle-based simulations also involves additional considerations. Mass-spring type simulations, such as peridynamics [171], can use our parallelization approach with vertex coloring. However, simulations involving disjoint or loosely-joined particles, such as particle-based fluid simulation [172, 173, 174], would not only require recoloring at each time step but also using a conservative neighborhood definition (including position change within a time step) for coloring, since position updates can alter the set of particles that interact with each particle.

Figure 3.10 shows a simple example where 20 particles, including one that is $1000\times$ heavier, are connected with springs of two different stiffness, simulated using VBD.

3.3.2 Rigid Body Simulation

For handling rigid body simulations with VBD, we can replace each vertex in our formulation with an entire rigid body, using the variational formulation of rigid body dynamics [68]. Unlike a vertex that has only 3 DoF, a rigid body also has rotational DoF, resulting in 6 DoF. Therefore, in our local system, we must solve a larger problem, where $\mathbf{x}_i \in \mathbb{R}^6$ and $\mathbf{H}_i \in \mathbb{R}^{6\times 6}$, including Hessians of all force elements wrt. all 6 DoF of $\mathbf{x}_i$. Note that, in this case, these force elements are not internal material forces, but external forces acting on the rigid body, due to collisions or other constraints.

Other than this additional complexity, we can follow the same procedure with VBD. Parallelization with coloring depends on the nature of the rigid body simulation. For example, pre-coloring, as we used in our examples for elastic bodies might work for problems like a rigid body chain. For disjoint rigid bodies interacting through collisions only, dynamic recoloring might be needed.

Articulated rigid bodies can be handled by defining joint constraints with an elastic potential. Infinitely stiff constraints are also possible, but VBD cannot guarantee that they will be satisfied using a fixed number of iterations. Another alternative is hard constraints can be introduced by reducing the total DoF in the system and replacing the vertices in our formulation with an articulated rigid body, having more than 6 DoF. Obviously, this would lead to an even larger local system, requiring modifications to the variational formulation.

Example rigid body simulations are shown in Figure 3.11 and Figure 3.12, simulated using VBD, as described above.

Note that since our collision formulation is based on penetration potential, it corresponds to penalty forces. We leave the exploration of handling impulse-based collisions [175] with VBD to future work.

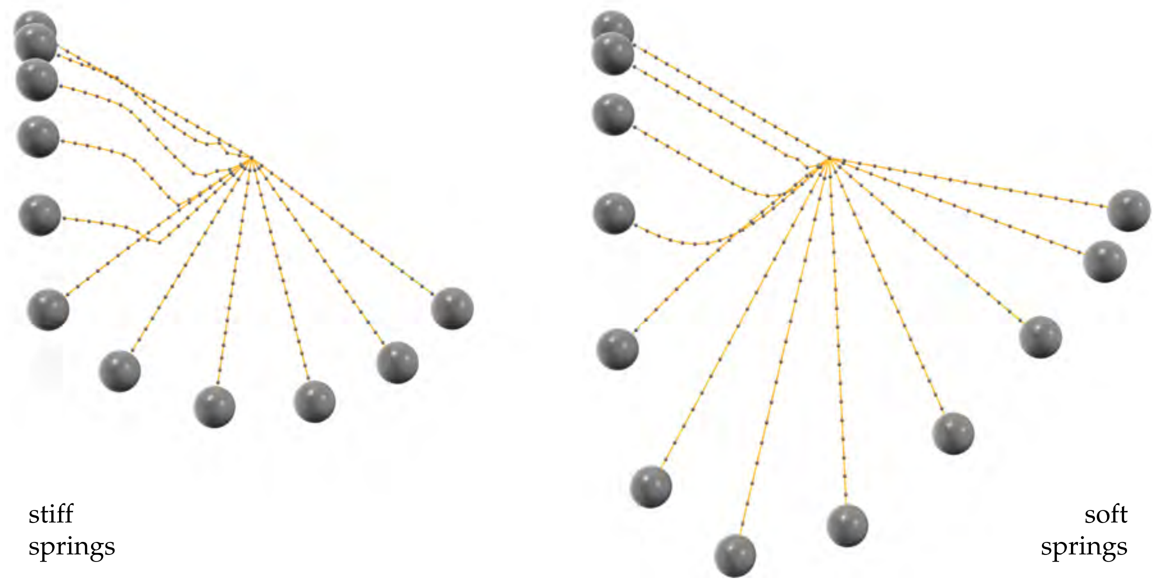


Figure 3.10: Twenty particles attached with springs, forming a swinging chain, simulated using VBD with a $S = 1$ substep and 100 iterations per step. The particle on one end of the chain is fixed and the particle on the other end has 1000× more mass than the others. (Left) using sufficiently stiff springs, they expand no more than 0.7% of their rest lengths, despite the substantial mass difference. (Right) using 100× less stiff springs, the chain undergoes a visible expansion as it swings.

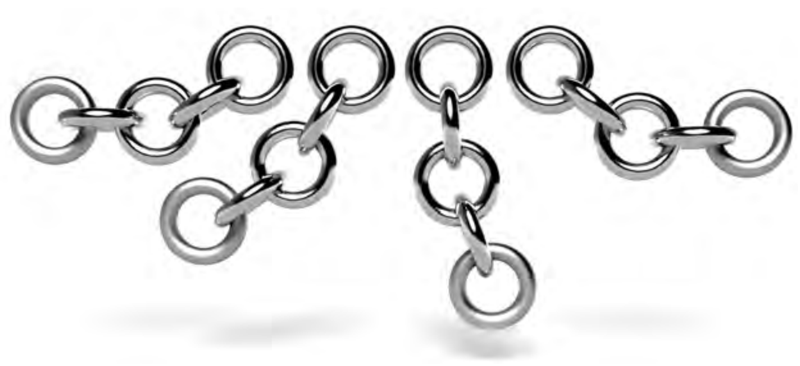


Figure 3.11: Five rigid bodies, each with 6 DoF, forming a chain through collisions, simulated using our VBD formulation for rigid body dynamics.

3.3.3 Unified Simulations

Unified simulation systems are useful for handling scenarios that involve different material types. Typical unified simulation systems use a fundamental building block, such as a particle, to represent all supported materials [176, 177, 178, 179, 180]. We can form a unified simulation system using VBD without representing all materials using the same building block. For any simulation system described above, we can combine it with another, provided that we can define the information exchange as an energy potential. For example, when the interactions take place as collisions, we can easily join rigid body simulations with elastic bodies or particles via the collision potential. Joint constraints with elastic potential would be another easy way to combine different simulation systems. The advantage here is that a large rigid body, for example, can be represented as a single object with just 6 DoF, as opposed to using multiple building blocks that are constrained to move as a rigid construction. This way, a heterogeneous collection of representations can be joined within the same integrator using VBD.

On the other hand, this form of defining a unified simulation system may be challenging for other types of information exchange, such as evaluating buoyancy. Exploring such problems would be another interesting direction for future research.

3.4 Results

We evaluate our method with elastic body dynamics qualitatively with various tests and quantitatively with comparisons to alternative methods. We use Neo-Hookean [181] materials (without the logarithmic term) for our volumetric objects, StVK [182] for clothes, and linear springs for elastic rods.

We use a fixed frame time of $1/60$ seconds and a fixed iteration count $n_{\max}$, instead of estimating convergence after every iteration. Each frame is computed using S substeps, such that $h = 1/(60S)$ seconds. Using smaller time steps increases accuracy and reduces numerical damping with any implicit Euler method. With VBD, a smaller time step only requires fewer iterations per step for similar visual quality, but it can also achieve a smaller residual with the same number of total iterations per frame (i.e., $Sn_{\max}$). The number of threads per block is set to 16 for all the experiments.

We use no line search in our tests, as none of our tests required it for stability, and running line search can result in a noticeable drop in performance. In our experiments, we apply CCD only in the first iteration (i.e., $n_{\text{col}} = n_{\text{max}}$), unless otherwise specified.

In our implementation, we handle collisions on the CPU using Intel's Embree library [183]. The two phases with parallel loops are implemented on the GPU using CUDA. All timing results are generated on a computer with an AMD Ryzen 5950X CPU, 64GB DDR3 RAM, and an NVIDIA RTX 4090 GPU. The runtime statistics and parameters of all our experiments can be found in Table 3.1.

3.4.1 Large-Scale Tests

We present two large-scale test, showcasing our method's performance, scalability, and stability in scenarios involving a large number of complex collisions, including stacking and rest in contact.

The first one has 216 squishy balls with tentacles, totaling 48 million vertices and 151 million tetrahedra acting as force elements, dropped into a Utah teapot. Intermediate frames of this simulation are shown in Figure 3.13.

The second one includes more than 10 thousand deformable objects, totaling over 36 million vertices and 124 million tetrahedra, dropped into a box and piled on a platform, which is then suddenly removed, making the pile collectively fall onto the ground. The intermediate frames are shown in Figure 3.14.

As can be seen in our supplemental video, both of these simulations exhibit stable motion, quickly forming static piles, while maintaining rest-in-contact behavior with over 1 million active collisions. VBD's parallelism and fast convergence resulted in an average computation time of 40 and 25 seconds per frame in these simulations, respectively.

3.4.2 Unit Tests

The convergence rate of VBD depends on the stiffness of the simulated system. This is demonstrated in Figure 3.15, comparing VBD with different numbers of iterations per frame to the converged solution computed using Newton's method. As expected, VBD converges slower for stiffer materials, which is common for descent-based solvers. In this example, 15 iterations are more than sufficient for the softest material, while stiffer ones require more. As can be seen in our supplemental video, even though VBD can qualitatively imitate the behavior of stiff materials with a relatively small number of iterations, the motion can quickly diverge from the converged solution, due to the remaining residual, unless a sufficient number of iterations are used.

We present our tests with different friction coefficients μ_c for friction forces in Figure 3.16. Notice that, μ_c impacts the motion, as expected, and with sufficiently high μ_c, we can properly preserve the position on an incline and form taller piles.

3.4.3 Stress Tests

We present a challenging frictional contact case in Figure 3.1, twisting two thin beams together. This example includes extreme deformations, generating strong material forces that compete with self-collisions and collisions between the two beams. It is simulated with collision detection occurring once every 5 iterations (i.e., $n_{\text{col}} = 5$). Notice that VBD can properly handle such strong deformations with frictional contact.

We show two simulations in Figure 3.2 for stress-testing the stability of VBD under extreme deformations. The first one shows an armadillo model that is perfectly flattened and the second one is a Utah teapot model with all vertices randomly scrambled and placed on the surface of a sphere. Even though the simulations begin with these extremely unstable energy configurations, VBD quickly recovers these models without performing a line search and using 100 iterations per frame.

Another stress test is shown in Figure 3.17. In this case, 10 vertices of a Stanford bunny model are first slowly pulled away, generating a state with considerable potential energy, and then suddenly released (right after Figure 3.17b), causing severe deformations. Once again, VBD successfully handles this challenging simulation case, involving self-collisions with high-velocity impacts, using only $n_{\max} = 10$ iterations per step and $S = 5$ per frame. Figure 3.18 presents a stability test under large residuals by using only a single iteration per frame (i.e., $S = 1$ and $n_{\max} = 1$). Notice that our method produces stable deformations with extreme stretching, even though the simulation lacks a sufficient iteration count to properly reduce the residual at each frame.

3.4.4 Convergence Rate

To evaluate the convergence rate of VBD, we present a simple test shown in Figure 3.19, where an armadillo model that was previously stretched is suddenly released. Here, we calculate the relative loss after each iteration and compare it to alternative solvers.

The first alternative is preconditioned gradient descent (GD) [44] implemented on GPU within the same framework as ours. GD requires a form of line search for stability, which is implemented as testing the variational energy after every 8 iterations and, when needed, reducing the optimization step size and backtracking (following the implementation of [44]). We also include a version of GD that is accelerated using the Chebyshev semi-iterative approach [42], as our method. The iterations of GD are about 30% faster than ours without line search. However, it necessitates line search for stability. This makes it about 10% slower than our VBD, which does not need line search. Furthermore, its convergence rate per iteration is considerably slower, as it corresponds to Jacobi iterations. In this example, when combined with acceleration, GD performs similar to our method without acceleration but lags significantly behind our method with acceleration.

We also compare to a version of our method that uses Jacobi iterations, called *Block Jacobi*, implemented by computing the position change for all vertices in parallel and then applying the position update simultaneously to all vertices at the end of each iteration. We incorporate the same line search scheme as GD for Block Jacobi, as it is necessary for stability. Block Jacobi outperforms GD, as it uses vertex blocks for computation, which corresponds to using a diagonal Hessian block as a precondition, as opposed to just a diagonal line that GD uses. Without line search, it achieves 20% faster iteration times than our VBD, due to its perfect vertex-level parallelization (without any coloring). However, combined with line search, its iterations are about 17% slower than VBD. More importantly, because it uses Jacobi iterations, its convergence is hindered, as compared to our Gauss-Seidel iterations.

Furthermore, we compare the convergence of our method to two implementations of Newton's method: first using a direct Cholesky (LDLT) factorization solver provided by Intel's MKL library [184] running on the CPU, and the second using a GPU-based conjugate gradient (CG) method. To ensure convergence, we do a PSD projection for each tet's Hessian of elasticity. Newton's method uses a line search for each iteration. Since its iterations are slow, the computational overhead of line search is negligible. Though the convergence of Newton's method per iteration is far superior to all others, because of its expensive computation time per iteration, in these examples it lags far behind. Nonetheless, for relatively tame experiments with less stretching and motion, and especially for highly stiff and high-resolution simulations that are much more expensive to simulate, we would expect Newton's method to eventually overtake all alternatives beyond a certain level of convergence.

Finally, we compare our method to a quasi-Newton approach using Laplacian preconditioning [27]. We utilize a GPU-based conjugate gradient solver, similar to the one employed in our GPU-CG Newton's method's implementation. Laplacian preconditioning accelerates each linear solve, as it eliminates the need for PSD projection and involves solving a smaller system. Practically, this method demonstrates faster convergence than Newton's method, particularly in the initial stages of the optimization process. Nevertheless, it still lags behind both the accelerated and non-accelerated versions of our VBD.

A part of the performance advantages of our VBD method presented above is related to its efficiency in parallel execution on the GPU. To demonstrate its convergence in the absence of parallel computation, we include a comparison using single-threaded CPU implementations of our VBD and Newton's method with Cholesky factorization and CG. Figure 3.20 shows the convergence results for the same experiment in the bottom row of Figure 3.19. In these tests, our method initially demonstrates faster convergence than both versions of Newton's method. Over time, the CG-based Newton's method catches up to our VBD without Chebyshev acceleration. However, VBD with Chebyshev acceleration maintains a significant performance lead over the others. This experiment shows that the performance advantages of our method are not only due to its GPU parallelism.

3.4.5 Comparisons to XPBD

Our method has an entirely different formulation than XPBD, but there are some strong similarities, as both methods operate with position updates using Gauss-Seidel iterations. Here we provide two direct comparisons to highlight some important differences.

XPBD replaces the Hessian matrix and uses only the Hessian of inertia potential. This omission is justified by using a small time step, because the significance of the inertia potential increases quadratically as the time step decreases. Nonetheless, with complex examples, the impact of this approximation can be severe, even with small time step. This is demonstrated in Figure 3.21 with a challenging collision-rich scenario involving a squishy ball with tentacles dropped to the ground. Comparing XPBD with 120 iterations per step (Figure 3.21a) to our method with the same number of iterations (Figure 3.21d), we can see that our method not only achieves a more stable animation, it also performs faster because of its improved parallelism, as compared to XPBD. Reducing the time step helps XPBD even when using a similar total number of iterations per frame (Figure 3.21b). However, simply reducing the time step is not sufficient in this case, as XPBD also needs more frequent collision detection (Figure 3.21c). Using collision detection with the same frequency as ours while taking small time step (Figure 3.21b) leads to collisions that are detected too late and cause stability issues in this case. This is not only because the collisions that are detected too late are deeper, but also because smaller time step lead to higher vertex velocities when resolving stiff collisions.

One of the fundamental challenges of XPBD is handling high mass ratios. This is demonstrated with a simple example in Figure 3.22, where a large and heavy elastic cube is dropped onto a smaller and much lighter cube, with a mass ratio of 1:2000. In this example, XPBD's collision constraints, even with infinite stiffness, cannot overcome the mass ratio and the smaller cube is entirely crushed upon contact. This is because of the dual formulation of XPBD [46]. Our method, on the other hand, has no such difficulties with handling high mass ratios.

3.5 Discussion

We derive our method as a block coordinate descent method for variational time integrators, which offers optimization techniques like PSD projection and line search. The fact that we do not require those techniques to guarantee stability actually makes our method a more general solver of nonlinear equations. When line search is not used, our method can effectively manage *non-conservative forces*, such as friction, the same as how it handles conservative forces. In other words, our method allows for a seamless transition between block coordinate descent and block Gauss-Sediel [185, 186]. While we do not practically utilize these optimization techniques derived from the descent view, they remain available options for users.

VBD is a descent-based method that operates through local iterations. Therefore, it may not be a good solution for problems that would benefit from a global treatment.

The speed of information travel with VBD depends on the connections of vertices and the number of iterations used. A perturbation applied on a vertex can impact other vertices of a connected chain through force elements at most as far as the number of colors within a single iteration. Therefore, VBD is not ideal for high-resolution stiff systems, as it may require too many iterations for a perturbation of a vertex to travel across the system. In such cases, a global solution using Newton's method may prove to be more effective.

Our collision formulation for VBD is based on penetration potential. Therefore, it cannot guarantee penetration-free results. In fact, penetrations are almost never completely resolved, as some amount of penetration is needed to maintain some collision force. In addition, defining a similar collision energy for codimensional objects, particularly for self-collisions, can be a challenge.

VBD is a primal solver [46], so it can easily handle high mass ratios (see Figure 3.5, 3.22, and 3.10), but it struggles with high stiffness ratios. This is shown in Figure 3.23 using a stiffness ratio of 1:10000, where VBD has poor convergence behavior.

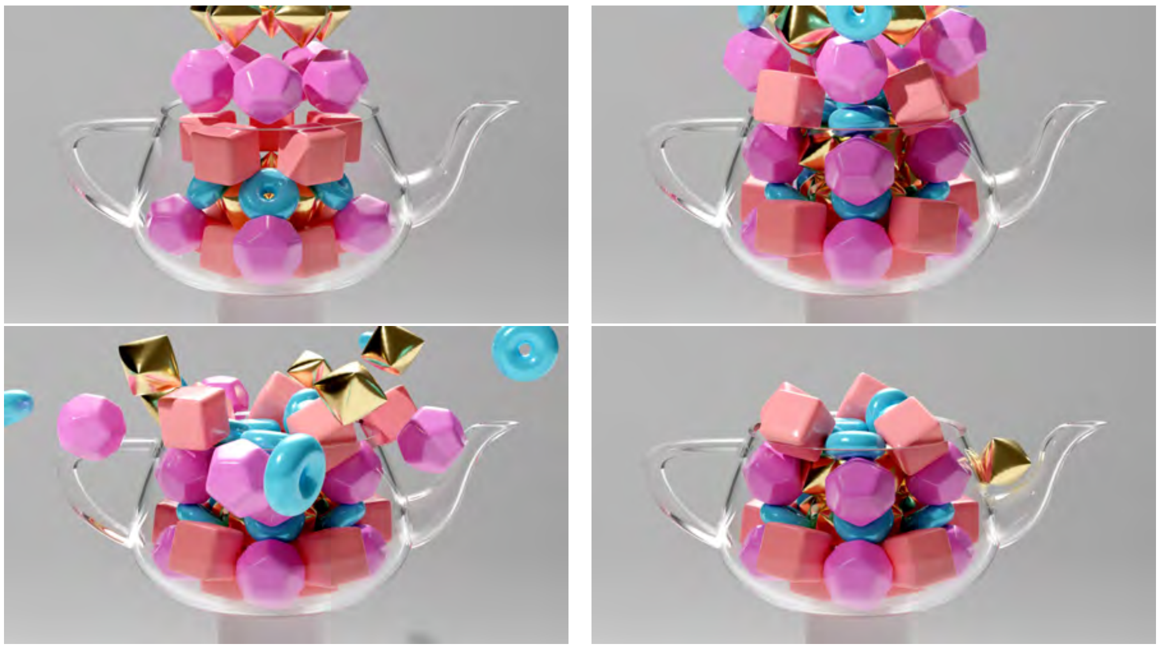


Figure 3.12: Dropping 60 rigid bodies into a Utah teapot, showcasing collisions and frictional contact. Remarkably, one rigid body stays on the spout due to friction.

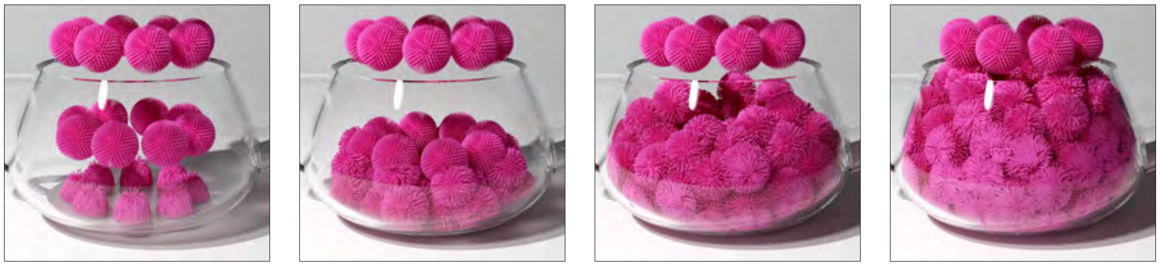


Figure 3.13: Simulation of 216 squishy balls with tentacles, a total of 48 million vertices and 151 million tetrahedra, dropped into a Utah teapot, forming a stable pile with active frictional contacts. The average and maximum computation time per time step is 3.6 and 3.9 seconds, respectively, using $S = 4$ substeps per frame and $n_{\max} = 40$ iterations per step.

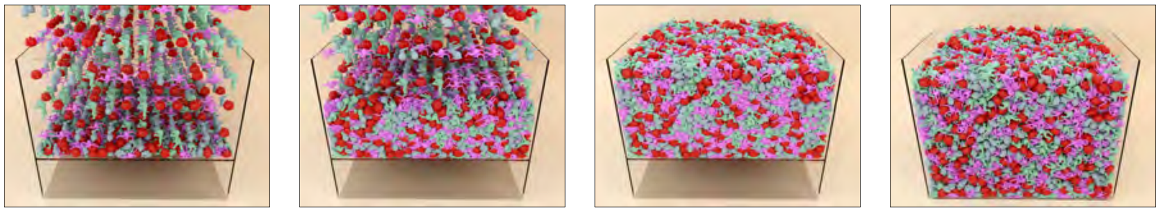


Figure 3.14: Simulation of 10,368 deformable objects, totaling over 36 million vertices and 124 million tetrahedra, dropped onto a platform inside a box container. Then, the platform is suddenly removed and the objects collectively fall onto the ground, forming stable piles both before and after the platform is removed. The average and maximum computation times per time step are 4.2 and 4.7 seconds, respectively, using $S = 2$ substeps and $n_{\max} = 60$ iterations per step.

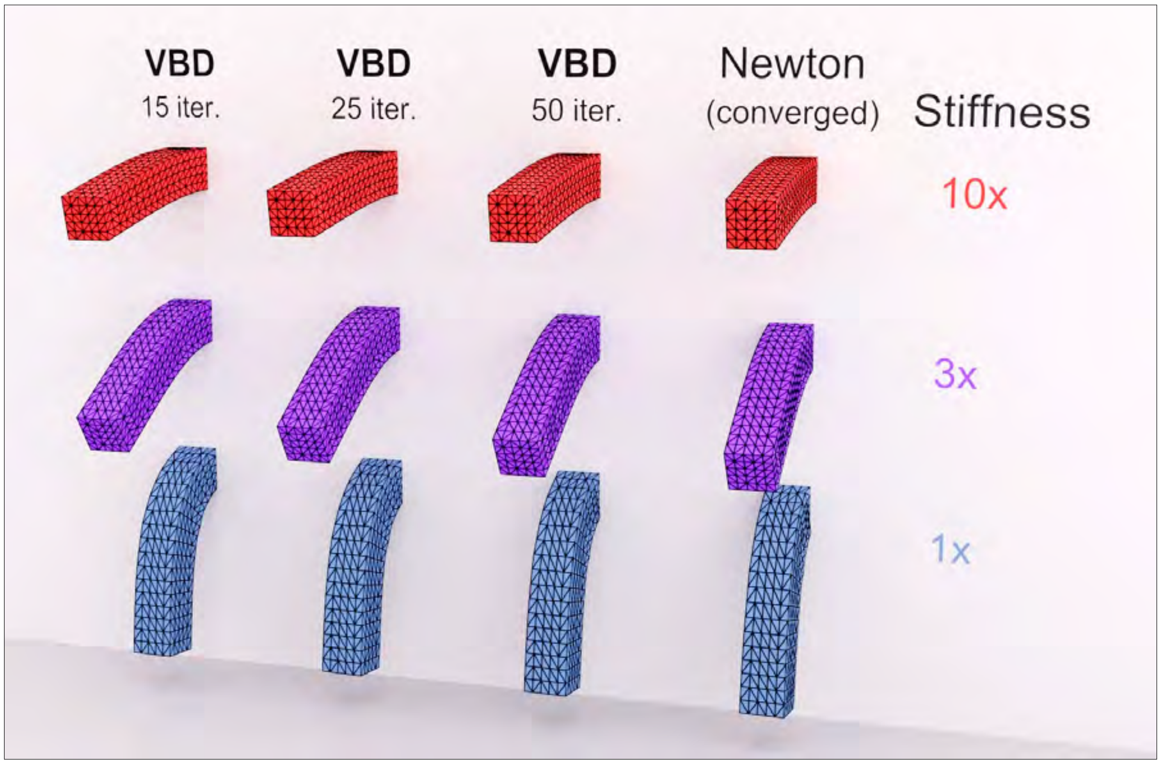


Figure 3.15: Visual convergence with different numbers of iterations per frame for different material stiffness (with accelerated iterations using $\rho = 0.75, 0.86, 0.93$ top to bottom), simulating a beam with 463 vertices and 1.5 thousand tetrahedra.

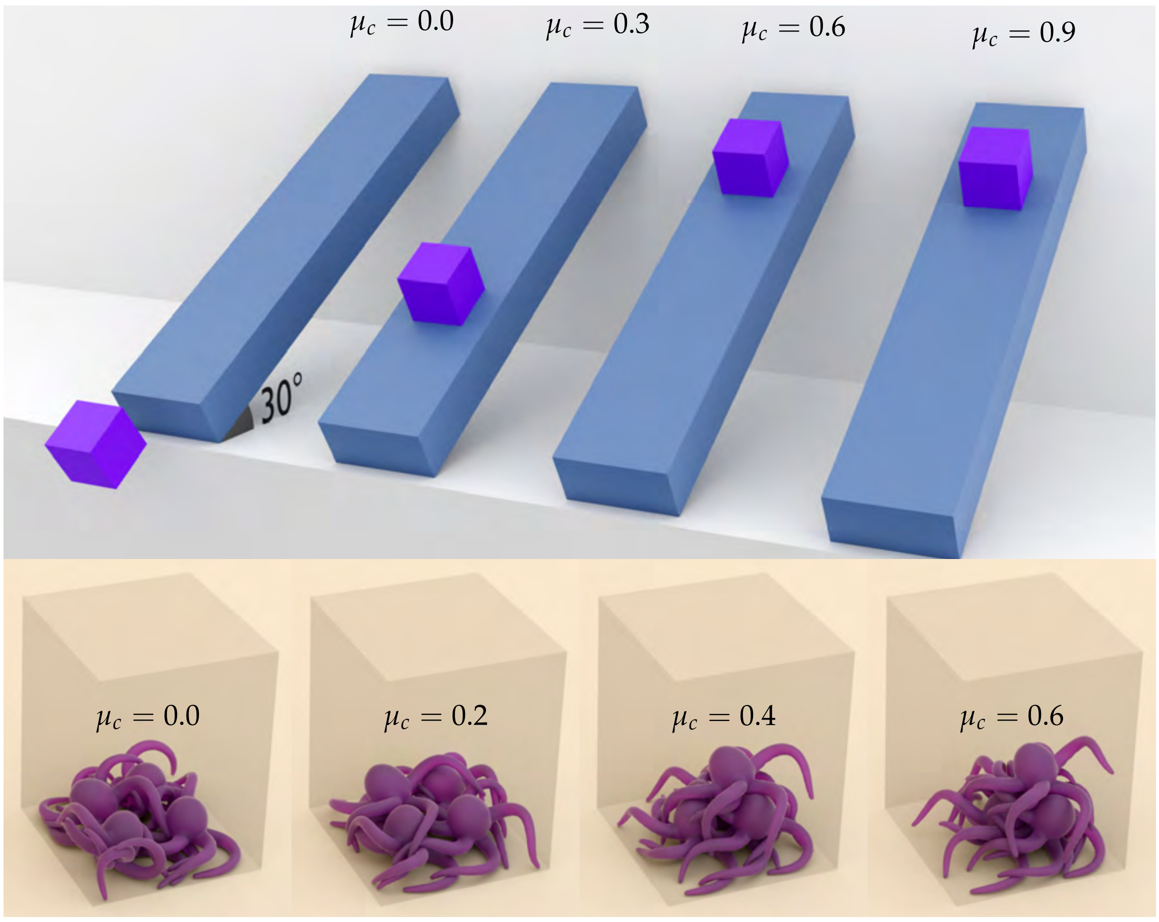


Figure 3.16: Testing different friction coefficients μ_c for (top) an elastic cube with 400 vertices and 1.45 thousand tetrahedra, initially resting on an incline, and (bottom) 4 elastic octopus models, totaling 15.6 thousand vertices and 60 thousand tetrahedra, dropped into a box.

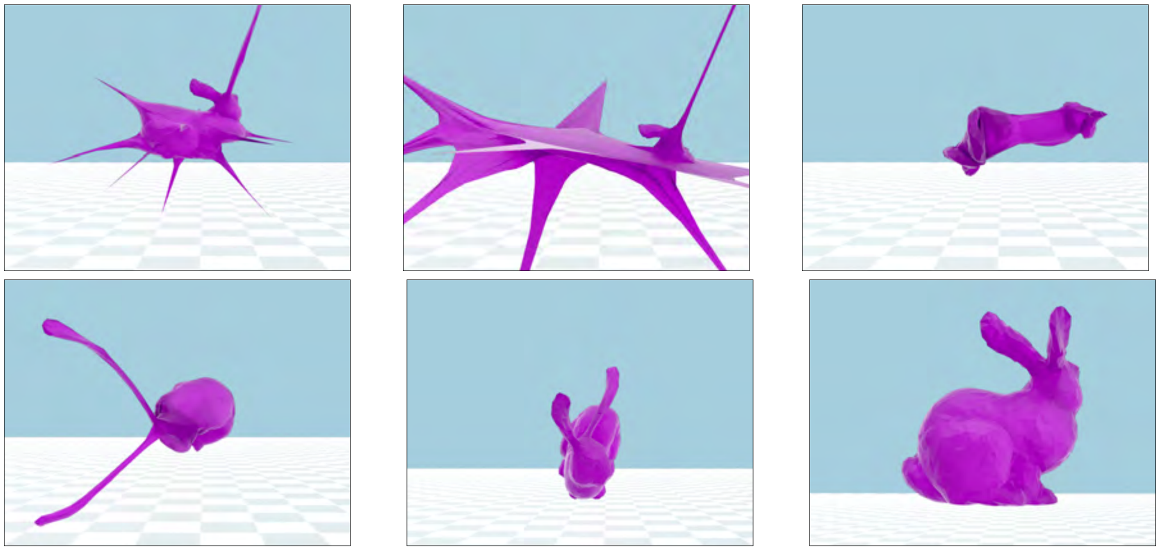


Figure 3.17: A stress test with extreme stretching: a Stanford bunny model with 1.8 thousand vertices and 5.9 thousand tetrahedra is stretched by slowly pulling 10 vertices away, which are then suddenly released. The model recovers its shape after going through considerable deformations and high-velocity motion, simulated with self-collisions and using $S = 5$ substeps and $n_{\text{max}} = 10$ iterations per step.

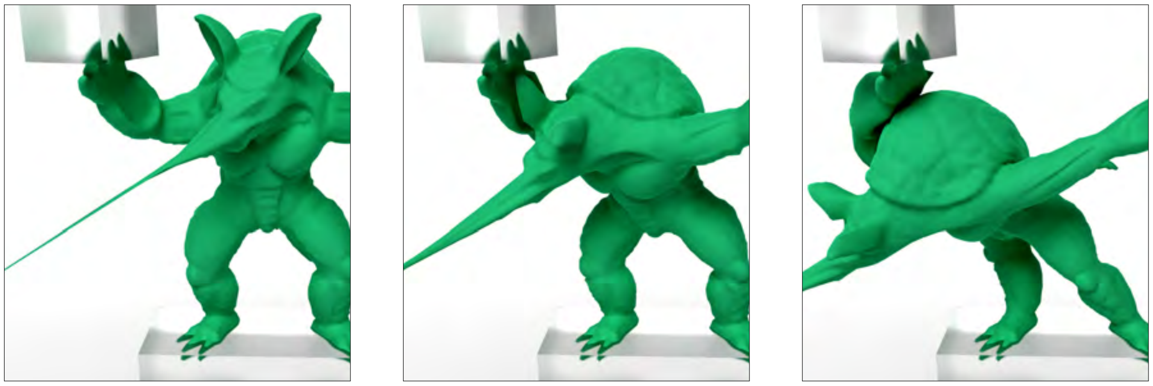


Figure 3.18: A stress test using only a single iteration per frame (i.e., a time step of $h = 1/60$ seconds and $n_{\text{max}} = 1$). One vertex on the armadillo model's nose is pulled while the finger and toe vertices are fixed. The model has 15 thousand vertices and 50 thousand tetrahedra.

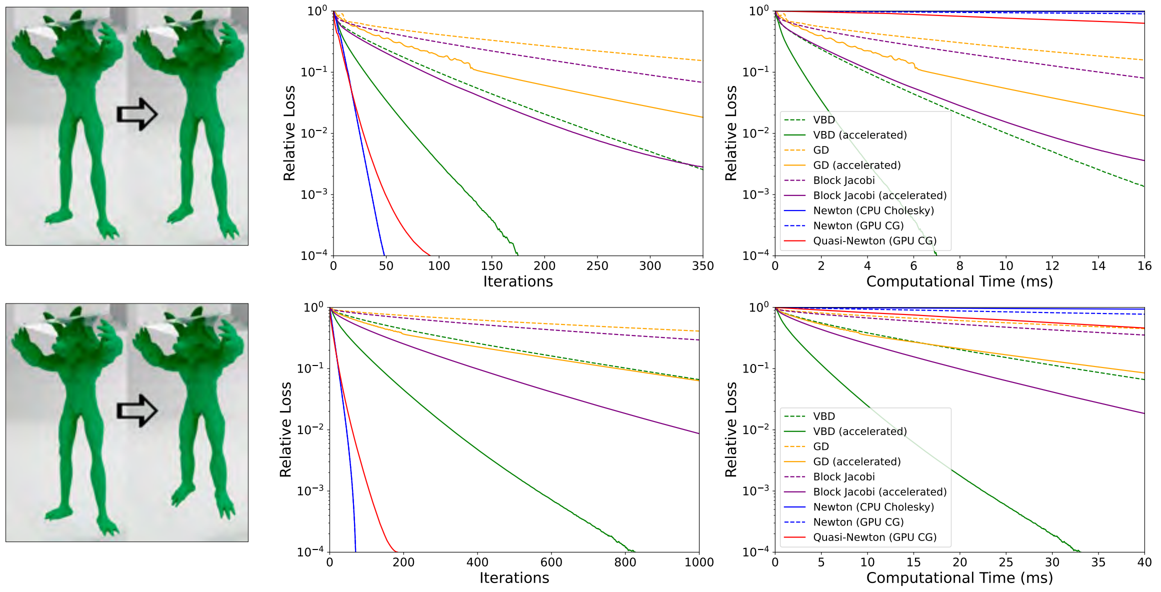


Figure 3.19: Convergence of different descent methods for simulating an armadillo model with 15 thousand vertices and 50 thousand tetrahedra with (top) a relatively soft material and (bottom) a 10× stiffer material. Vertices near the top inside the glass block are fixed and the models are initially stretched, as shown on the left, by pulling down foot vertices. Then, the position constraints on foot vertices are suddenly removed, allowing the model to deform for 33 ms. The deformation is computed using a single time step of $h = 33$ ms. The graphs show relative loss over iterations and computation time. All methods are implemented on the GPU using the same framework with single precision (32-bit) floating-point numbers, except for Newton's method with Cholesky factorization, which runs on the CPU using double precision (64-bit). Accelerated versions use $\rho = 0.95$.

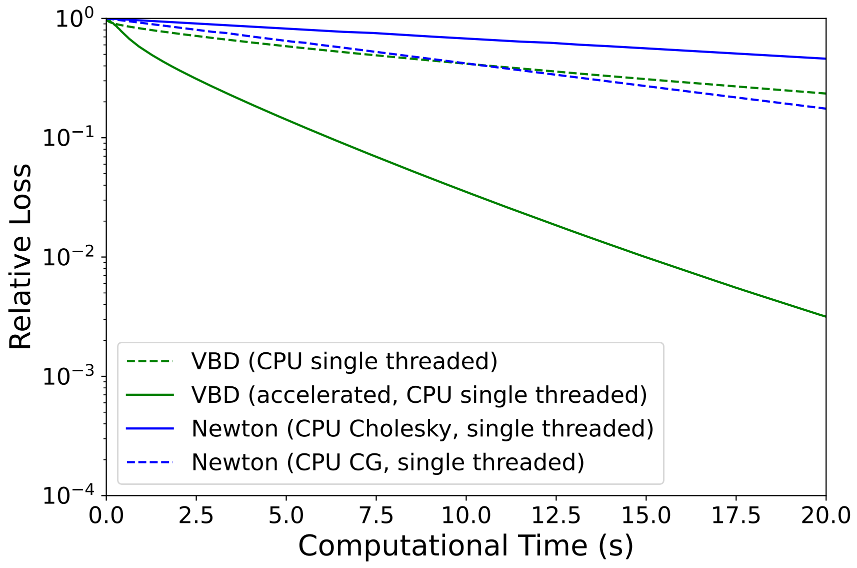


Figure 3.20: Comparing a single-threaded CPU implementation of our method with single-threaded Newton's method (using both CG and Cholesky). The scene is identical to the bottom row of Figure 3.19 .

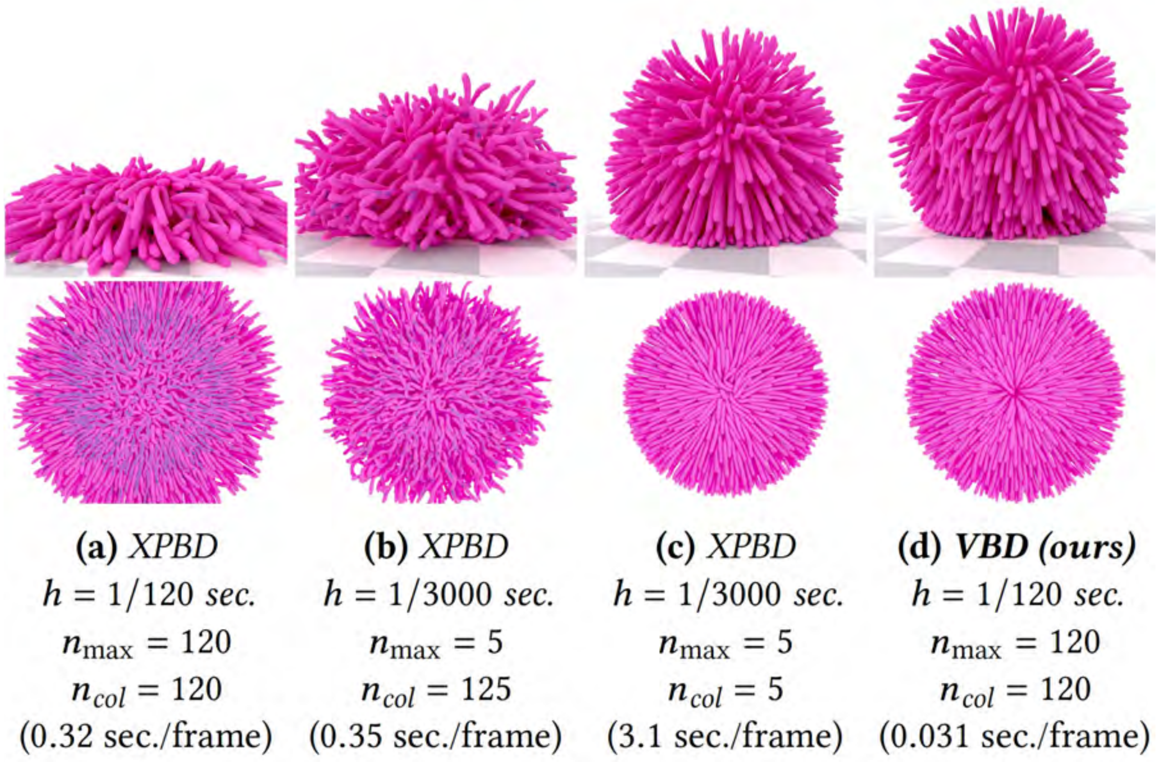


Figure 3.21: A squishy ball with tentacles, comprising 230 thousand vertices and 700 thousand tetrahedra, dropped on the ground, simulated using (a) XPBD with a large time step and 240 iterations per frame, (b) XPBD with a 25× smaller time step and 250 total iterations per frame, (c) XPBD with the same small time step and iteration count but with 25× more frequent collision detection, and (d) VBD with a large time step and 240 iterations per frame. Comparing (a) and (d), VBD is faster than XPBD with the same settings. XPBD's solution approaches VBD as the time step decreases, but it also requires more frequent collision detection to achieve a visually similar result to VBD.

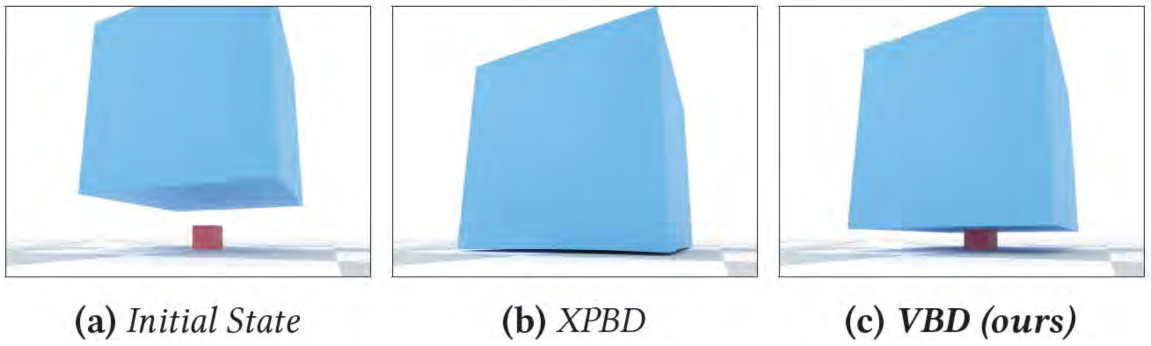


Figure 3.22: Dropping a large and heavy elastic cube onto a smaller and much lighter box with a mass ratio of 1:2000. Each cube has 400 vertices and 1.5 thousand tetrahedra.

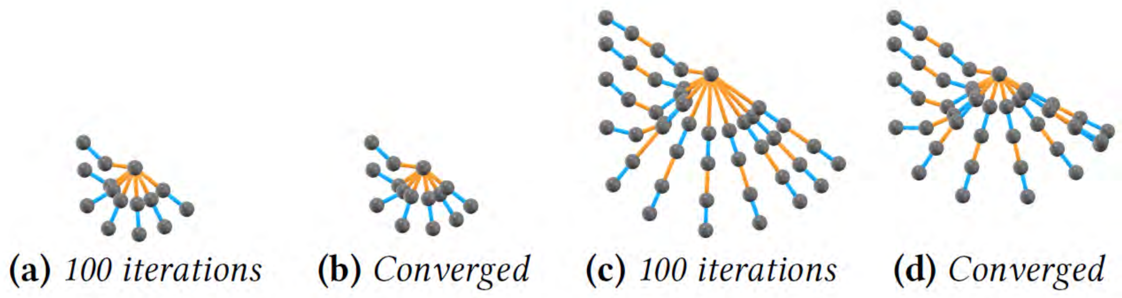


Figure 3.23: A chain of particles connected with soft springs (orange) and 10,000$\times$ stiffer (blue) springs. Simulations with VBD using (a,c) 100 iterations per frame fail to converge and result in excessive extensions, as compared to (b,d) converged results.

Table 3.1: Performance results and simulation parameters.

Experiment Name	Number of			Material			Contact &Friction		Simulation Parameters		Time per step
	Vert.	Tet.	Color	Type	Stiffness	Damping	k_c	μ_c, ϵ_v	h(sec.)	Iterations	avg./max
Twisting Thin Beams (Figure 3.1)	97K	266K	8	NeoHookean	$\mu = 5e4, \lambda = 1e6$	1e-6	1e6	0.1, 1e-2	1/300	100	60/78ms
Flattening initialization (Figure 3.2)	15K	50K	8	NeoHookean	$\mu = 2e6, \lambda = 1e7$	1e-6	NA	NA	1/60	100	3.3/3.8ms
Random initialization (Figure 3.2)	2K	8.5K	8	NeoHookean	$\mu = 2e6, \lambda = 1e7$	1e-6	NA	NA	1/60	100	2.6/2.8ms
Tetmesh Pendulum (Figure 3.5)	304	755	6	NeoHookean	$\mu = 1e7, \lambda = 1e8$	0	NA	NA	1/300	20	¡0.1ms
Squishy Ball Drops (Figure.3.6,3.8,3.21)	230K	700K	8	NeoHookean	$\mu = 2e6, \lambda = 2e7$	1e-7	1e7	0.1, 1e-2	1/120	120	15/17ms
Tearing Cloth (Figure 3.9)	2500	4800	3	StVK	$\mu = 1e4, \lambda = 1e4$	1e-5	NA	NA	1/300	20	11.2/11.5 ms(cpu)
Dropping 216 Squshy Balls (Figure 3.13)	48M	151M	9	NeoHookean	$\mu = 2e6, \lambda = 2e7$	1e-7	1e7	0.1, 1e-2	1/240	40	3.6/3.9s
Dropping 10368 Models (Figure 3.14)	36M	124M	8	NeoHookean	$\mu = 1e6, \lambda = 1e7$	1e-7	1e7	0.1, 1e-2	1/120	60	4.2/4.7s
Beam Sagging (Figure 3.15)	463	1.5K	6	NeoHookean	$\mu = 1e6/3e6/1e7$	1e-6	NA	NA	1/300	3/5/10	avg.: 0.08/0.12/0.24ms
					$\lambda = 1e7/3e7/1e8$			NA			max: 0.08/0.16/0.31ms
Cude Sliding (Figure 3.16)	800	2.9K	6	NeoHookean	$\mu = 1e6, \lambda = 1e7$	1e-6	1e7	0/0.3/0.6/0.9, 1e-2	1/300	10	0.10/0.17ms
Octopi Stacking (Figure 3.16)	15.6K	60K	8	NeoHookean	$\mu = 1e6, \lambda = 1e7$	1e-6	1e7	0/0.1/0.4/0.6, 1e-2	1/300	10	1.1/1.3ms
Extreme Stretch (Figure 3.18)	1.8K	5.9K	8	NeoHookean	$\mu = 2e6, \lambda = 1e7$	1e-6	1e7	0.2, 1e-2	1/300	10	0.36/1.02ms

CHAPTER 4

SHORTEST PATH TO BOUNDARY FOR SELF-INTERSECTING MESHES

In this section, we propose an algorithm to compute the shortest path to the boundary for self-intersecting meshes. Using this path, we can derive collision energy E_n which can be built into Equation 2.8 to handle collisions between volumetric objects.

A typical solution for resolving intersections (detected via DCD) is finding the closest boundary point for each intersecting point and then applying corresponding forces/constraints along the line segment toward this point, i.e., the shortest path to boundary. The length of this path is the penetration depth. A collision energy can be defined based on the penetration depth to resolve the intersection.

When two separate objects intersect, finding the closest boundary point is a trivial problem: it is the closest boundary point on the other object. In the case of self-intersections, however, even the definition of the shortest path to boundary is somewhat ambiguous.

Consider a point on the boundary and also inside the object due to self-intersections. Since this point is already on the boundary, its Euclidean closest boundary point would be itself. Yet, this information is not helpful for resolving the self-intersection. Therefore, we must give a proper definition for the shortest path to surface in self-intersecting meshes.

4.1 Shortest Path to Boundary

In this section, we provide a formal definition of the shortest path to boundary based on the geodesic path of the object in the presence of self-intersections (Section 4.1.1). Then, we present an efficient algorithm to compute it for triangular/tetrahedral meshes in 2D/3D, respectively, (Section 4.1.2). We also describe how to handle meshes that contain some inverted elements, (Section 4.1.5). The resulting method provides a robust solution for handling self-collisions that can be used with various simulation methods and collision resolution techniques (using forces or constraints).

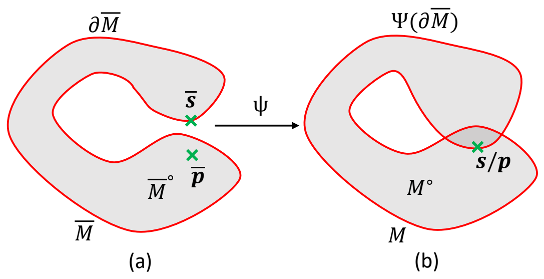


Figure 4.1: Illustrations of the notations. (a) Notations on the undeformed pose. (b) Notations on the deformed model. The image of the undeformed pose boundary $\Psi(\partial\overline{M})$ is marked as the red curve.

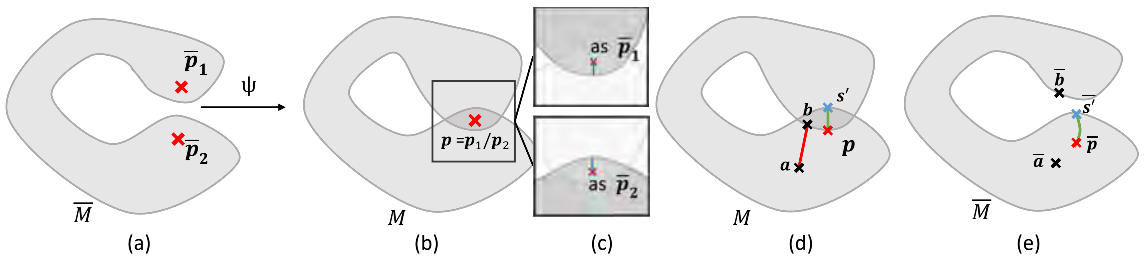


Figure 4.2: Illustration of the definition of shortest path to surface in self-intersecting meshes. (a) An intersection-free pose of the deformable model $\overline{M}$. $\overline{\mathbf{p}}_1, \overline{\mathbf{p}}_2 \in \overline{M}^\circ$. (b) $\overline{M}$'s image under Ψ, where $\overline{\mathbf{p}}_1, \overline{\mathbf{p}}_2$ are mapped to the same point $\mathbf{p}$. (c) Treated as different pre-image, $\mathbf{p}$ has different shortest paths to the boundary (blue line). (d) Two paths are contained by M. (e) Only $\mathbf{ps}'$ is a valid path.

4.1.1 Shortest Path to Boundary

Consider a self-intersecting model M, such that a boundary point $\mathbf{s}$ coincides with an internal point $\mathbf{p}$. Figure 4.1b shows a 2D illustration, though the concepts we describe here apply to 3D (and higher dimensions) as well. In this case, $\mathbf{s}$ and $\mathbf{p}$ have the same geometric positions, but topologically they are different points. In fact, to fix the self-intersection, we need to apply a force/constraint that would move $\mathbf{s}$ along $\mathbf{p}$'s geodesic shortest path to boundary.

To provide a formal definition of this geodesic shortest path, we consider a self-intersection-free form of this model as $\overline{M}$ which we call undeformed pose, and a deformation Ψ that maps all points in $\overline{M}$ to its current shape M, such that $M = \Psi(\overline{M})$. Note that our algorithm (explained in Section 4.1.2) does not actually need computing $\overline{M}$ or Ψ. For any point $\overline{\mathbf{p}}$ in $\overline{M}$, we represent its image under Ψ as $\mathbf{p} \in M$, such that $\mathbf{p} = \Psi(\overline{\mathbf{p}})$. In the following, we assume that $\overline{M}$ is a path-connected (i.e., a single piece) manifold, though the concepts below can be trivially extended to models with multiple separate pieces.

To cause self-intersection, Ψ should not be injective. In this case, Ψ is an immersion of $\overline{M}$ but not embedding, meaning multiple points from $\overline{M}$ are mapped to the same position $\mathbf{p}$ inside M. To differentiate such points that coincide in M, we label them using their unique positions in $\overline{M}$. For simplicity, we say $\mathbf{p}$ *as* $\overline{\mathbf{p}}$, when we are referring $\mathbf{p}$ as the image of $\overline{\mathbf{p}}$.

For simplicity, let us consider non-degenerate Ψ that forms no inversion, i.e., $\det(\nabla\Psi) > 0$. We discuss inversions later in Section 4.1.5. Note that under this Ψ, the boundary of the undeformed model $\partial\overline{M}$ does not completely overlap with the boundary of the deformed model ∂M, i.e., $\Psi(\partial\overline{M}) \neq \partial M$, see Figure 4.1b. We use $\overline{M}^{\circ}$ to denote the set of interior points of $\overline{M}$, such that $\overline{M} = \partial\overline{M} \cup \overline{M}^{\circ}$.

Let $\mathbf{s}$ as be a point on the boundary, i.e., $\mathbf{s} \in \Psi(\partial\overline{M})$ and we refer to it as an undeformed pose boundary point $\overline{\mathbf{s}}$. For a given point $\mathbf{p}$ (as $\overline{\mathbf{p}}$), we can construct a path $\mathbf{c}(t) : [0,1] \mapsto M$ as a continuous curve that connects $\mathbf{p} = \mathbf{c}(0)$ to $\mathbf{s} = \mathbf{c}(1)$.

Definition 1 (Valid path)**.** *The path* $\mathbf{c}(t)$ *from* $\mathbf{p}$ *(as* $\overline{\mathbf{p}}$*) to* $\mathbf{s}$ *(as* $\overline{\mathbf{s}}$*) is a* valid path *if there exists a continuous curve* $\overline{\mathbf{c}}(t) : [0,1] \mapsto \overline{M}$ *such that* $\mathbf{c}(t) = \Psi(\overline{\mathbf{c}}(t))$, $\overline{\mathbf{c}}(0) = \overline{\mathbf{p}}$, $\overline{\mathbf{c}}(1) = \overline{\mathbf{s}}$.

Based on this definition, a *valid path* must be the image of a path that is fully contained within $\overline{M}$, which connects the two points on the undeformed pose we are referring to. Any path that moves outside of $\overline{M}$ is considered an *invalid path*; see Figure 4.2de. Our goal is to find the shortest valid path from a given point $\mathbf{p}$ (as $\overline{\mathbf{p}}$) to the boundary.

Definition 2 (Shortest path to boundary)**.** *For an interior point* $\mathbf{p}$ *(as* $\overline{\mathbf{p}}$*), the* shortest path to boundary *is the shortest curve* $\mathbf{c}(t)$ *in* M *that connects* $\mathbf{p}$ *to a boundary point* $\mathbf{s}$ *(as* $\overline{\mathbf{s}}$*) that is a valid path between* $\mathbf{p}$ *and* $\mathbf{s}$*.*

Definition 3 (Closest boundary point)**.** *For an interior point* $\mathbf{p}$ *(as* $\overline{\mathbf{p}}$*), the* closest boundary point *is the boundary point* $\mathbf{s}$ *(as* $\overline{\mathbf{s}}$*) at the other end of* $\mathbf{p}$*'s shortest path to boundary* $\mathbf{c}(t) = \Psi(\overline{\mathbf{c}}(t))$*, such that* $\mathbf{s} = \mathbf{c}(1)$ *and* $\overline{\mathbf{s}} = \overline{\mathbf{c}}(1)$*.*

Here we must emphasize that the definition of the shortest path is dependent on the pre-image point we are referring to. For a point located at the overlapping part of M, referring to it as a different point on the undeformed pose may lead to a different shortest path to the boundary (see Figure 4.2c). Also, this definition is equivalent to the image of $\overline{\mathbf{p}}$'s global geodesic path to boundary in $\overline{M}$ evaluated under the metrics pulled back by Ψ. Thus the shortest path we defined is a special class of geodesics.

To construct an efficient algorithm for finding the shortest path, we rely on two properties:

- First, by definition, the shortest path must be a continuous curve that is fully contained inside undeformed model $\overline{M}$.
- Second, the shortest path (under the Euclidean distance metrics) that connects two points in the deformed model M must be a line segment.

Based on these properties, we can construct and prove the fundamental theorem of our algorithm:

Theorem 1. *For any point* $\mathbf{p} \in M$ *(as* $(\overline{\mathbf{p}}$*), its shortest path to the boundary is the shortest line segment from* $\mathbf{p}$ *to a boundary point* $\mathbf{s} \in \Psi(\partial\overline{M})$ *(as* $\overline{\mathbf{s}}$*), that is a valid path.*

Here we verbally prove the theorem, we also provide a formal proof in the supplementary document. If the shortest path is not a line segment, we can continuously deform it into a line segment, while keeping the end points fixed. This procedure can induce a deformation on the undeformed pose, which continuously deforms the pre-image of that curve to the pre-image of the line segment, while keeping the end points fixed. This is always achievable because the curve cannot touch the boundary of the undeformed pose during the deformation, otherwise, we will form an even shorter path to the boundary. Thus the line segment is also a valid path.

Based on these properties, our algorithm investigates a set of candidate boundary points $\mathbf{s}$ and checks if the line segment from the interior point $\mathbf{p}$ to $\mathbf{s}$ is a valid path. This is accomplished without having to construct $\overline{M}$ or determine the deformation Ψ by relying on the topological connections of the given discretized model.

4.1.2 Shortest Path to Boundary for Meshes

In practice, models we are interested in are discretized in a piecewise linear form. These are triangular meshes in 2D and tetrahedral meshes in 3D. We refer to each piecewise linear component as an *element* (i.e., a triangle in 2D and a tetrahedron in 3D) and the one-dimension-lower-simplex shared by two topologically-connected elements as a *face* (i.e., an edge between two triangles in 2D and a triangular face between two tetrahedra in 3D). This discretization makes it easy to test the validity of a given path, without constructing a self-intersection-free $\overline{M}$ or the related deformation Ψ.

We propose the concept of *element traversal* for meshes, as a sequence of topologically connected elements:

Definition 4 (Element traversal)**.** *For a mesh M, and two-point $\mathbf{a} \in e_{\mathbf{a}}, \mathbf{b} \in e_{\mathbf{b}}$, we define a element traversal from $\mathbf{a}$ to $\mathbf{b}$ as a list of elements $\mathcal{T}(\mathbf{a}, \mathbf{b}) = (e_0, e_1, e_2, \dots, e_k)$, where e_i is a element of M, $e_0 = e_{\mathbf{a}}$, $e_k = e_{\mathbf{b}}$, and $e_i \cap e_{i+1}$ must be a face.*

Specifically, we call it *tetrahedral traversal* for 3D meshes, and *triangular traversal* for 2D meshes.

Let $\mathbf{c}(t)$ be a line segment from a point $\mathbf{p}$ inside an element $e_{\mathbf{p}}$ to a boundary point $\mathbf{s}$ of a boundary element $e_{\mathbf{s}}$ (with a boundary face that contains $\mathbf{s}$). If $\mathbf{c}(t)$ is a valid path, there must be a corresponding piecewise linear path $\overline{\mathbf{c}}(t)$ in $\overline{M}$ from $\overline{\mathbf{p}}$ to $\overline{\mathbf{s}}$ that passes through an element traversal of $\overline{M}$. Actually, an element traversal containing $\mathbf{c}(t)$ is the sufficient and necessary condition for $\mathbf{c}(t)$ being a valid path. Please see the supplementary material for a rigorous proof.

Thus, evaluating whether $\mathbf{c}(t)$ is a valid path, is equivalent to searching for an element traversal from $\bar{\mathbf{s}}$ to $\bar{\mathbf{p}}$, and a piece-wise linear curve $\bar{\mathbf{c}}(t) : I \mapsto \overline{M}$ defined on it, such that $\mathbf{c}(t) = \Psi(\bar{\mathbf{c}}(t))$. Such an element traversal and piece-wise linear curve can be efficiently constructed in M.

Going through the element traversal, $\bar{\mathbf{c}}(t)$ must pass through faces shared by neighoring elements at points $\bar{\mathbf{r}}_i \in e_i \cap e_{i+1}$, where $i = 0, 1, 2, \ldots, k-1$. When Ψ forms no inversion, corresponding face points $\mathbf{r}_i$ must be along the line segment $\mathbf{c}(t)$, i.e., $\mathbf{r}_i = \mathbf{c}(t_i)$ for some $t_i \in [0, 1]$; see Figure 4.3a. If we can form such an element traversal using the topological connections of the model, we can safely conclude that the path is valid.

This gives us an efficient mechanism for testing the validity of the shortest path from $\mathbf{p}$ to $\mathbf{s}$. Starting from $e_\mathbf{p}$, we trace a ray from $\mathbf{p}$ towards $\mathbf{s}$ and find the first face point $\mathbf{r}_0$. If $\mathbf{r}_0$ is not on the boundary, this face must connect $e_\mathbf{p}$ to a neighboring element e_1. Then, we enter e_1 from $\mathbf{r}_0$ and trace the same ray to find the exit point $\mathbf{r}_1$ on another face. We continue traversing until we reach $e_\mathbf{s}$, in which case we can conclude that this is a valid path; see Figure 4.3a. This also includes the case $e_\mathbf{p} = e_\mathbf{s}$. If we reach a face point $\mathbf{r}_i$ that is on the boundary (see Figure 4.3b) or we pass-through $\mathbf{s}$ without entering $e_\mathbf{s}$, $\mathbf{s}$ cannot be the closest boundary point to $\mathbf{p}$.

This process allows us to efficiently test the validity of a path to a given boundary point, but we have infinitely many points on the boundary to test. Fortunately, we are only interested in the shortest path and we can use the theorem below to test only a single point per boundary face.

Theorem 2. *For each interior point* $\mathbf{p}$ *(as* $\bar{\mathbf{p}}$*), if its closest boundary point* $\mathbf{s}$ *(as* $\bar{\mathbf{s}}$*) is on the boundary face* f*,* $\mathbf{s}$ *must also be the Euclidean closest point to* $\mathbf{p}$ *on* f*.*

The proof is similar to Theorem 1, which is included in the supplementary document. Based on Theorem 2, we only need to check a single point (the Euclidean closest point) on each boundary face to find the closest boundary point. If we test these boundary points in the order of increasing distance from the interior point $\mathbf{p}$, as soon as we find a valid path to one of them, we can terminate the search by returning it as the closest boundary point. In practice, we use a BVH (bounding volume hierarchy) to test these points, which allows

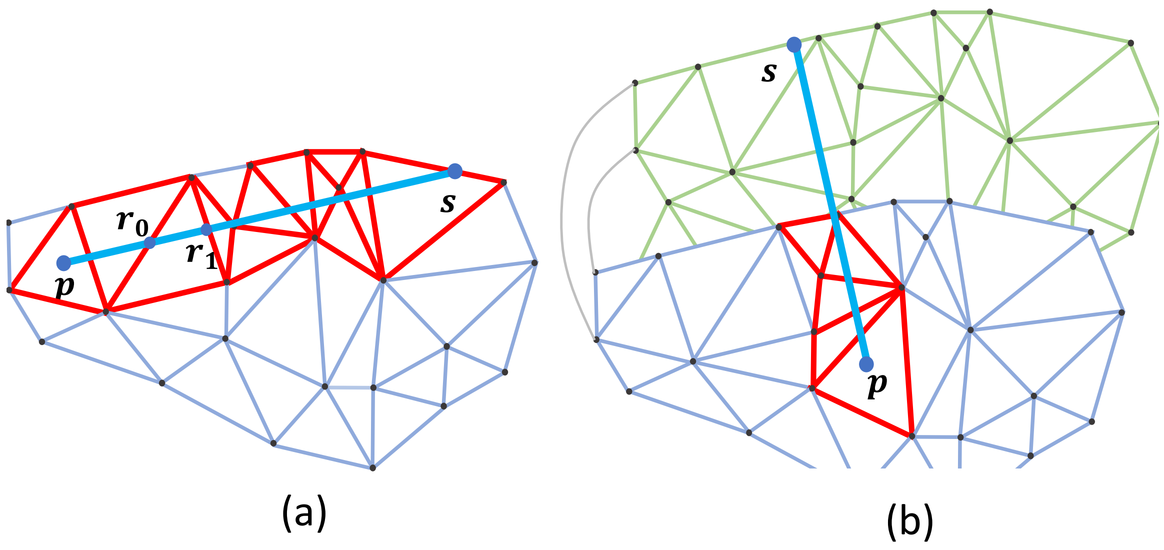


Figure 4.3: Illustration of elemental traversal. (a) An example of a triangular traversal, marked by red triangles. A line segment connecting **p** and **s** is included in this triangular traversal. (b) An example of a line segment being an invalid path when there are self-intersections, the triangular traversal (marked by the red triangles) stops at the boundary of the mesh but the line segment penetrates the boundary and continues going.

testing them approximately (though not strictly) in the order of increasing distance and, once a valid path is found, quickly skipping the further away bounding boxes.

4.1.3 Robust Topological Ray Traversal

The process we describe above for testing the validity of the linear path to a candidate boundary point involves traversing a ray through the mesh. This ray traversal is significantly simpler than typical ray traversal algorithms used for rendering with ray tracing. This is because it directly follows the topological connections of the mesh.

At each step, the ray enters an element through one of its faces and must exit from one of its other faces. Therefore, we do not need to rely on an acceleration structure to quickly determine which faces to test ray intersections, as they are directly known from the mesh topology. In fact, we do not need to check each one of the other faces individually, since the ray exits from exactly one of them. Therefore, we can quickly test all possible exit faces together.

For example, [187] present such a tetrahedral traversal algorithm in 3D. Yet, due to limited numerical precision, this algorithm is prone to forming infinite loops. Such infinite loops are easy to detect and terminate (e.g., using a maximum iteration count), but such premature terminations are entirely unacceptable in our case. This is because incorrectly deciding on the validity of a path would force our algorithm to pick an incorrect shortest path to boundary, which can be arbitrarily far from the correct one. Therefore, the simulation system that relies on this shortest path to boundary can place strong and arbitrarily incorrect forces/constrains in an attempt to resolve the self-intersection.

Our solution for properly resolving such cases that arise from limited numerical precision is three fold:

1. We allow ray intersections with more than one face by effectively extending the faces using a small tolerance parameter ϵ_i in the intersection test. This forms branching paths when a ray passes between multiple faces and, therefore, intersects (within ϵ_i) with more than one of them.

2. We keep a list of traversed elements and terminate a branch when the ray enters an element that was previously entered.

3. We keep a stack containing all the candidate intersecting faces from the intersection test. After a loop is detected, we pick the latest element from it and continue the process.

Please see our supplementary material for the pseudo-code and more detailed explanations of our algorithm.

In practice such branching happens rarely, but solution ensures that we never incorrectly terminate the ray traversal. Note that ϵ_i is a conservative parameter for extending the ray traversal through branching to prevent problems of numerical accuracy issues. It does not introduce any error to the final shortest paths we find. Using an unnecessarily large ϵ_i would only have negative, though mostly imperceptible, performance consequences. We verified this by making the ϵ_i ten times larger, which did not result in a measurable performance difference.

One corner case is when the internal point $\mathbf{p}$ (as $\overline{\mathbf{p}}$) and the boundary point $\mathbf{s}$ (as $\overline{\mathbf{s}}$) coincide, such that $\mathbf{p} = \mathbf{s}$ (within numerical precision). This forms a line segment with zero length and, therefore, does not provide a direction for us to traversal. This happens when testing self-intersections of boundary points, which pick themselves as their first candidate for the closest boundary point. This zero-length line segment cannot be a valid

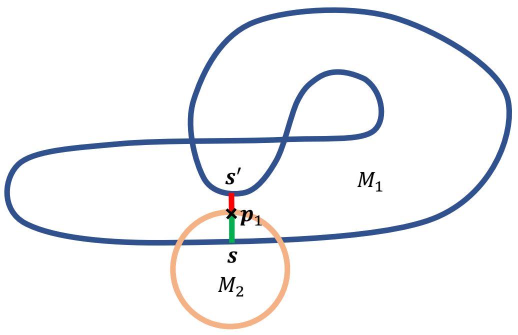


Figure 4.4: An object M_2 intersects with a self-intersecting object M_1. A surface point of M_2 is overlapping with an interior point $\mathbf{p}_1 \in M_1$. $\mathbf{s}$ and $\mathbf{s}'$ are $\mathbf{p}_1$'s closest boundary point by our definition and Euclidean closest boundary point, respectively.

path. Fortunately, since we know we are testing self-intersection for $\mathbf{s}$, when the BVH query returns the boundary face includes $\mathbf{s}$, we can directly reject it.

4.1.4 Intersections of Different Objects

Although our method is mainly designed for solving self-intersections, it is still needed for handling intersections of different objects when they may have self-intersections as well. As shown in Figure 4.4, an object M_2 intersects with a self-intersecting object M_1, where a surface point of M_2 is overlapping with an interior point $\mathbf{p}_1 \in M_1$. Simply querying for $\mathbf{p}_1$'s Euclidean closest boundary point in M_1 will give us $\mathbf{s}'$, which does not help resolve the penetration. This is because $\mathbf{p}_1\mathbf{s}'$ is not a valid path between $\mathbf{p}_1$ (as $\overline{\mathbf{p}}_1 \in \overline{M}_1^{\circ}$) and $\mathbf{s}_1'$ as ($\overline{\mathbf{s}}' \in \partial\overline{M}_1$). What is actually needed is $\mathbf{p}_1$'s shortest path to boundary as $\overline{\mathbf{p}}_1$, which is the same problem as the self-intersection case, a surface point of M_1 is overlapping with an interior point $\mathbf{p}_1 \in M_1$.

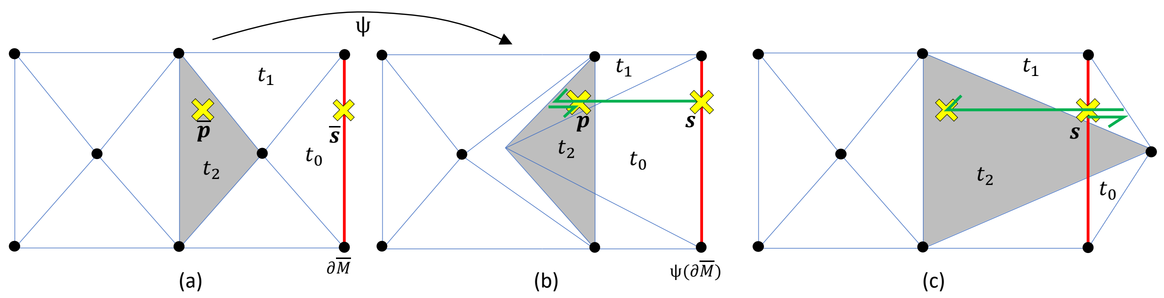


Figure 4.5: Handling internal inverted elements. (a) A part of the undeformed pose of a triangular mesh $\overline{M}$, which is inversion free. $\overline{\mathbf{p}} \in \overline{M}, \overline{\mathbf{s}} \in \partial\overline{M}$. A surface edge is marked with red color. (b) The image of $\overline{M}$ under Ψ, the tetrahedron t_2 (colored with gray), is inverted by Ψ. The green line illustrates $\mathbf{p}$'s global geodesics to the surface, it has a self-overlapping part, which is marked by the two-sided arrow. (c) An interior tetrahedron is inverted and got out of the surface. In this case, the global geodesics to the surface path can go backward.

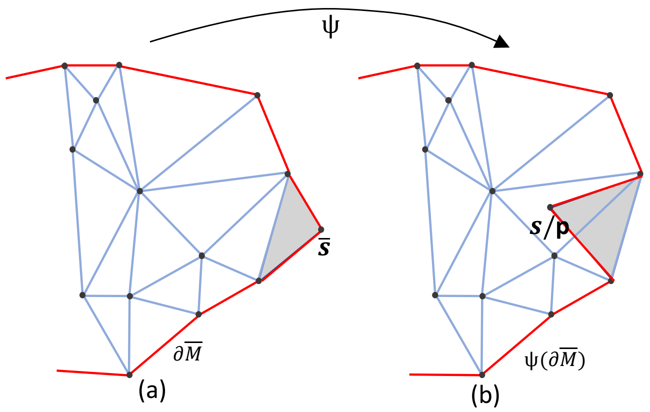


Figure 4.6: Handling surface inverted elements. (a) A part of the undeformed pose of a triangular mesh $\overline{M}$, which is inversion free. The surface edges are marked with red color. (b) After deformation, a triangle (marked by gray color), is inverted and folded into the interior of the mesh. A deformed surface point $\mathbf{s}$ overlaps with the interior point $\mathbf{p}$.

4.1.5 Inverted Elements

Our derivations in Section 4.1.1 assume that $det(\nabla\Psi) > 0$ everywhere. For a discrete mesh, this would mean no inverted or degenerate elements. Unfortunately, though inverted elements are often highly undesirable, they are not always unavoidable. Fortunately, the algorithm we describe above can be slightly modified to work in the presence of certain types of inverted elements.

If the inverted elements are not a part of the mesh boundary, we can still test the validity of paths by allowing the ray traversal to go backward along the ray. This is because the ray would need to traverse backward within inverted elements. In addition, we cannot simply terminate the traversal once the ray passes through the target point, because an inverted element further down the path may cause backward traversal to reach (or pass through) the target point; see Figure 4.5b. Therefore, ray traversal must continue until a boundary point is reached. We also need to allow the ray to go behind the starting point; see Figure 4.5c.

A consequence of this simple modification to our algorithm is that, when we begin from an internal point $\mathbf{p}$ toward a boundary point $\mathbf{s}$, it is unclear if we would reach $\mathbf{s}$ by beginning the traversal toward $\mathbf{s}$ or in the opposite direction. While one may be more likely, both are theoretically possible.

To avoid this decision, in our implementation, we start the traversal from the target boundary point $\mathbf{s}$. In this case, there is no ambiguity, since there is only one direction we can traverse along the ray. This also allows using the same traversal routine for the first element and the other elements along the path by always entering an element from a face. Therefore, it is advisable even in the absence of inverted elements.

Nonetheless, our algorithm is not able to handle all possible inverted elements. For example, if the inverted element is on the boundary, as shown in Figure 4.6, the inversion itself can cause self-intersection. In such a case, a surface point $\mathbf{s}$ is overlapping with an interior point $\mathbf{p}$ (as $\overline{\mathbf{p}}$). Our algorithm will not be able to try to construct a tetrahedral traversal between those two points because we cannot determine a ray direction for a zero-length line segment. Actually, in this case, the very definition of the closest boundary point can be ambiguous.

Our solution is to skip the self-intersection detection of inverted boundary elements. As a result, the only way for us to solve such self-intersections caused by inverted boundary elements is to resolve the inversion itself. Fortunately, inverted elements are undesirable for most simulation scenarios, and they are often easier to fix for boundary elements. Unfortunately, if the inverted boundary elements have global self-intersections with other parts of the mesh, our solution ignores them. Though this does not form a complete solution, because the inverted boundary elements are rare, the other boundary elements surrounding the inverted elements are often enough to solve the global self-intersection.

4.1.6 Infeasible Region Culling

In a lot of cases, it is possible to determine that a given candidate boundary point $\mathbf{s}$ cannot be the closest boundary point to an interior point $\mathbf{p}$, purely based on the local information about the mesh around $\mathbf{s}$, without performing any ray traversal.

For this test we construct a particular region of space, i.e., the *feasible region*, around $\mathbf{s}$. When $\mathbf{p}$ is outside of this region of $\mathbf{s}$, thus in its *infeasible region*, we can safely conclude that $\mathbf{s}$ is not the closest boundary point.

The formulation of the feasible region can be viewed as a discrete application of the well-known Hilbert projection theorem. The construction of this feasible region depends on whether $\mathbf{s}$ is on a vertex, edge, or face.

4.1.6.1 Vertex Feasible Region

In 2D, when $\mathbf{s}$ is on a vertex, the feasible region is bounded by the two lines passing through the vertex and perpendicular to its two boundary edges, as shown in Figure 4.7a. For a neighboring boundary edge of $\mathbf{s}$ and its perpendicular line that passes through $\mathbf{s}$, if $\mathbf{p}$ is on the same side of the line as the edge, based on Theorem 2, there must be a closer boundary point on the face. More specifically, for any neighboring boundary vertex $\mathbf{v}_i$ connected to $\mathbf{s}$ by a edge, if the following inequality is true, $\mathbf{p}$ is in the infeasible region:

$$(\mathbf{p} - \mathbf{s}) \cdot (\mathbf{s} - \mathbf{v}_i) < 0\,, \tag{4.1}$$

The same inequality holds in 3D for all neighboring boundary vertices $\mathbf{v}_i$ connected to $\mathbf{s}$ by an edge (Figure 4.7b). The 3D version of the vertex feasible region is actually the space bounded by a group of planes perpendicular to its neighboring edges.

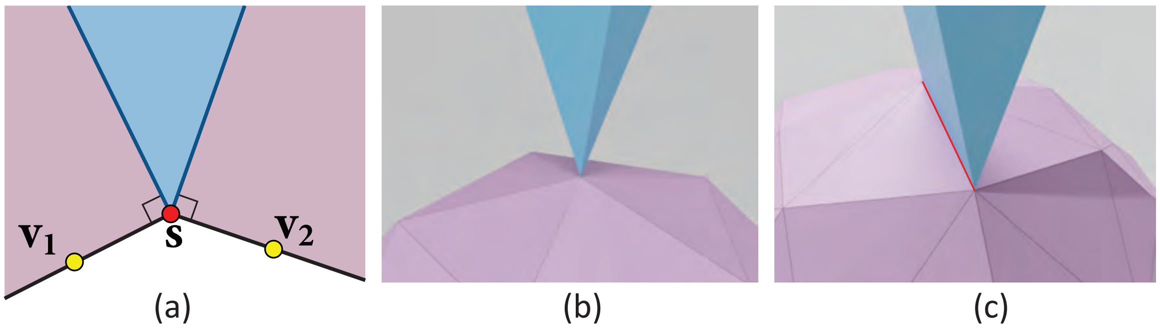


Figure 4.7: The feasible region, shaded in blue, for (a) a boundary vertex in 2D, (b) a boundary vertex in 3D, and (c) a boundary edge in 3D. Note that the 3D meshes in (b) and (c) are observed from the inside.

4.1.6.2 Edge Feasible Region

In 3D, when $\mathbf{s}$ is on the edge of a triangle, its feasible region is the intersection of 4 half-spaces defined by four planes: two planes that contain the edge and perpendicular to its two adjacent faces, and two others that are perpendicular to the edge and pass through its two vertices, as shown in Figure 4.7c. Let $\mathbf{v}_0$ and $\mathbf{v}_1$ be the two vertices of the edge and $\mathbf{n}_0$ and $\mathbf{n}_1$ be the two neighboring face normals (pointing to the interior of the mesh). $\mathbf{p}$ is in the infeasible region if any of the following is true:

$$(\mathbf{p} - \mathbf{v}_0) \cdot (\mathbf{v}_1 - \mathbf{v}_0) < 0 \tag{4.2}$$

$$(\mathbf{p} - \mathbf{v}_1) \cdot (\mathbf{v}_0 - \mathbf{v}_1) < 0 \tag{4.3}$$

$$(\mathbf{p} - \mathbf{s}) \cdot (\mathbf{n}_0 \times (\mathbf{v}_1 - \mathbf{v}_0)) < 0 \tag{4.4}$$

$$(\mathbf{p} - \mathbf{s}) \cdot (\mathbf{n}_1 \times (\mathbf{v}_0 - \mathbf{v}_1)) < 0 \tag{4.5}$$

note that $\mathbf{n}_0$ is from the face whose orientation accords to $\mathbf{v}_0 \rightarrow \mathbf{v}_1$.

4.1.6.3 Face Feasible Region

We can similarly construct the feasible region when $\mathbf{s}$ in on the interior of a face as well. Nonetheless, this particular feasible region test is unnecessary, because when $\mathbf{s}$ is the closest point on the face to $\mathbf{p}$, which is how we pick our candidate boundary points (based on Theorem 2), $\mathbf{p}$ is guaranteed to be in the feasible region.

Our *infeasible region culling* technique performs the tests above and skips the ray traversal if $\mathbf{p}$ is determined to be in the infeasible region, quickly determining that $\mathbf{s}$ cannot be the closest boundary point. Due to numerical precision, the feasible region check can return false results when $\mathbf{p}$ is close to the boundary of the feasible region. There are two types of errors: false positives and false negatives. A false positive is not a big problem: it will only result in an extra traversal. But if a false negative happens, there is a risk of discarding the actual closest surface point. In practice, however, we replace the zeros on the right-hand-sides of the inequalities above with a small negative number ϵ_r to avoid false-positives due to numerical precision limits. In our tests, we have observed that infeasible region culling can provide more than an order of magnitude faster shortest path query.

4.2 Collision Handling Application

As mentioned above, an important application of our method is collision handling with DCD. When DCD finds a penetration, we can use our method to find the closest point on the boundary and apply forces or constraints that would move the penetrating point towards this boundary point.

In our tests with tetrahedral meshes, we use two types of DCD: vertex-tetrahedron and edge-tetrahedron collisions. For vertex-tetrahedron collisions, we find the closest surface point for the colliding vertex. For edge-tetrahedron collisions, we find the center of the part of the edge that intersects with the tetrahedron and then use our method to find the closest surface point to that center point. If an edge intersects with multiple tetrahedra, we choose the intersection center that is closest to the center of the edge. The idea is by keep pushing the center of the edge-tetrahedron intersection towards the surface, which eventually resolves the intersection.

This provides an effective collision handling method with XPBD [38]. Once we find the penetrating point $\mathbf{x}$ we use the standard PBD collision constraint [37]

$$c(\mathbf{x}, \mathbf{s}) = (\mathbf{x} - \mathbf{s}) \cdot \mathbf{n} \tag{4.6}$$

where $\mathbf{s}$ is the closest surface point computed by our method when this collision is from DCD, or the colliding point when it is from CCD, and $\mathbf{n}$ is the surface normal at $\mathbf{s}$. If $\mathbf{s}$ is on a surface edge or vertex, we use the area-weighted average of its neighboring face normals. The XPBD integrator applies projections on each collision constraint c to satisfy $c(\mathbf{x}, \mathbf{s}) \geq 0$. We also apply friction, following [189].

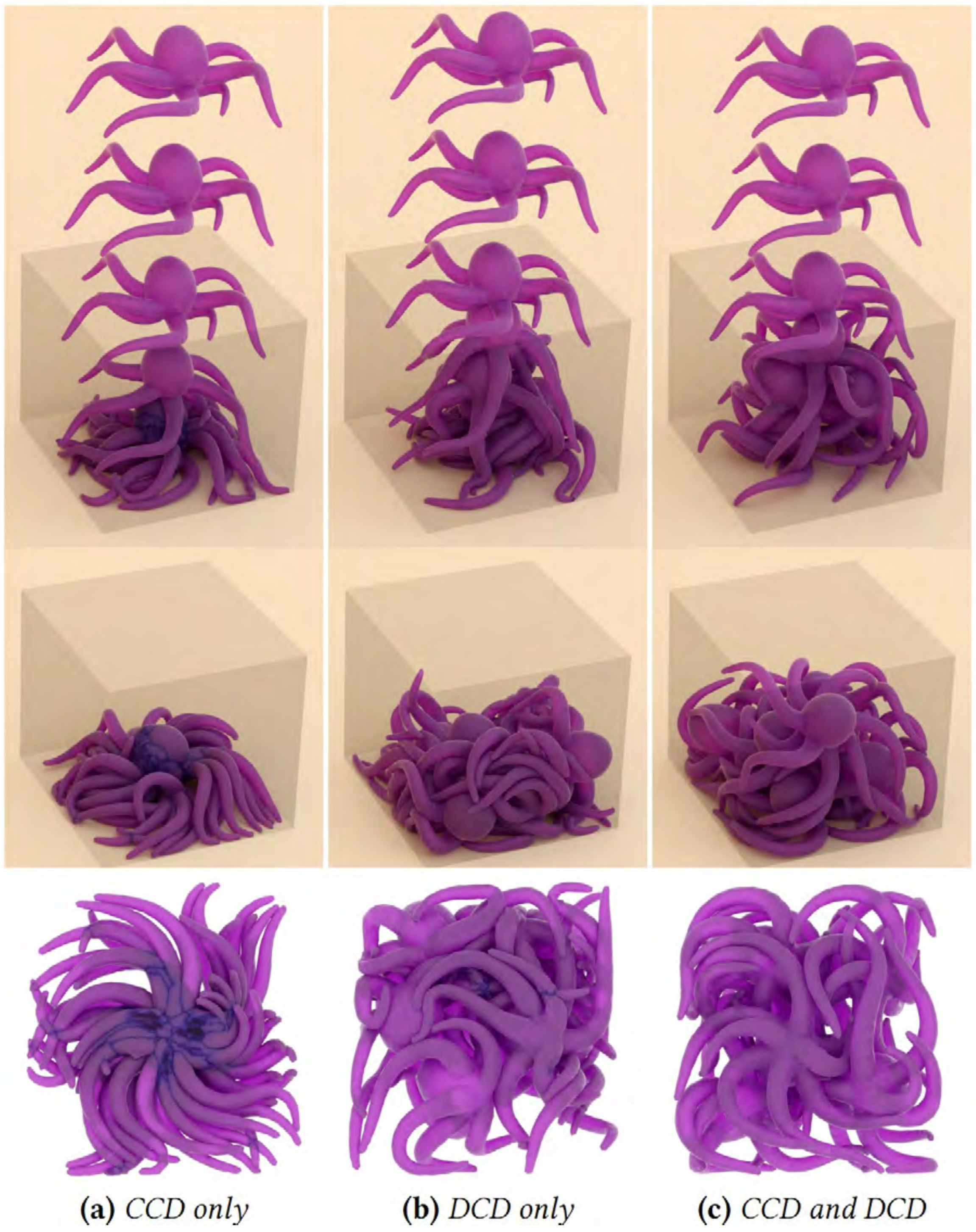


Figure 4.8: Dropping 8 octopi to a box simulated with (a) CCD only, (b) DCD with our shortest path query only, and (c) CCD and DCD with our shortest path query. The bottom row shows the bottom view of the final state. The blue tint highlights the intersecting geometry. The octopus model is from [188].

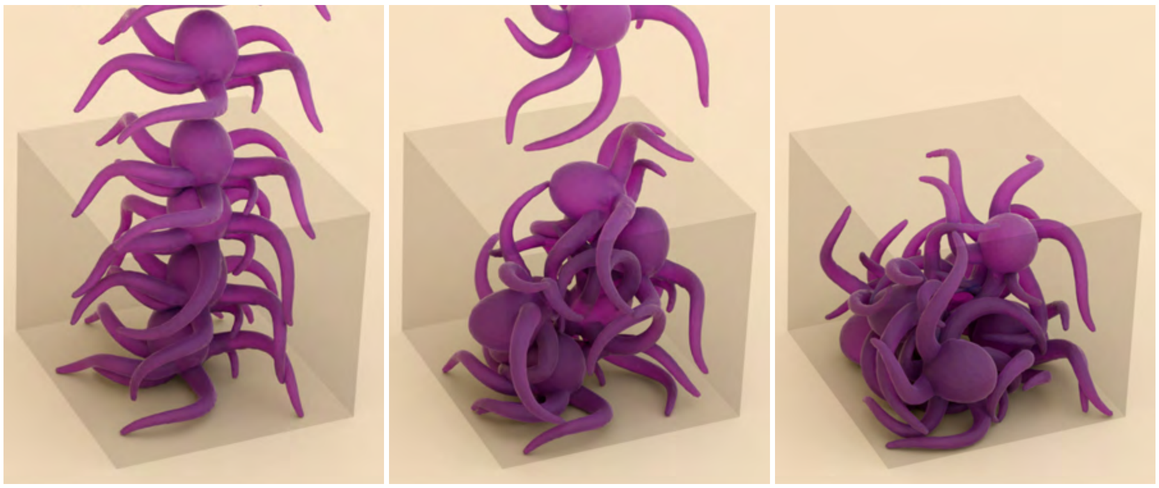


Figure 4.9: Dropping 6 octopi into a box simulated using implicit Euler. This simulation contains 30K vertices and 88K tetrahedra and it takes an average of 15s to simulate each frame.

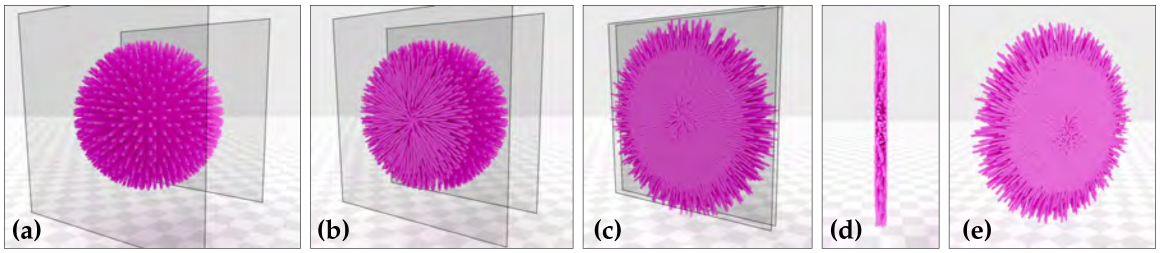


Figure 4.10: Flattening a squishy ball (774K vertices, 2.81M tetrahedra) using two planes. (a-c) the flattening process, (d) Side view of the flattened ball to 1/20 of its radius, and (e) the other side of the flattened squishy ball.

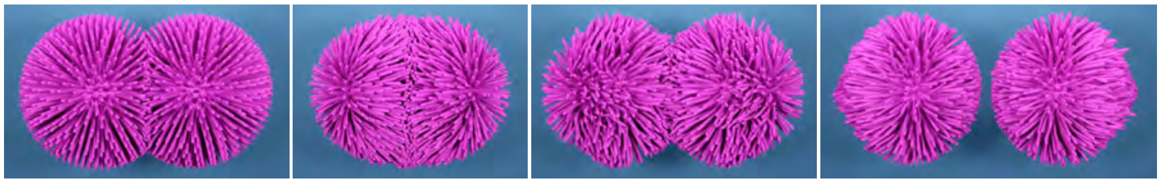


Figure 4.11: Simulation of two squishy balls in head-on collision that come to contact at a relative speed of $50m/s$. Both self-collisions and collisions between the two squishy balls are handled using our method.

Unlike CCD alone, DCD with our method significantly improves the robustness of collision handling when using a simulation system like XPBD that does not guarantee resolving all collision constraints. This is demonstrated in Figure 4.8, comparing different collision detection approaches with XPBD. Using only CCD leads to missed collisions when XPBD fails to resolve the collisions detected in previous steps, because CCD can no longer detect them. This quickly results in objects completely penetrating through each other (Figure 4.8a). Our method with only DCD effectively resolves the majority of collisions (Figure 4.8b), but it inherits the limitations of DCD. More specifically, using only DCD with sufficiently large time steps and fast enough motion, some collisions can be missed and deep penetrations can resolve the collisions by moving the objects in incorrect directions, again resulting in object parts passing through each other. Furthermore, our method only provides the closest path to the boundary and properly resolving the collisions is left to the simulation system. Unfortunately, XPBD cannot provide any guarantees in collision resolution, so detected penetrations may remain unresolved.

We recommend a hybrid solution that uses both CCD and DCD with our method. This hybrid solution performs DCD in the beginning of the time step to identify the preexisting penetrations or collisions that were not properly resolved in the previous time step. The rest of the collisions are detected by CCD without requiring our method to find the closest surface point. The same simulation with this hybrid approach is shown in Figure 4.8c. Since all penetrations are first detected by CCD and proper collision constraints are applied immediately, deep penetrations become much less likely even with large time steps and fast motion. Yet, this provides no theoretical guarantees. The addition of CCD allows the simulation system to apply collision constraints immediately, before the penetrations become deep, and DCD with our method allows it to continue applying collision constraints when it fails to resolve the initial collision constraints. Note that, while this significantly reduces the likelihood of failed collisions, they can still occur if the simulation system keeps failing to resolve the detected collisions.

The collision handling application of our method is not exclusive to PBD. Our method can also be used with force-based simulation techniques for defining a penalty force with penetration potential energy

$$E_c = \frac{1}{2} k \left((\mathbf{p} - \mathbf{s}) \cdot \mathbf{n} \right)^2 \tag{4.7}$$

where k is the collision stiffness.

An example of this is shown in Figure 4.9.

4.3 Results

We use XPBD [38] to evaluate our method, because it is one of the fastest simulation methods for deformable objects, providing a good baseline for demonstrating the minor computation overhead introduced by our method. We use mass-spring or NeoHookean [190] material constraints implemented on the GPU. We handle the collision detection and handling part on the CPU, including the position updates of the collision constraints. We use the hybrid collision detection approach that combines CCD and DCD, as explained above.

We implement both collision detection and closest point query on CPU using Intel's Embree framework [183] to create BVH.

We generate our timing results on a computer with an AMD Ryzen 5950X CPU, an NVIDIA RTX 3080 Ti GPU, and 64GB of RAM. We acknowledge that our timings are affected by the fact that we copy memory from GPU to CPU every iteration in order to do collision detection and handling, and the whole framework can be further accelerated by implementing the collision detection and shortest path querying on the GPU. As to the parameters of the algorithm, we set ϵ_r to -0.01. ϵ_i is related to the scale and unit of the object, when the object is at a scale of a few meters, we set ϵ_i to 1^{-10}.

4.3.1 Stress Tests

Figure 4.10 shows a squishy ball with thin tentacles compressed on two sides and flattened to a thickness that is only 1/20 of its original radius. Notice that all collisions, including self-intersections of tentacles, are properly resolved even under such extreme compression. Also. the model was able to revert to its original state after the the two planes compressing it were removed.

Figure 4.11 shows a high-speed head-on collision of two squishy balls. Though the tentacles initially get tangled with frictional-contact right after the collision, all collisions are properly resolved and the two squishy balls bounce back, as expected.

Figure 4.12 shows two challenging examples of self-collisions caused by twisting a thin beam and two elastic rods. Both instances have shown notable buckling after the twisting. A Such self-collisions are particularly challenging for prior self-collision handling methods that pre-split the model into pieces, since it is unclear where the self-collisions might occur before the simulation.

Another challenging self-collision case is shown in Figure 4.13, where nested knots were formed by pulling an elastic rod from both sides. In this case, there is a significant amount of sliding frictional self-contact, changing pairs of elements that collide with each other. Though a substantial amount of force is applied near the end, the simulation is able to form a stable knot.

Figure 4.14 shows the same experiment using naive closest boundary point computation for the collisions between the two squishy balls (by picking the closest boundary point on the other object purely based on Euclidean distances), only handling self-collisions with our method. Notice that it includes (temporarily) entangled tentacles between the two squishy balls and visibly more deformations of tentacles elsewhere, as compared to using our method (Figure 4.14). This is because, in the presence of self-collisions, naively handling closest boundary point queries between different objects is prone to picking incorrect boundary points that do not resolve the collision, resulting in prolonged contact and inter-locking.

4.3.2 Solving Existing Intersections

Our method can successfully resolve existing self collisions. A demonstration of this is provided in Figure 4.15. In this example, the initial state (Figure 4.15a) is generated by dropping a noodle model without handling self-collisions. When we turn on self-collisions, all existing self-intersections are quickly resolved within 10 substeps (Figure 4.15b), resulting in numerous inverted elements due to strong collision constraints. Then, the simulation resolves them (Figure 4.15c) and finally the model comes to a rest with self-contact (Figure 4.15d). In this experiment, we perform collision projections for all vertices (not only for surface vertices) and all tetrahedra's centroids to resolve the intersection in the completely overlapping parts. Note that we do not provide a theoretical guarantee to resolve all the existing intersections. In practice, however, in all our tests all collisions are resolved after just a few iterations/substeps.

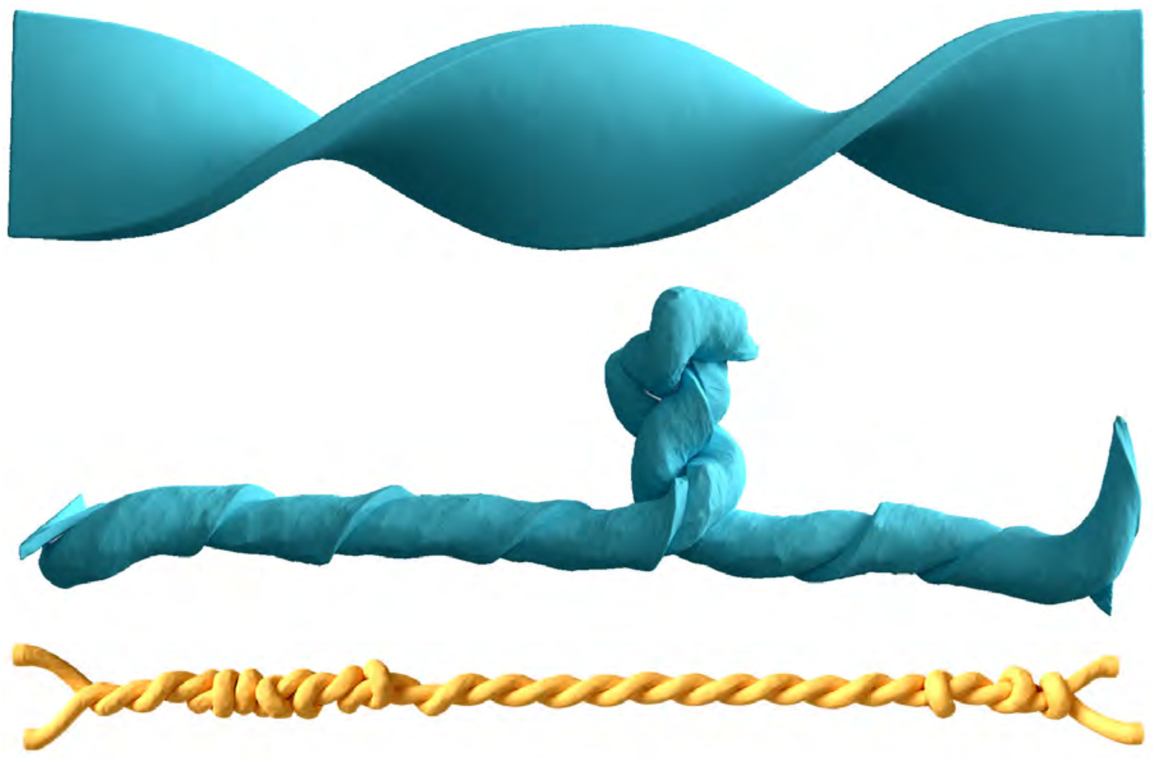


Figure 4.12: Twisting (top-middle) a thin beam with 400K vertices & 1.9M tetrahedra, and (bottom) two elastic rods with 281K vertices & 1.3M tetrahedra, demonstrating unpredictable self-collisions and buckling.

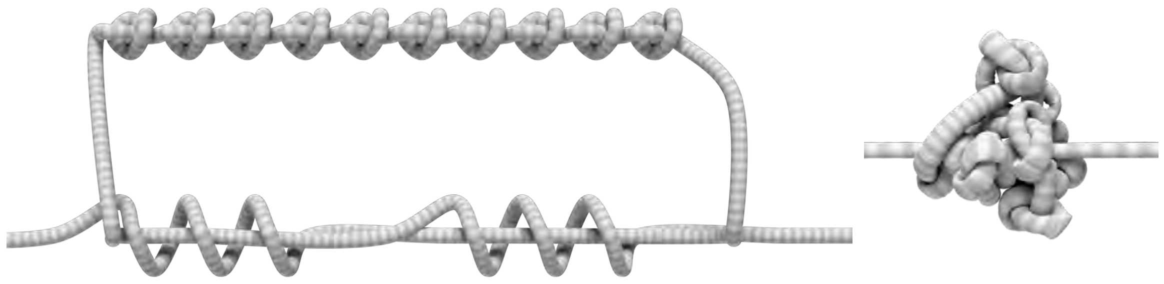
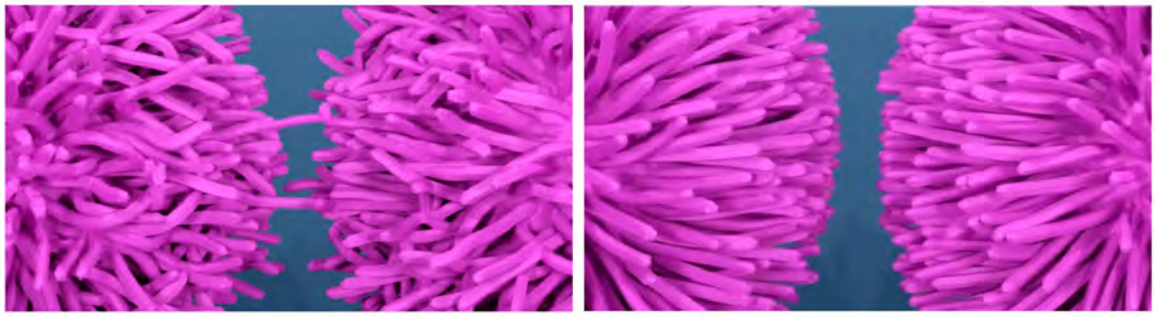


Figure 4.13: Simulation of a nested knot starting from (left) the initial state to (right) the final knot.



(a) *Naive collisions between objects* **(b)** *Our inter-object collisions*

Figure 4.14: Two squishy balls in head-on collision comparing collision handling between two different objects (a) by naively accepting the Euclidean closest point as the closest boundary point and (b) with our method. All self-collisions are handled using our method in both cases.

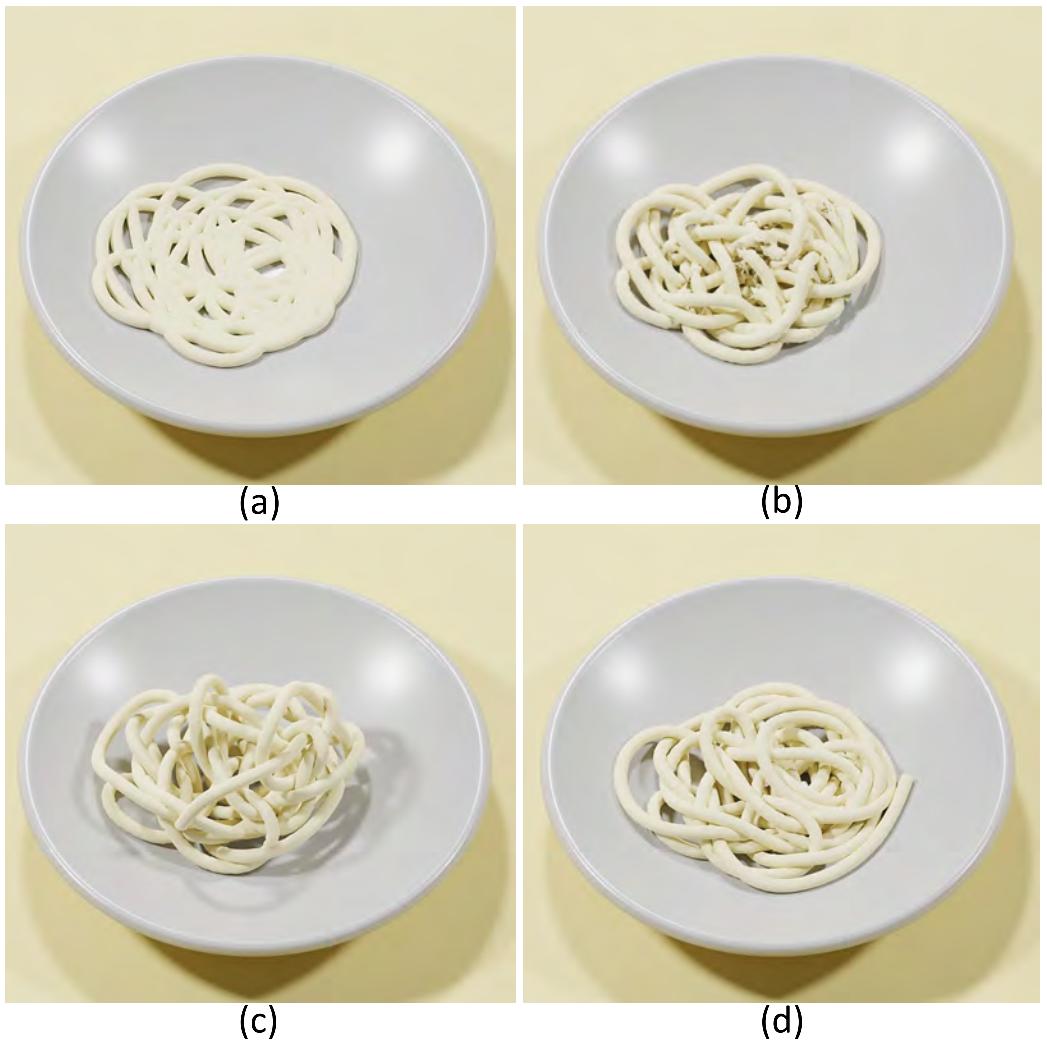


Figure 4.15: Solving existing collisions. Starting from (a) an initial state with many self-collisions, (b) after collision handling is enabled, (c) our method can quickly resolve them, and (d) achieve a self-collision-free final state.

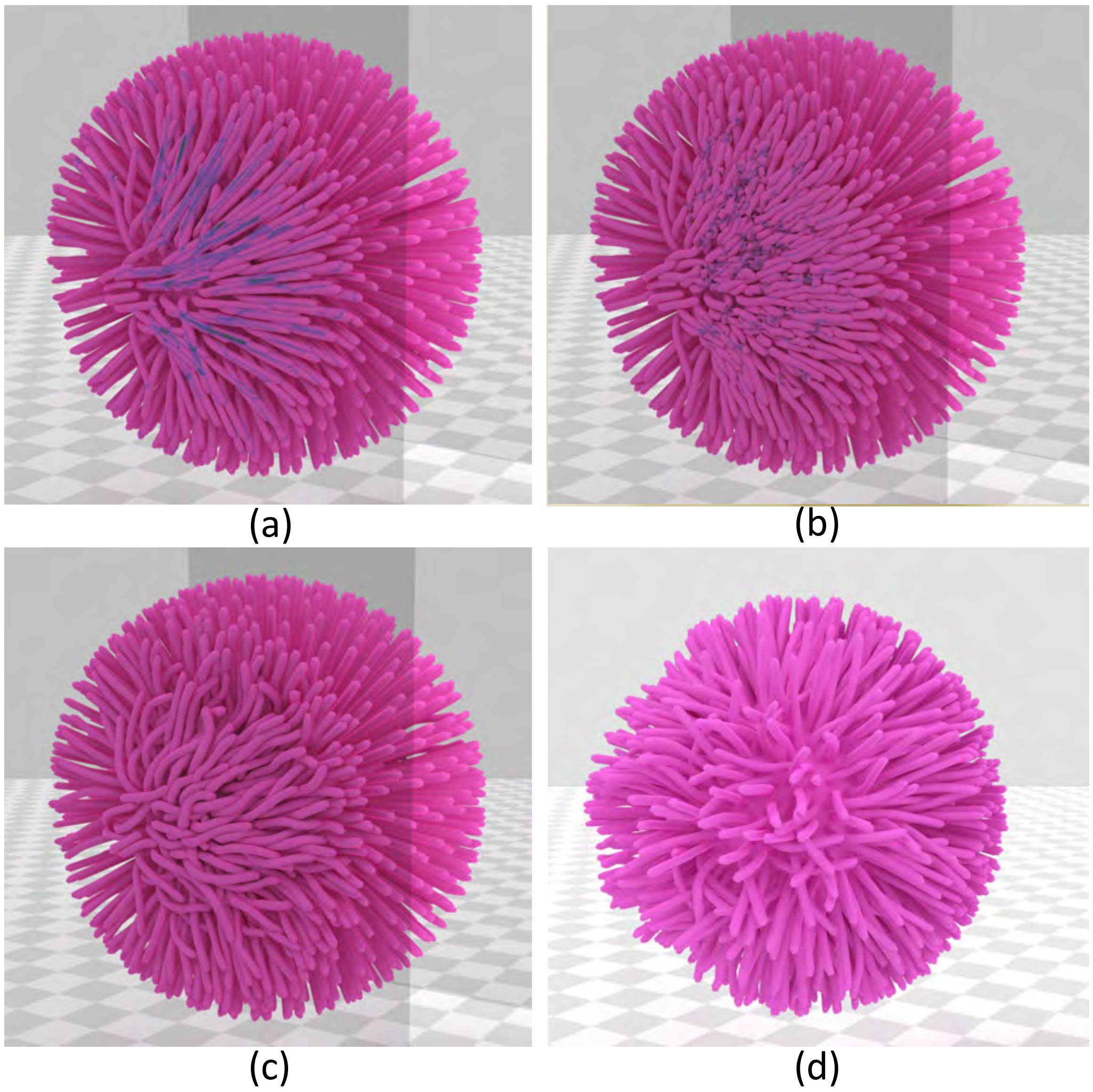


Figure 4.16: Our method simulates a squishy ball with (a) an initial state with complicated self-intersections where the intersecting part was colored blue. Within a few substeps (b,c), our method is able to resolve self-intersections back to a penetration-free state (c). Then we release the compressing plane constraint, and all the tentacles are able to bounce back without any self-intersections, as shown in (d).

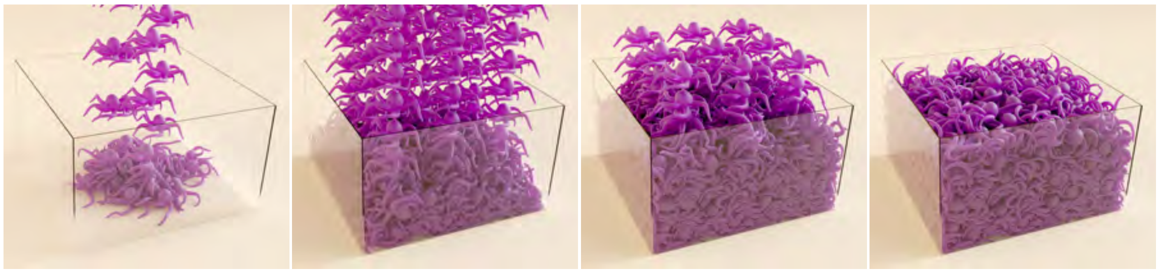


Figure 4.17: 600 deformable octopus models (3.1M vertices and 8.88M tetrahedra in total) dropped into a container, forming a pile with collisions.

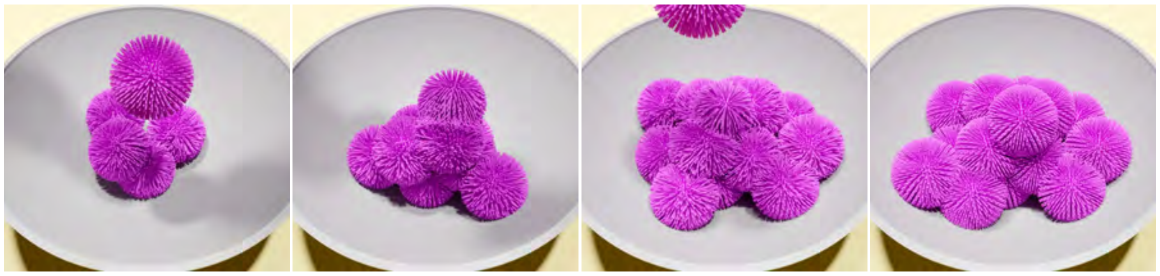


Figure 4.18: Simulation of 16 squishy balls (a total of 11.2 million tetrahedra) dropped into a bowl, forming a stable pile with active collisions.

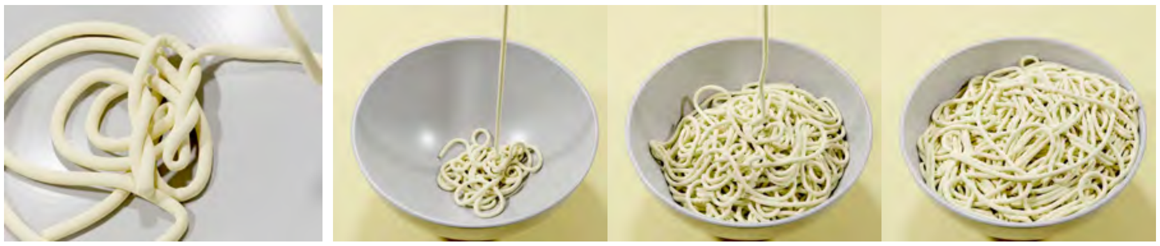


Figure 4.19: Simulation of a long noodle presenting unpredictable complex self-collisions and forming a large pile with self-collisions.

Another example is shown in Figure 4.16, generated by compressing a squishy ball with two planes on either side, similar to Figure 4.10 but without handling self-collisions. This results in a significant number of complex unresolved self-collisions (Figure 4.16a), which are quickly resolved within a few substeps when self-collision handling is turned on (Figure 4.16b, c, d).

4.3.3 Large-Scale Experiments

An important advantage of our method is that, by providing a robust collision handling solution, we can use fast simulation techniques for scenarios involving a large number of objects and complex collisions.

An example of this is demonstrated in Figure 4.17, showing 600 deformable octopus models forming a pile. Due to its complex geometry, the octopus model can cause numerous self-collisions and inter-object collisions. Both collision types are handled using our method. At the end of the simulation, a stable pile is formed with 185K active collisions per time step.

Figure 4.18 shows another large-scale experiment involving 16 squishy balls. At the end of the simulation, the squishy balls form a stable pile and remain in rest-in-contact with active self-collisions (12K) and inter-object collisions (125K) between neighboring squishy balls.

We also include an experiment with a single long noodle piece in Figure 4.19 that is dropped into a bowl. This simulation forms numerous complex and unpredictable self-collisions (Figure 4.19a). At the end of the simulation, we achieve a stable pile with 104K active self-collisions per time step in this example.

4.3.4 Performance

We provide the performance numbers for the experiments above in Table 4.1. Notice that, even though we are using a highly efficient material solver that is parallelized on the GPU, our method provides a relatively small overhead. This includes some highly-challenging experiments, involving a large number of complex collisions. The highest overhead of our method is in experiments in which deliberately disabled self-collisions to form a large number of complex self-collisions. Note that all collision detection and handling computations are performed on the CPU, and a GPU implementation would

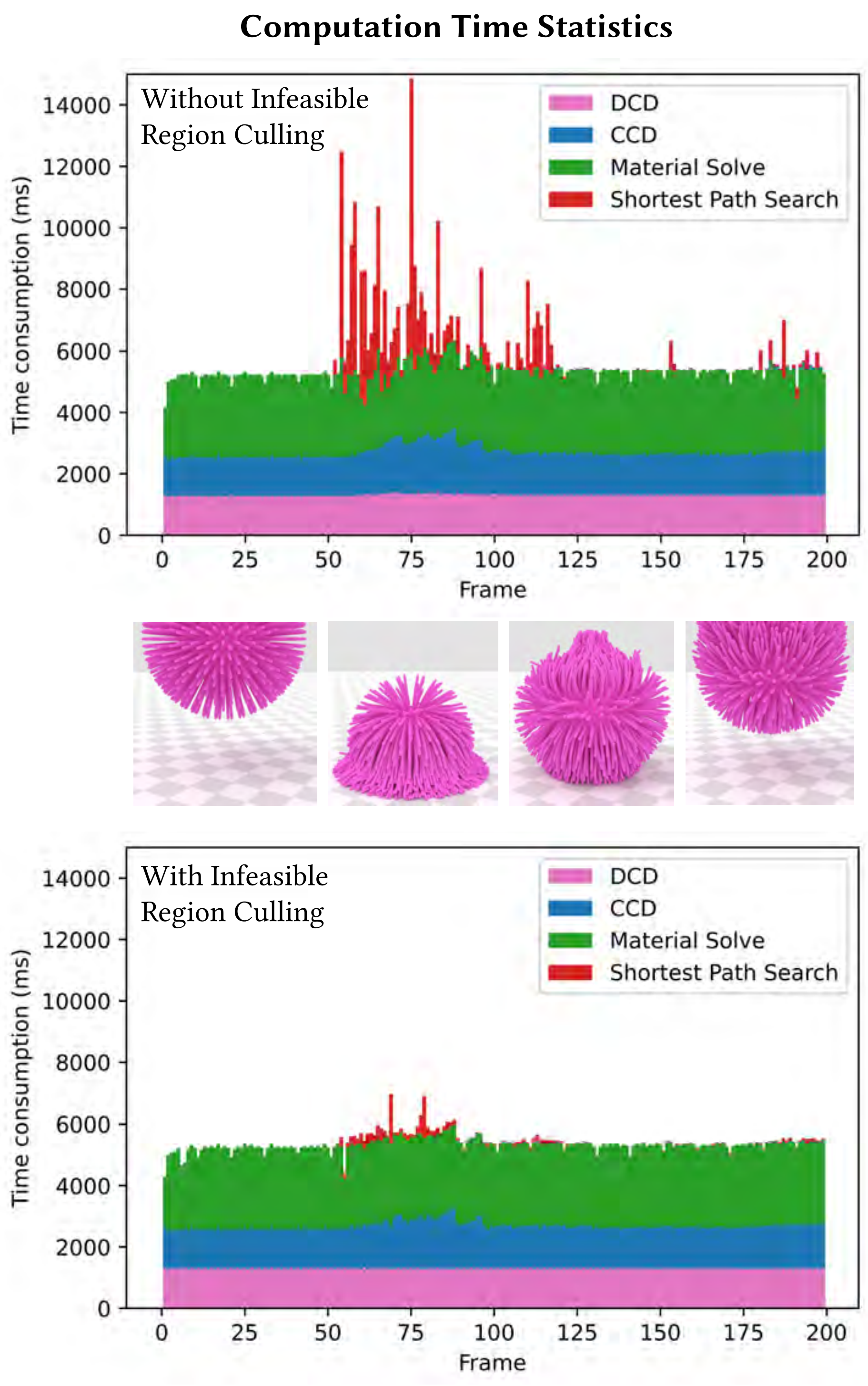


Figure 4.20: The computation time statistics of each simulation component on stacked bar charts: (top) without infeasible region culling and (bottom) with infeasible region culling. The middle row shows the simulation at frames 0, 75, 125, 175 respectively.

likely result in a smaller overhead.

We demonstrate the effect of our infeasible region culling by simulating a squishy ball dropped to the ground with and without this acceleration. The computation time breakdown of all frames are visualized in Figure 4.20. In this example, using our infeasible region culling, the shortest path query gains a speed-up of 10-30$\times$ for some frames, providing identical results. Additionally, the accelerated shortest path query results in a more uniform computation time, avoiding the peaks visible in the graph.

4.3.5 Comparisons to Rest Shape Shortest Paths

A popular approach in prior work for handling self-collisions is using the rest shape of the model that does not contain self-collisions for performing the shortest path queries. This makes the computation much simpler, but obviously results in incorrect shortest boundary paths. With sufficient deformations, these incorrect boundary paths can lead to large enough errors and instabilities.

Figure 4.21 shows a simple example, where a small cube is dropped onto a deformed object. Notice that the rest shape of the object (Figure 4.21a) is sufficiently different from the deformed shape (Figure 4.21b). With collision handling using this rest shape, the cube moves against gravity and eventually bounces back (Figure 4.21c), instead of sliding down the surface, as simulated using our method (Figure 4.21d). Figure 4.21e shows a 2D illustration of example shortest paths generated by both methods. Notice that using the rest shape results in a longer path to the surface that corresponds to higher collision energy. In contrast, our method minimizes the collision energy by using the actual shortest path to the boundary.

Figure 4.22 shows a more complex example with self-collisions that is initially simulated using our method (Figure 4.12) until complex self-collisions are formed. When we switch to using the rest shape to find the boundary paths, the simulation explodes following a number of incorrectly-handled self-collisions.

In general, using the rest shape not only generates incorrect shortest boundary paths, but also injects energy into the simulation. This is because an incorrect shortest boundary path is, by definition, longer than the actual shortest boundary path, thereby corresponds to higher potential energy.

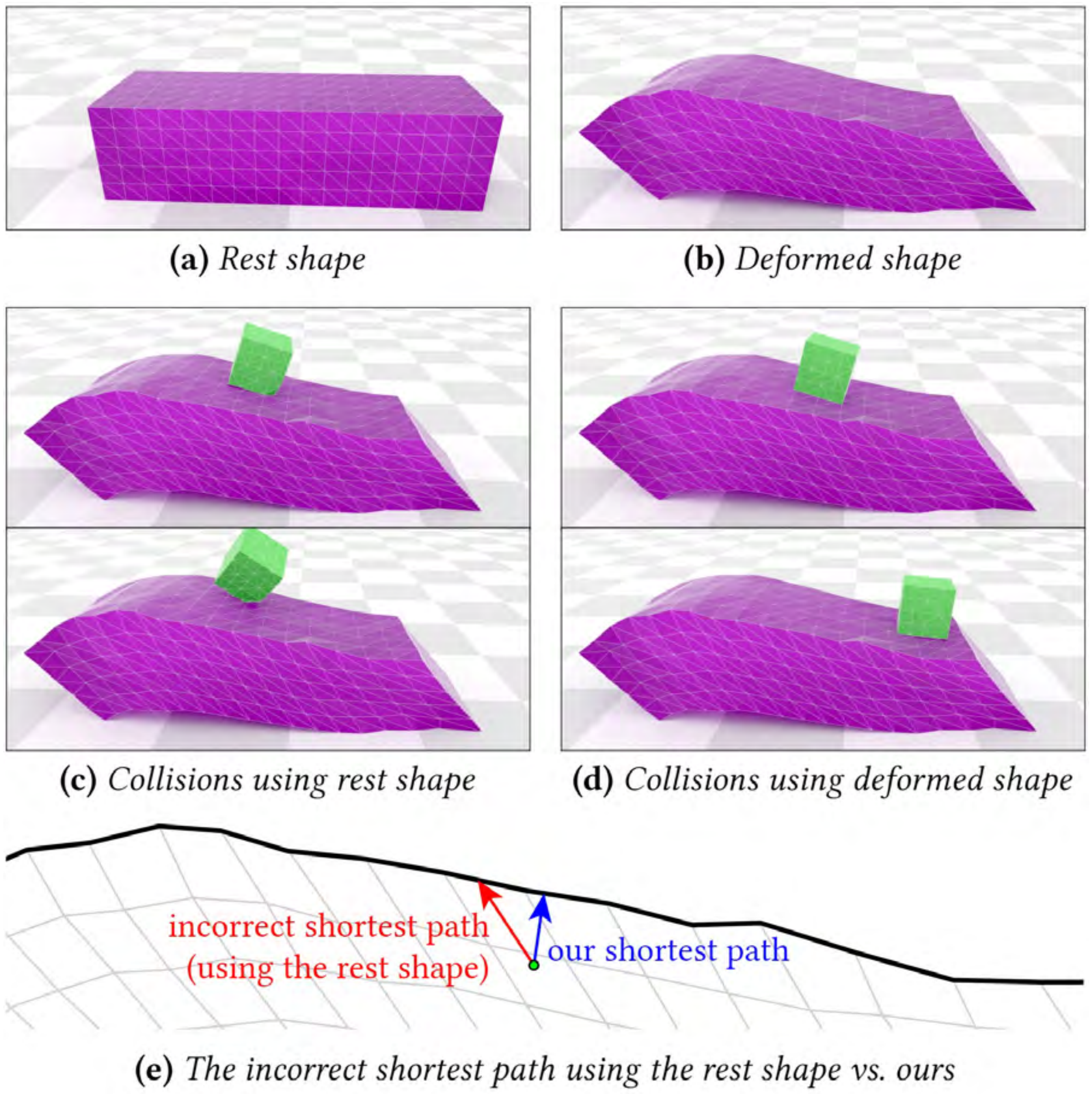
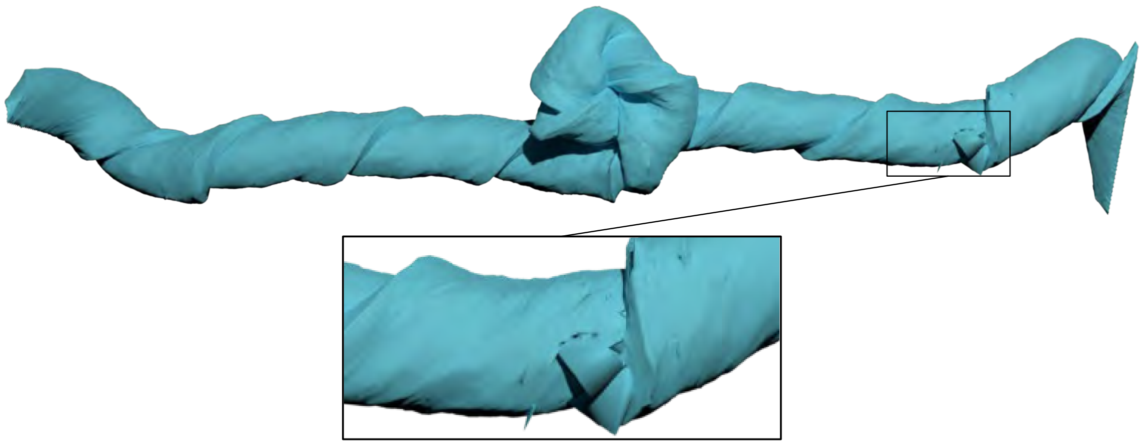
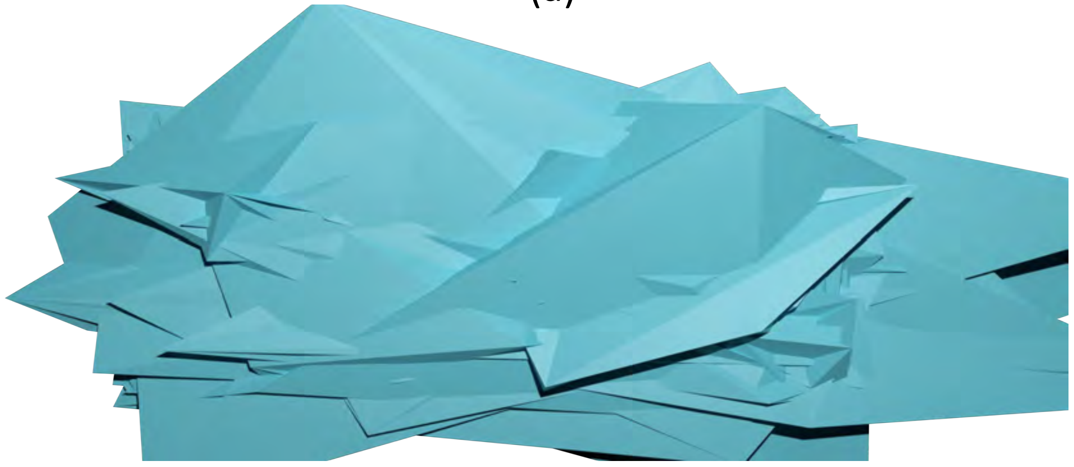

(a) Rest shape
(b) Deformed shape
(c) Collisions using rest shape
(d) Collisions using deformed shape
incorrect shortest path
(using the rest shape)
our shortest path
(e) The incorrect shortest path using the rest shape vs. ours

Figure 4.21: Comparison between the rest pose closest boundary point and our closest boundary point. (a) The rest pose the cuboid model. (b) We deform the cuboid to a certain shape, then drop a cube on top of it. (c) In the simulation using the rest pose closest boundary point, the cube got incorrectly pulled up. (d) Using our exact closest point, the cube successfully slides down. (e) The shortest path to the surface for an example point, showing that using the closest surface point queried from the rest shape results in an incorrect and longer path.



(a)



(b)

Figure 4.22: Simulation of twisting a thin beam, shown in Figure 4.12, soon after replacing our method with using the rest pose for finding the closest boundary point: (a) instabilities caused by incorrect closest boundary points found using this approach, and (b) exploded simulation after a few frames.

4.4 Discussion

An important advantage of our method is that it can work with simulation systems that do not provide any guarantees about resolving collisions. Therefore, we can use fast simulation techniques like XPBD to handle complex scenarios involving numerous self-collisions, as demonstrated above.

Yet, our method cannot handle all types of self-collisions and it requires a volumetric mesh. We cannot handle collisions of codimensional objects, such as cloth or strands. Our method would also have difficulties handling meshes with thin volumes or no interior elements.

Our method is essentially a shortest boundary path computation method. It is based on the fact that an interior point's shortest path to the boundary is always a line segment. This assumption always holds for objects like tetrahedral meshes in 3D or triangular mesh in 2D Euclidean space. Therefore, our method cannot handle shortest boundary paths in non-Euclidean spaces, such as geodesic paths on surfaces in 3D.

Using our method for collision handling with DCD inherits the limitations of DCD. For example, when with large time steps and sufficiently fast motion, penetration can get too deep, and the shortest boundary path may be on the other side of the penetrated model, causing undesirable collision handling. In practice, this problem can be efficiently solved by coupling CCD and DCD, as we demonstrate with our results above.

Table 4.1: Performance results. Time step size and frame times are given in seconds, where frame times are measured at 60 FPS. Operations Q., Tr., and Tet. represent the number of BVH queries, traversals, and total tetrahedra visited on average per time step, respectively.

		Number of		Avrg. Collisions		Avrg. Operations			Time Step	Frame Time		Average Time %			
		Vert.	Tet.	CCD	DCD	Q.	Tr.	Tet.	Size × Iter.	Avrg.	Max.	XPBD	CCD	DCD	**Ours**
Flattened Squishy Ball	(Fig. 4.10)	774 K	2.81 M	16.8 K	7.1 K	56	6.6	5.2	3.3e-4 × 3	10.89	18.04	30.9 %	29.0 %	31.7 %	**8.3 %**
Twisted Thin Beam	(Fig. 4.12)	400 K	1.9 M	8.3 K	3.1 K	45	5.7	7.2	3.3e-4 × 3	8.16	15.92	29.7 %	31.3 %	32.1 %	**6.9 %**
Twisted Rods	(Fig. 4.12)	281 K	1.3 M	4.8 K	2.6 K	33	5.3	4.7	3.3e-4 × 3	5.25	11.17	42.1 %	26.9 %	27.0 %	**4.0 %**
Nested Knots	(Fig. 4.13)	38.1 K	103 K	3.1 K	0.6 K	31	4.2	4.4	5.5e-4 × 3	0.25	0.32	61.8 %	23.3 %	9.5 %	**5.2 %**
2 Squishy Balls	(Fig. 4.11)	418 K	1.4 M	22.4 K	1.3 K	36	9.6	11.3	3.3e-4 × 3	1.96	2.87	52.2 %	28.4 %	18.5 %	**10.9 %**
Pre-Intersect. Noodle	(Fig. 4.15)	40 K	110 K	N/A	15.2 K	65	12.0	13.6	8.3e-4 × 3	0.21	0.45	51.6 %	17.6 %	18.4 %	**12.4 %**
Pre-Intersect. Squishy Ball	(Fig. 4.16)	219 K	704 K	N/A	45.8 K	89	12.0	14.0	3.3e-4 × 3	1.54	2.63	44.3 %	18.2 %	19.1 %	**18.4 %**
600 Octopi	(Fig. 4.17)	3.1 M	8.88 M	104.0 K	6.4 K	12	3.6	4.1	8.3e-4 × 3	16.40	17.90	68.3 %	15.4 %	13.4 %	**2.9 %**
16 Squishy Balls	(Fig. 4.18)	3.5 M	11.2 M	118.5 K	8.5 K	29	4.5	6.6	3.3e-4 × 3	18.50	20.20	49.3 %	25.0 %	21.8 %	**3.9 %**
Long Noodle	(Fig. 4.19)	860 K	2.29 M	102.6 K	6.1 K	11	3.6	3.2	8.3e-4 × 3	4.10	4.50	67.6 %	14.8 %	14.9 %	**2.7 %**
8 Octopi CCD Only	(Fig. 4.8a)	40 K	118 K	2.1 K	N/A	N/A	N/A	N/A	3.3e-3 × 5	0.028	0.036	86.6 %	13.4 %	N/A	N/A
8 Octopi DCD Only	(Fig. 4.8b)	40 K	118 K	N/A	2.4 K	13	3.7	4.1	3.3e-3 × 5	0.038	0.045	79.2 %	N/A	10.7 %	**10.1 %**
8 Octopi hybrid	(Fig. 4.8c)	40 K	118 K	2.3 K	0.2 K	11	3.3	3.9	3.3e-3 × 5	0.035	0.038	79.3 %	10.1 %	9.6 %	**1.0 %**

CHAPTER 5

OFFSET GEOMETRY CONTACT

In this section, we propose a collision energy E_n hat ensures the derived contact forces are always orthogonal to the surface. This energy can be incorporated into Equation 2.8 to effectively handle collisions between codimensional objects. The orthogonality of the contact force allows for a larger contact radius, thereby mitigating the stiffness problem associated with collision energy. We also propose a constraint handling scheme for solving the penetration-free constraint in Equation 2.12, thereby providing a penetration-free guarantee for the whole simulation process.

5.1 Contact Force

Contact occurs between two surfaces. For each point on one surface that contacts another surface, it is subjected to a contact force from the opposing surface. This force generally consists of two components: a normal force, which acts perpendicular to the contact surface, and a friction force, which acts parallel to the contact surface and is linearly related to the normal force. The normal force is a conservative force, while the friction force is not. Therefore we can write normal force as the negative gradient of a *normal contact energy* E_n. The formulation of E_n differentiates different contact models.

5.1.1 Basic Contact Model

In physics-based simulation, surfaces are represented by polygonal mesh, denoted by $M = \{\mathcal{V}, \mathcal{E}, \mathcal{T}\}$, where $\mathcal{V}, \mathcal{E}, \mathcal{T}$ denote the set of vertices (0-face), edges (1-face) and facets (2-face, e.g., triangles, quadrilaterals, etc.), respectively. It is important to note that M can consist of multiple connected components, which accommodates the presence of multiple models. Therefore, without loss of generality, in the following discussion, we assume the presence of a single mesh M. We denote $X \in \mathbb{R}^{K\times 3}$ as the stacked positions of all the vertices, where $K = |\mathcal{V}|$. The position of vertex v is represented as $\mathbf{x}_v$.

We define the normal contact energy as a function of the distance between two primitives, namely, between a vertex and a facet or between two edges. Based on the first law of friction, the contact force can be computed in such order: computing the normal force first, and then calculating the friction force using the friction coefficient. Therefore, the normal contact force plays a key role in the computation of contact force.

We start with the vertex-facet contact pair. Given a mesh M, the normal contact energy E_n^{vf} of M is usually defined in the following form:

$$E_n^{vf}(M,r) = \sum_{a \in \mathcal{F}(v)} g(dis(\mathbf{x}_v, a), r) \ , \tag{5.1}$$

where $\mathcal{F}(v)$ is the set of all the faces that is in contact with v, $dis(\mathbf{x}_v, a)$ is the function computing the distance between vertex $\mathbf{x}_v$ position and a face a, r is contact radius, and g is a nonlinear function. We define the closest point from $\mathbf{x}_v$ to a as:

$$\mathbf{c}(\mathbf{x}_v, a) = \arg\min_{\mathbf{x} \in a} ||\mathbf{x} - \mathbf{x}_v||. \tag{5.2}$$

Therefore $dis(\mathbf{x}, a) = ||\mathbf{x} - \mathbf{c}(\mathbf{x}, a)||$. Since g is just a scalar function, it does not change the direction of the force. Therefore, the contact force between f and $\mathbf{x}$, is always parallel to $\mathbf{x} - \mathbf{c}(\mathbf{x}, a)$.

The different choices of g and $\mathcal{F}(v)$ result in different collision energies. We start with discussing the choice of $\mathcal{F}(v)$, termed *contact face set*.

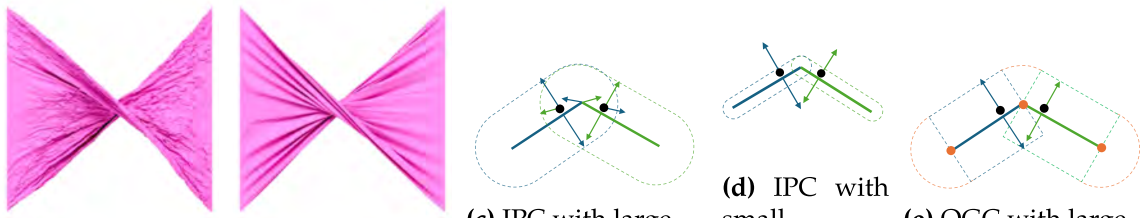
5.1.2 Contact Face Set

One common choice of $\mathcal{F}$ is:

$$\mathcal{F}_{\text{IPC}}(v) = \{t \in \mathcal{T} | dis(\mathbf{x}_v, t) < r, v \not\subset t\} \ . \tag{5.3}$$

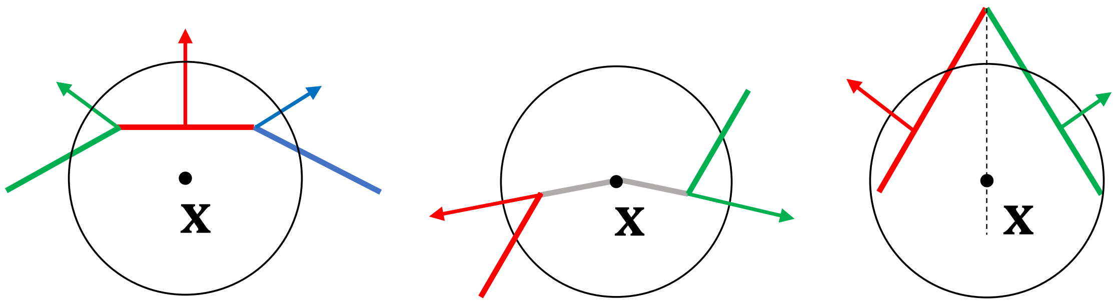
Namely, $\mathcal{F}_{\text{IPC}}(v)$ takes all the facets who does not include v and whose distance to v is less than the contact radius r. This contact face set is employed by the well-known Incremental Potential Contact [2]. Intuitively, this formulation is like inflating the facets in all directions, forming volumetric shapes as illustrated in Figure 5.1c. A facet contact with v if v is located inside the inflated facets.

There are two major problems with this energy formulation. The first one is illustrated in Figure 5.1c and Figure 5.2a: the "normal" contact force applied on the green and blue facets is not perpendicular to them, causing a stretching force component on the tangential



(a) IPC **(b)** OGC **(c)** IPC with large contact radius **(d)** IPC with small contact radius **(e)** OGC with large contact radius

Figure 5.1: Illustrating both the artifact produced by the IPC contact model and its underlying cause. We simulate a twisted square cloth with 40K vertices and 79.2K faces, each side measuring 1 meter, rotated by half a circle. The simulation is conducted using both the IPC and OGC models, with a fixed contact radius of 5 mm. (a) and (b) show the final states of the cloth using the IPC and our OGC contact models, respectively. Panels (c) and (d) depict the IPC contact model, which is equivalent to offsets the face in all directions to form a capsule-like shape. The dashed line marks the boundary of this shape, black dots represent contact points, and the colored arrows indicate the forces exerted from or onto the face with corresponding color. Panel (e) visualizes our proposed contact model, where the dashed lines marks the boundaries of blocks from corresponding faces with the same color.



(a) $\mathcal{F}_{\text{IPC}}$ **(b)** $\mathcal{F}_{\text{IPC}}$ **(c)** $\mathcal{F}_{\text{SDF}}$

Figure 5.2: 2D illustration of different contact face sets and the normal contact force derived from them. $\mathbf{x}$ is the position of the vertex of the vertex-facet (v-f) contact pair, and the black circle visualizes the contact radius of point $\mathbf{x}$. The colored line segments represent facets, and the colored arrows represent the normal contact force applied to the facets of the same color. (a, b) visualize $\mathcal{F}_{\text{IPC}}$. (c) visualizes $\mathcal{F}_{\text{SDF}}$, where the dashed black line represents the bisector of those two facets.

plane. Another problem is that it pushes $\mathbf{x}$'s topological neighbors away, as illustrated in Figure 5.2b. While it is possible to ignore the contact between v and its neighboring facets, the problem still exists between a vertex and its 2-ring neighbors. Unfortunately, we cannot filter out these contacts as this can cause penetration. As illustrated in Figure 5.1a, when a large contact radius is used, those problems can cause serious artifact including stretching and oscillating.

IPC mitigates those problems by dynamically adjusting the contact radius to a very small value (Figure 5.1d), to the extent where it is nearly impossible for a vertex to be in contact with multiple adjacent facets. However, choosing a small contact radius leads to other problems including numerical issues such as the stiffness of the contact energy. Additionally, IPC's CCD-aware line search to avoid penetration limits the optimization step size when v is close to the contacting surface, as smaller distances trigger earlier penetration. Moreover, since the contact radius is so small, the surface region that stops v may not be in contact with v when computing the optimization direction. Therefore, the optimization direction may not be a direction that separates those colliding pairs, which further restricts its movement per iteration. These factors contribute to IPC's high iteration count for convergence.

An alternative selection for $\mathcal{F}(\mathbf{x})$ is to select only the closest facet:

$$\mathcal{F}_{\text{SDF}}(v) = \{t | t = \underset{t \in \mathcal{T}}{\arg\min}\, dis(\mathbf{x}_v, t) \text{ and } dis(\mathbf{x}_v, t) < r\} \tag{5.4}$$

This contact face set is the basis of many signed distance field (SDF) based collision energy. The main advantage of this approach is that it guarantees the normal force is always perpendicular to the contacting point on M, a natural property of the closest point on a smooth manifold as established by the Hilbert Projection Theorem.

However, it has some serious problems due to its limiting the number of a vertex's contacting facets to only one. This restriction could impede the convergence of the problem because it fails to generate a sufficient number of collision pairs to push away vertex-facet pairs that are close enough. Instead, it results in the vertex constantly switching with a few facets. This is illustrated in Figure 5.2c, where the point $\mathbf{x}$ alternates between contacting the red facet and the green facet, oscillating along their bisector. To make matters worse, this formulation can not resolve self-intersection. Since v is part of M, the SDF at $\mathbf{x}_v$ is 0, and this information will not help resolve v's contact with M. That is why this contact face set is usually used to handle contact of static objects.

We propose a new normal contact force model that has the following properties:

- **Orthogonality**: our normal contact force is always orthogonal to the contact surface. It will not create a stretching artifact even with a large contact radius.
- **Large Contact Radius**: The contact radius can be arbitrarily large and still not cause artifacts.
- **Multiple Contacts within Contact Radius**: multiple primitives within the contact radius can affect $\mathbf{x}$, which allows repulsive force to be generated for an arbitrary number of close-by primitive pairs.
- **Self-Intersection Aware**: this contact force can identify arbitrary layers of self-intersection, and effectively resolve them.

Our normal contact force is derived from an offset geometry of the original mesh, hence the name Offset Geometric Contact (OGC). We construct the building blocks of this offset geometry by offsetting a face along all its normal directions, which is given by its Polyhedral Gauss Map (Section 5.1.3). We further provide constructive definitions of those building blocks to determine whether a point is inside (Section 5.1.4, 5.1.7), and define our own contact face set (Section 5.1.5). Subsequently, we derive the penetration in the offset geometry (Section 5.1.6) and introduce a new activation function to formulate our normal contact energy (Section 5.1.8. At last we propose our approach to guarantee penetration-free simulation (Section 5.1.9), and compare it to IPC's method (Section 5.1.10).

5.1.3 Polyhedral Gauss Map

We build the offset geometry using a way akin to tetmesh: offset each face individually and use them as the building blocks of the offset geometry. From the previous discussion, it is evident that the normal contact force is always parallel to $\mathbf{x}_v - \mathbf{c}(\mathbf{x}_v, a)$, see Equation 5.1. Since our goal is to achieve an orthogonal normal contact force, intuitively, for any face a we can design its building block such that it contacts only points that generate contact forces parallel to the normal at the contact point. In other words, it should only contact points $\mathbf{x} \in \mathbb{R}^N$ that satisfies $\mathbf{x} - \mathbf{c}(\mathbf{x}, a)$ being parallel to the normal of $\mathbf{c}(\mathbf{x}, a)$.

On a smooth surface, enforcing such a condition is relatively straightforward because each point on the surface has a unique normal. However, on a polyhedral surface, normals are not so trivially defined, introducing additional complexity. These considerations naturally lead to the concept of the Polyhedral Gauss Map (PGM).

Polyhedral Gauss Map is an analogy of a Gauss Map on a polyhedral surface, mapping a point on a polyhedral surface to their associated *normals*. The key difference is that a point on a polyhedral surface can have multiple normals, as opposed to those on smooth surfaces that only have one. The points that have more than one normal locates on faces with dimensionality less than 2, e.g. vertices and edges of a 3D polyhedral mesh.

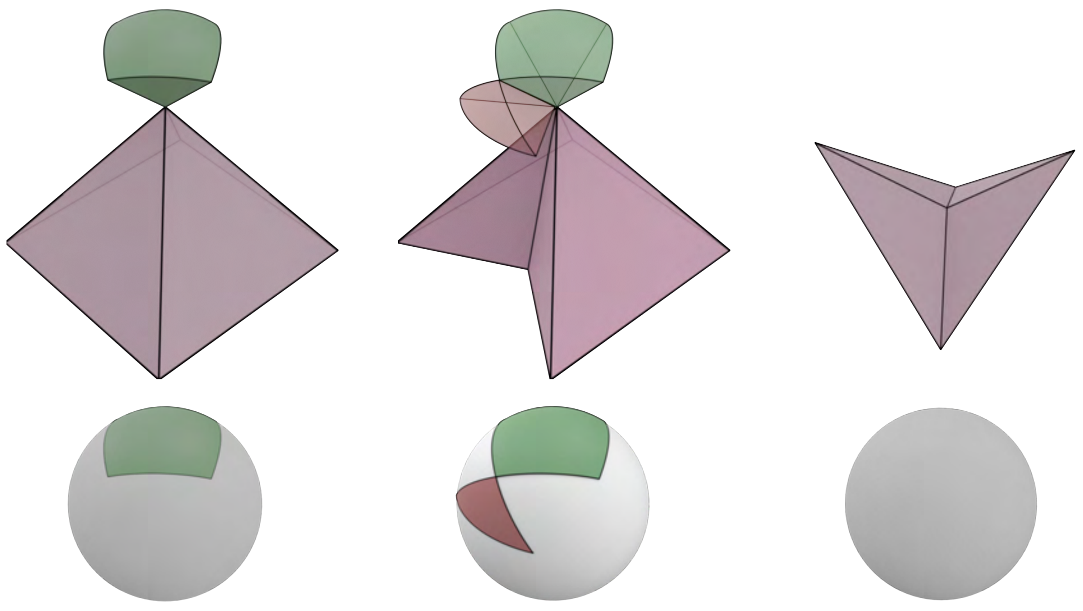

(a) Convex Vertex **(b)** Mixed Vertex **(c)** Saddle Vertex

Figure 5.3: Gauss map of different types of vertices (top row) and their spherical indicatrix (bottom row). The area with the pink color represents the local geometry of the triangular mesh, and the solid green area represents the normals where the point maps to.

[94] proposed a form of Polyhedral Gauss Map for vertices on a polyhedral mesh. Since all the normals are unit vectors, we can draw them on a unit sphere. As illustrated in Figure 5.3, [94] classifies vertices into three types: convex, mixed, and saddle, based on the geometry of their neighborhood which are visualized using the pink surface. Following the terminology of [94]), the neighbor area is called $star(v)$.

As the name indicates, a convex vertex v is one whose neighborhood is convex. The Gauss Map of a convex vertex is relatively straightforward, as shown in Figure 5.3a as the green volume. Intuitively, this volume corresponds to the set of points that are closer to v than any other points in $star(v)$.

A mixed vertex v is one that remains a vertex of the convex hull of $star(v)$. As shown in Figure 5.3b, the Gauss Map of a mixed vertex consists of two distinct types of regions. one corresponds to the positive curvature, as visualized by the green volume, which corresponds to its Gauss map as a vertex of the convex hull of $star(v)$. The red volume corresponds to regions of negative curvature, with each negative segment associated with a non-convex neighboring edge.

The final type is the saddle vertex, which lies within the interior of the convex hull of $star(v)$. The Gauss Map of a saddle vertex is an empty set because such vertices exhibit no angular deficit.

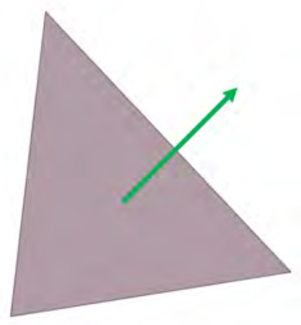
[94] provided an intuitive explanation of their proposed Gauss Map: plot all the normals of the neighboring facets of a vertex onto the unit sphere, resulting in a set of points. Connect these points in the counter-clockwise order of the neighboring facets, following great circles, to form a polygon on the unit sphere. In this representation:

- A neighboring facet corresponds to a vertex of the polygon.
- A neighboring edge corresponds to an arc-edge of the polygon, which is perpendicular to the neighboring edge.
- The vertex itself corresponds to the polygon as a whole.

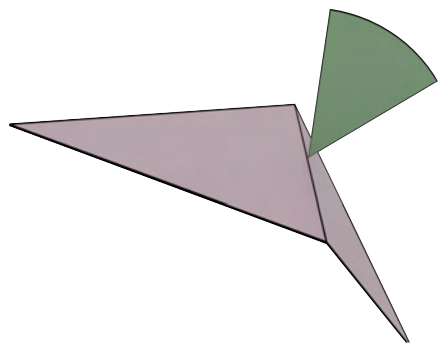
For a mixed vertex, due to its concavity, the polygon may contain inverted areas. These inverted areas represent regions of negative curvature.

The motivation for [94] to define the Polyhedral Gauss Map is to extend Gauss–Bonnet theorem to a polyhedral surface. That is why they only proposed the Gauss map of vertices because only vertices are integrated. However, in our case, we also need to define the Polyhedral Gauss Map of edges and facets. Since all points in the interior of a face share the same normal, we can instead define the Gauss Map at the level of faces. We denote the Gauss Map of a face a as $\mathcal{N}_a$.

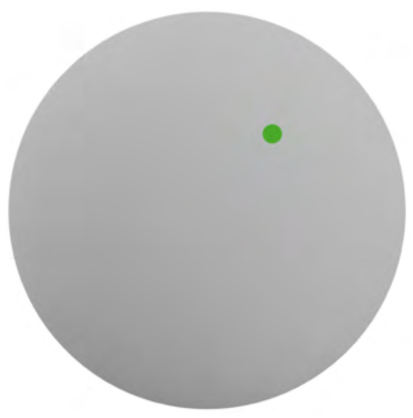
The Gauss Map for points on facets and edges is relatively straightforward. As shown in Figure 5.4c, all points on the interior of a facet map to a single point on the unit sphere, corresponding to the normal of that face, see Figure 5.4a. Conversely, a point on the interior of an edge corresponds to multiple normals, maps to an arc on the unit sphere



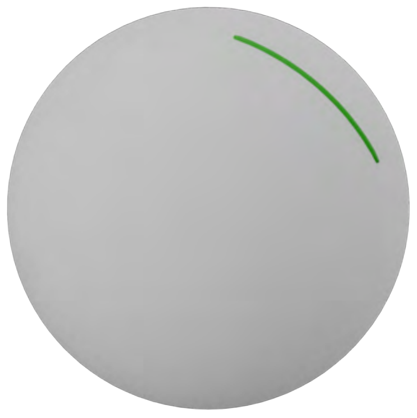
(a) Face Normals



(b) Edge Normals



(c) Face Normal Spherical Indicatrix



(d) Edge Normals Spherical Indicatrix

Figure 5.4: Illustration of Gauss Maps of a facet (triangle) and an edge. In the top row, the area with the pink color represents the local geometry of the triangular mesh, and the solid green area represents the normals where the point maps to. In the bottom row, we show the spherical indicatrix, i.e. visualize the corresponding point's normal on a unit sphere.

see Figure 5.4b and Figure 5.4d. This is because that the mesh is flats on all the triangles, and it ”turns” on edges, and the normals of each edge correspond to how much angle the surface turns.

For vertex Gauss maps, we adopt a slightly modified definition tailored to our use case in contact detection. Unlike [94], which defines the Gauss Map from a curvature point of view, we follow a discrete interpretation of the Hilbert Projection Theorem. Specifically, if $\mathbf{n}$ is the normal at point $\mathbf{y}$ on M, then for any point $\mathbf{x}$ satisfies $\mathbf{x} = w\mathbf{n} + \mathbf{y}, w > 0$, $\mathbf{y}$ must be closer to $\mathbf{x}$ than its local neighborhood. This perspective alters the Gauss Map of mixed vertices to only include the region corresponding to positive curvature, i.e., the green area in Figure 5.3b. This adjustment not only simplifies the computation but also ensures that when a vertex contacts a point, it is the locally closest point to that point.

We also make some specific treatments to a facet's t (e.g., triangles) Gauss Map, we define:

$$\mathcal{N}_t = \{\mathbf{n}_t, -\mathbf{n}_t\}, \tag{5.5}$$

where $\mathbf{n}_t$ is the normal of t. In other words, we offset a facet to both of its sides.

5.1.4 Constructive Definition of Blocks

Now we have clarified the definition of normals on a discrete surface. We can offset points on M along their normal directions to construct building blocks of the offset geometry.

For a face $a \in M$, we offset its *interior* points to construct the fundamental building blocks of the offset geometry:

$$U_a = \{\mathbf{x} \in \mathbb{R}^N | \mathbf{x} = \mathbf{y} + wr\mathbf{n}_a, \text{where } \mathbf{y} \in a^\circ, \mathbf{n}_a \in \mathcal{N}_a, w \in [0,1]\} \tag{5.6}$$

where $r > 0$ is the offset radius, and a° denote the interior of face a. For convenience we let $v^\circ = v$. We only offest the interior of a face because the boundary points are actually points on a lower dimensional face.

We refer to U_a as the *block* derived from face a, s it serves as a fundamental building block of the offset manifold. The definition provided in Equation 5.6 reflects the essence of these blocks but is computationally challenging to implement. Fortunately, the earlier specialized treatment of mixed vertices enables this constructive formulation of blocks.

A vertex block of a vertex $v \in \mathcal{V}$ is the region enclosed by all planes passing through v and perpendicular to its convex neighboring edges (i.e., the edges that remain as edges in the convex hull of $star(v)$). As illustrated in Figure 5.5d, its shape resembles a ball with radius r, cut by multiple planes that are perpendicular to its neighboring edges. The constructive definition of U_v is as follows:

$$U_v = \{\mathbf{x} \in \mathbb{R}^N \mid ||\mathbf{x} - \mathbf{x}_v|| \leq r, (\mathbf{x} - \mathbf{x}_v)(\mathbf{x}_v - \mathbf{x}'_v) \geq 0 \text{ for } v' \in \mathcal{V}_v\}, \tag{5.7}$$

where $\mathbf{x}_v$ denote the position of vertex v, and $\mathcal{V}_v$ is the set of all vertices adjacent to v. Note that we do not differentiate non-convex neighboring edges. This is because the planes associated with non-convex neighboring edges only intersect with U_v at v and, therefore, do not influence the shape of U_v in the definition given by Equation 5.7.

An edge block of an edge $e \in \mathcal{E}$ is illustrated in Figure 5.5b. It is a cylinder with a radius of r being cut by 4 planes: 2 being perpendicular to the edge and 2 being perpendicular to each of the edge's two neighboring faces. The constructive definition of U_e is as follows:

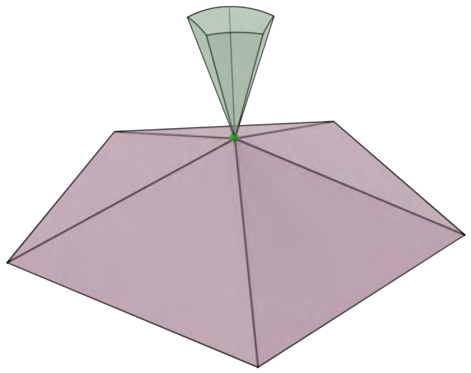
$$\begin{aligned} U_e = \{ & \mathbf{x} \in \mathbb{R}^N \mid \\ & dis(\mathbf{x}, e) \leq r, \\ & (\mathbf{x} - \mathbf{x}_{v_{e,1}})(\mathbf{x}_{v_{e,2}} - \mathbf{x}_{v_{e,1}}) > 0, \\ & (\mathbf{x} - \mathbf{x}_{v_{e,2}})(\mathbf{x}_{v_{e,1}} - \mathbf{x}_{v_{e,2}}) > 0, \\ & (\mathbf{x} - \mathbf{p}(\mathbf{x}(v_{e,1}), \mathbf{x}(v_{e,2}), \mathbf{x}_{v_{e,\text{next}}}) \cdot \\ & (\mathbf{p}(\mathbf{x}_{v_{e,1}}, \mathbf{x}_{v_{e,2}}, \mathbf{x}_{v_{e,\text{next}}}) - \mathbf{x}_{v_{e,\text{next}}}) \geq 0 \\ & \text{for } v_{e,\text{next}} \in \mathcal{V}_e \} \end{aligned} \tag{5.8}$$

where $v_{e,1}$ and $v_{e,2}$ are the two vertices of e, $\mathbf{p}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$ computes the perpendicular foot for $\mathbf{x}_3$ on the line defined by $\mathbf{x}_1, \mathbf{x}_2$, and $\mathcal{V}_e$ is the set of the vertices that share a facet with e. It is worth noting that since M is a manifold, $\mathcal{V}_e$ can contain at most two vertices, each belonging to one of e's neighboring faces. Additionally, when e is a boundary edge, $\mathcal{V}_e$ contains only one vertex, resulting in a half-cylinder-like block for the boundary edge, as illustrated in Figure 5.5c.

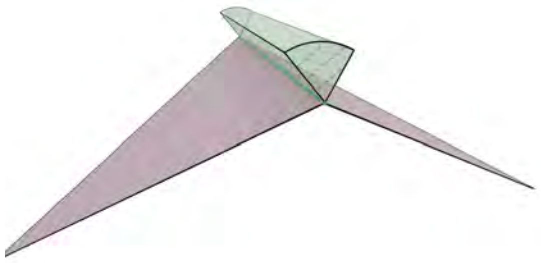
The block of a face $t \in \mathcal{T}$ is straightforward: it is formed by offsetting the face along its normal direction by a distance r, forming a prism (see Figure 5.5a). The constructive definition of U_t is as follows:

$$\begin{aligned} U_t = \{ & \mathbf{x} \in \mathbb{R}^N \mid \\ & dis(\mathbf{x}, t) \leq r, \\ & (\mathbf{p}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) - \mathbf{x})(\mathbf{p}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) - \mathbf{x}_3) > 0, \\ & (\mathbf{p}(\mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_1) - \mathbf{x})(\mathbf{p}(\mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_1) - \mathbf{x}_1) > 0, \\ & (\mathbf{p}(\mathbf{x}_1, \mathbf{x}_3, \mathbf{x}_2) - \mathbf{x})(\mathbf{p}(\mathbf{x}_1, \mathbf{x}_3, \mathbf{x}_2) - \mathbf{x}_2) > 0 \\ & \text{where } \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3 \text{ are the three vertices of } t \} \end{aligned} \tag{5.9}$$

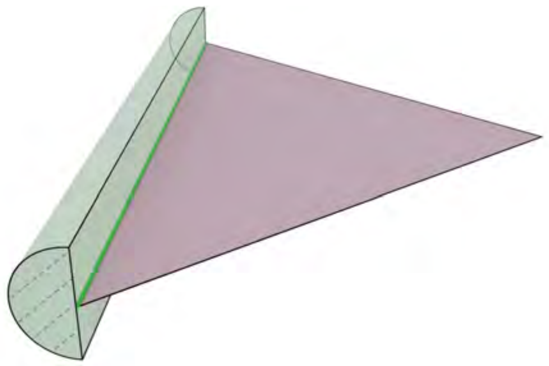
Another advantage of this constructive definition is that it naturally gives the definition of boundary edges and vertices, whose normals are not defined by [94].



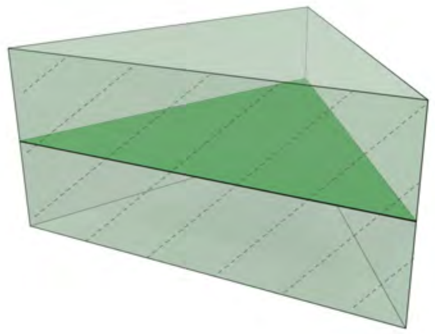
(a) Vertex Block



(b) Edge Block



(c) Boundary Edge Block



(d) Facet Block

Figure 5.5: Illustration of blocks corresponding to different types of faces. The regions shaded in light green represent the blocks, while the areas in solid green indicate the faces associated with these blocks. Boundaries marked with dashed lines are open, whereas those with solid colors are closed.

5.1.5 Contact Face Set

$$\mathcal{U} = \{U_a \mid a \in \mathbf{V} \cup \mathcal{E} \cup \mathcal{T}\} \tag{5.10}$$

We call $\mathcal{U}$ the *Intersection Aware Offset Geometry* of M. The Intersection Aware Offset Geometry serves as an analogy to volumetric meshes. The elements in $\mathcal{U}$ act as a building block of the geometry, as tetrahedron to tetmesh. A point can intersect with multiple $U_a \in \mathcal{U}$, this is how we know it has multi-layers of intersection with $\mathcal{U}$.

Based on $\mathcal{U}$, we can define a new type of Contact Face Set as:

$$\mathcal{F}_{\text{OGC}}(v) = \{a \mid \mathbf{x}_v \in U_a, v \not\subset a\}. \tag{5.11}$$

We refer to $|\mathcal{F}_{\text{OGC}}(\mathbf{x}_v)|$ as the number of layers of intersections for v.

For a point $\mathbf{x} \in \mathbb{R}^N$ and a face a, if $\mathbf{x} \in U_a$, there must exist $\mathbf{y} \in a^\circ$ such that $\mathbf{x} = \mathbf{y} + wr\mathbf{n}$, where $\mathbf{n} \in \mathcal{N}_a, w \in [0,1]$. Since a is a linear element, $\mathbf{y} = \mathbf{c}(\mathbf{x}, a)$ must hold, which means $\mathbf{x} - \mathbf{c}(\mathbf{x}, a) = wr\mathbf{n}$. Therefore, our selection of Contact Face Set $\mathcal{F}_{\text{OGC}}$ guarantees that each point will only contact faces that generate orthogonal normal contact force. In fact, our contact model has the following advantages:

- **Orthogonality**: for each point $\mathbf{x} \in U_a$, $(\mathbf{x} - \mathbf{c}(\mathbf{x}, a)) \perp a$ in a discrete sense.
- **Local Exclusiveness**: if $a \subset b$, $U_a \cap U_b = \emptyset$.
- **Covering** M^{+r}: $\bigcup_{U_a \in \mathcal{U}} U_a = M^{+r}$, i.e., $\mathcal{U}$ is a cover of M^{+r}.
- **Local Closest-ness**: if $\mathbf{x} \in U_a$, for b satisfies $a \subset b$ or $b \subset a$, we have $dis(\mathbf{x}, a) < dis(\mathbf{x}, b)$.

The covering property ensures the geometry we defined reflects the offset geometry M^{+r}. However, it added more information to M^{+r}. The local exclusiveness ensures that each block U_a can be a unique indicator of layers of intersection of the offset geometry, such as the overlapped part shown in Figure 2.1c.

It is worth noting that the block of a saddle vertex is an empty set, i.e., it will not contact with any other point. This is acceptable because if a point's distance to such a vertex is less than r, there must be at least one neighbor face or edge that is contacting with such a point.

5.1.6 Penetration Depth

Akin to [66], each layer of intersection requires a separate contact force to resolve. Naturally, we want to push the intersecting point along the normal direction to the boundary.

From the definition of blocks provided in Equation 5.6, we can see that if $\mathbf{x} \in U_a$, the distance to the surface of U_a along the normal direction is:

$$d_p = r - ||\mathbf{x} - \mathbf{c}(\mathbf{x}, a)|| = r - dis(\mathbf{x}, a), \tag{5.12}$$

we refer to d_p as the *penetration depth* of point $\mathbf{x}$ in U_a. d_p is a function of the vertex position $\mathbf{p}$ and $\mathbf{c}(\mathbf{p}, a)$. Therefore, the normal contact potential derived from d_p still accords with the formulation Equation 5.1.

5.1.7 Offset Geometry for Edge-edge Contact

We have defined the offset manifold for vertex-facet contact. Now we can define a new geometry by offsetting all the edges to support edge-edge contact. We extract all the vertices and edges of M to construct a new geometry M^e, which we refer to as the edge-only manifold. M^e will be a 1-dimensional manifold which only contains M's wireframes.

In M^e, the Gauss map of an edge e is a circle perpendicular to e (see Figure 5.6a), and its corresponding block forms a cylinder with e being its axis. The Gauss map of a vertex v is a sphere cut by 2 planes perpendicular to v's two neighbor edges, as illustrated in Figure 5.6b, with its block being shaped correspondingly (Figure 5.6d).

The constructive definition of the edge block of M^e is:

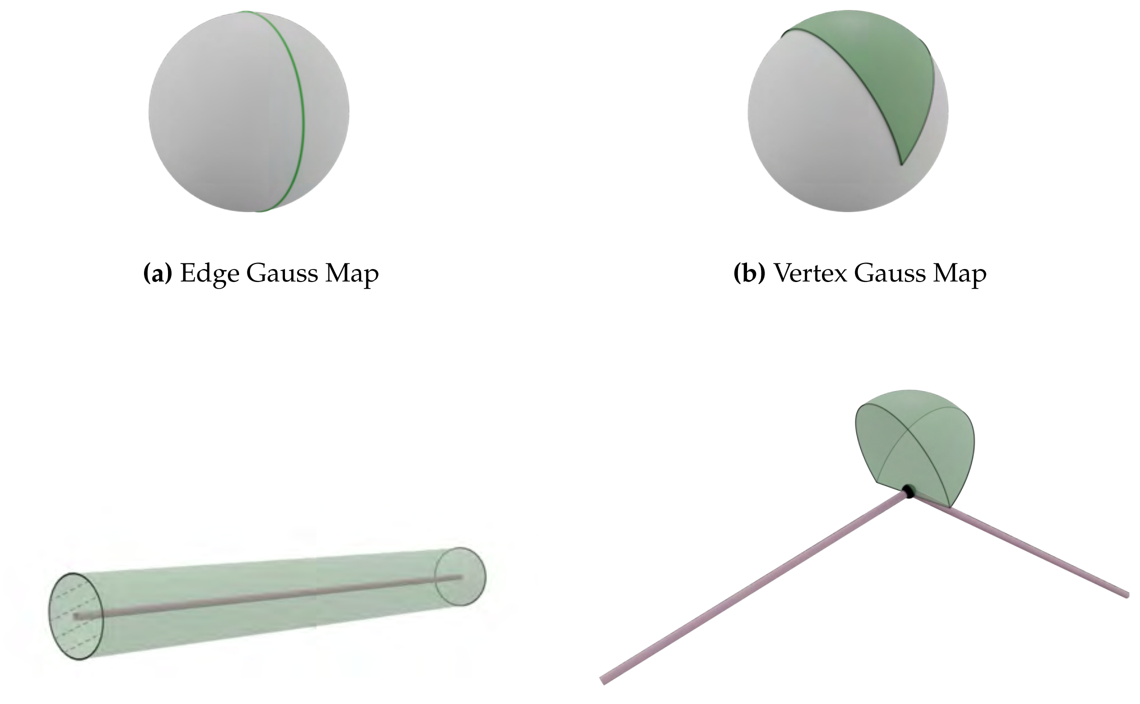
$$\begin{aligned} U_e^E =\{&\mathbf{x} \in \mathbb{R}^N \mid \\ &dis(\mathbf{x}, e) \leq r \\ &(\mathbf{x} - \mathbf{x}(v_{e,1}))(\mathbf{x}(v_{e,2}) - \mathbf{x}(v_{e,1})) > 0, \\ &(\mathbf{x} - \mathbf{x}(v_{e,2}))(\mathbf{x}(v_{e,1}) - \mathbf{x}(v_{e,2})) > 0\} \end{aligned} \tag{5.13}$$

Similarly, the constructive definition of the vertex block of M^e is:

$$U_v^e = \{\mathbf{x} \in \mathbb{R}^N \mid ||\mathbf{x} - \mathbf{x}_v|| \leq r, (\mathbf{x} - \mathbf{x}_v))(\mathbf{x}_v - \mathbf{x}_{v'}) \geq 0, \forall v' \in \mathcal{V}_v\} \tag{5.14}$$

The difference between edge-edge contact and vertex-facet contact is, that the force is applied on an edge instead of a single vertex. Similarly, we can define the normal contact potential for the edge-edge contact:

$$E_n^{ee}(M) = \sum_{e, e' \in \mathcal{E}_{\mathrm{OGC}}^{\mathrm{C}}} g(dis(e, e'), r) \tag{5.15}$$



(a) Edge Gauss Map **(b)** Vertex Gauss Map

(c) Edge Block **(d)** Vertex Block

Figure 5.6: Illustration of Gauss Maps and blocks in the edge-only manifold M^e of an edge and a vertex, respectively. The black dot represents the vertex in the vertex block diagram, and the dashed lines indices open boundaries in the edge block diagram.

where $\mathcal{E}_{\text{OGC}}$ is the set of all the actively contacting edge-edge pairs:

$$\begin{aligned}\mathcal{E}_{\text{OGC}} =& \{\{e_1, e_2\} \mid \\ & e_1, e_2 \in \mathcal{E}, \\ & e_1 \cap e_2 = \emptyset, \\ & \exists a \subset e_i, \mathbf{c}(e_i, e_j) \in U_a \text{holds for } i = 1, j = 2 \text{ and } i = 2, j = 1\}\end{aligned} \tag{5.16}$$

where $\mathbf{c}(e_i, e_j)$ is e_i's closest point to e_j.

The contact force between two edges is applied on two points: $\mathbf{c}(e, e')$ and $\mathbf{c}(e', e)$. [2] provided a way to smoothly filter out the parallel edge contact to avoid instability. Here we apply a similar procedure to our edge-edge contact force.

5.1.8 A C^2 Continuous 2-Stage Activation Function

The quadratic activation function is widely used because of its simplicity and non-stiff nature. However, it has a drawback: the contact normal force does not become infinite as two primitives approach each other. This would results in penetration when the large forces are pushing primitives towards each other.

To make sure the contact force will eventually get strong enough to overcome all other forces to successfully separate contacting primitives, the barrier activation function [2] became a popular choice. Their barrier function is a logarithmic function, multiplied by a quadratic function to make sure it is C^2 continuous at the point where the contact force disappears.

We propose a novel 2-stage activation function, which possesses the advantage of both of those energies:

$$g(d, r) = \begin{cases} \frac{k_c}{2}(r-d)^2 & \text{if } \tau \leq d \leq r \\ -k_c' log(d) + b & \text{if } 0 < d < \tau \end{cases} \tag{5.17}$$

where k_c and k_c' are 2 stiffness factors of the 2 stages, τ is a parameter determining where to stitch between those 2 stages. To make the function C^1 continuous at $d = \tau$, k_2 and b need to satisfy:

$$k_c' = \tau k_c (\tau - r)^2 \tag{5.18}$$

$$b = \frac{k_c}{2}(r-\tau)^2 + k_c' log(\tau) \tag{5.19}$$

This leaves us only one configurable parameter k_c, from which k_c' and b can be computed accordingly. We further let $\tau = \frac{r}{2}$ to make it C^2 continuous.

This is a combination of a pure quadratic function and a pure logarithmic function. With the k_1 properly set, most of the contacts will be handled in the quadratic stage, benefiting from the faster convergence of the quadratic function. In the second stage, since it is a pure logarithmic function, it is still less stiff than IPC's formulation [2].

Combining this activation function with our contact sets $\mathcal{F}_{\text{OGC}}$ and $\mathcal{E}_{\text{OGC}}$, we have obtained a normal contact energy that is C^2 continuous on most part (see the explanation in the Limitation Section) of $\mathcal{U}$. Additionally, the normal contact force derived from this normal contact energy is always orthogonal to the primitive it acts upon.

5.1.8.1 Friction

With the properly designed normal contact force, we can compute the friction force using the friction coefficient to compute the friction force. We use the lagged formulation of friction provided by [2], with the modification proposed by [1] to improve the stability.

5.1.9 Penetration-Free Simulation

We employ the technique provided in [81] to guarantee penetration-free. This technique relies on computing a *conservative bound* for each vertex v:

$$b_v = \gamma_p \min(d_{\min,v}, d^E_{\min,v}, d^T_{\min,v}), \tag{5.20}$$

where $0 < \gamma_p < 0.5$ is a relaxation parameter and $d_{\min,v}$ is v's minimal distance to all the facets that do not include v:

$$d_{\min,v} = \min_{t \in \mathcal{T}, v \notin t} dis(\mathbf{x}_v, t), \tag{5.21}$$

and $d^E_{\min,v}$ is the minimal value of v's neighbor edges' minimal distances to all other edges:

$$d^E_{\min,v} = \min_{e \in \mathcal{E}_v} d_{\min,e}, \tag{5.22}$$

$$d_{\min,e} = \min_{e' \in \mathcal{E}, e \cap e' = \emptyset} dis(e, e'), \tag{5.23}$$

and $d^T_{\min,v}$ is the minimal value of v's neighbor facets' minimal distances to all other vertices:

$$d^T_{\min,v} = \min_{t \in \mathcal{T}_v} d_{\min,t}, \tag{5.24}$$

$$d_{\min,t} = \min_{v' \in \mathcal{V}, v' \not\subset t} dis(v', t), \tag{5.25}$$

where $\mathcal{E}_v$ and $\mathcal{T}_v$ represents v's neighbor edges and facets respectively.

If the model starts in an intersection-free state X^{prev}, it will remain intersection-free in state if each $\mathbf{x}_v$ satisfies:

$$||\mathbf{x}_v - \mathbf{x}_v^{\text{prev}}|| \leq b_v, \forall v \in \mathcal{V}. \tag{5.26}$$

Starting with a penetration-free state X^{prev}, our method computes b_v and records $\mathbf{x}_v^{\text{prev}}$ for each v. Then after some solver iterations, each vertex will reach a new position $\mathbf{x}_v$. Our method checks each vertex individually to see if it satisfies the condition in Equation 5.26. If a vertex does not satisfy the condition, its displacement is truncated to stay within the conservative bound. b_v. Subsequently, our method recomputes b_v and records $\mathbf{x}$ as the penetration-free starting state, repeating this process iteratively.

Since the conservative bound and X^{prev} can be updated as needed during the solver's iterations, they do not restricted each vertex's total displacement within a *time step*, only limiting the displacement within each individual iteration. A vertex near an obstacle may be constrained in initial iterations, but the bound-update will be triggered once it reaches the bound. The new bound becomes larger since the repulsive force pushes it away from the obstacle. This procedure will repeat until convergence. As shown in Figure 5.15, the solver converges under these bounds without introducing additional artifacts

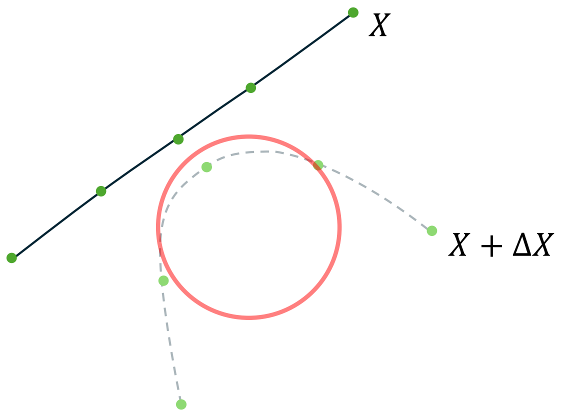
5.1.10 Comparing to IPC

Our method can be viewed as a trust region based method for a constrained optimization problem, where the constraints are the penetration-free constraints. The spherical region we compute for each vertex is the trust region we formulate to enforce the constraints. In contrast, IPC [2] employs a CCD-aware line search technique to achieve penetration free-state, which requires truncating the step.

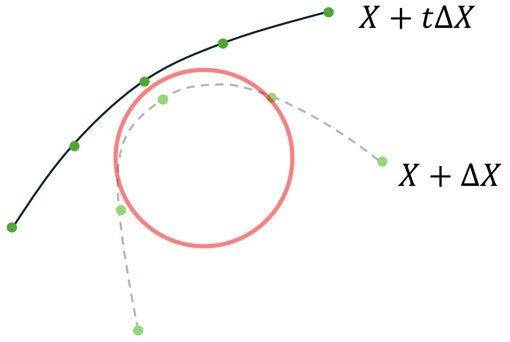
The CCD-aware line search technique maintains a penetration-free state by applying CCD after every iteration of the optimization. It truncates the optimization step of the physics solver at where the first collision happens, thereby preventing penetration. However, this also means a local collision stops the progress of all other points, even if those points are still far from intersecting. This is illustrated in Figure 5.7b, where the whole step is stopped by the vertex in the middle which is closest to the obstacle. Each iteration can be computationally expensive, and truncating the global optimization step entirely often results in only a small fraction of the step being utilized. This approach overlooks the fact that most parts of the model could still make significant progress along the optimization direction, leading to wasted computation and slower convergence. This issue becomes particularly evident when parts of the model are in close proximity to one another.

In our formulation, b_v is different for each $v \in \mathcal{V}$, as illustrated in Figure 5.7c. The value of b_v is smaller in regions that are actively in contact and larger in regions that are far from others. As a result, each b_v has only a local impact. Even when certain parts of the model are close to each other, such as the vertices in the middle of Figure 5.7c, the conservative bounds of other regions, like the vertices on the sides, remain unaffected. These unaffected regions can still utilize relatively large step sizes.

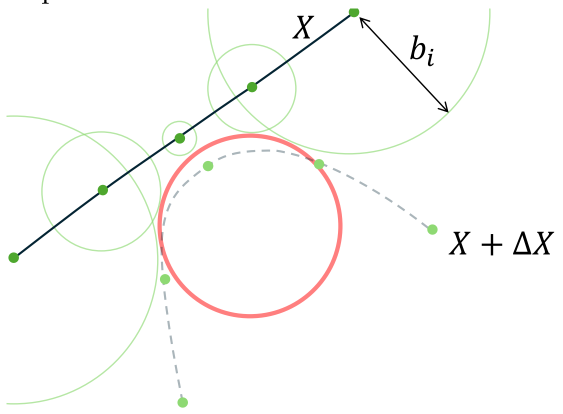
Our contact force formulation allows for a significantly larger contact radius compared to IPC. As a result, even primitives that are actively in contact can maintain a relatively large distance. This, in turn, enables our method to take bigger steps per iteration and achieve fast convergence, despite employing conservative bound truncation. In contrast,



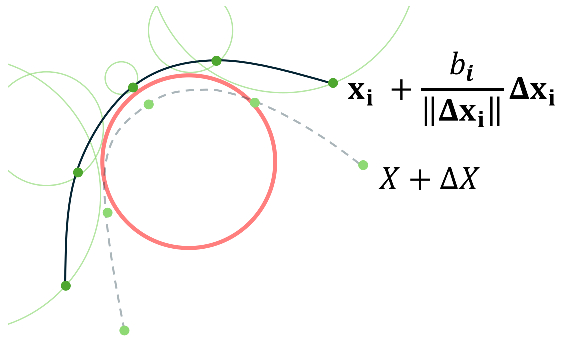
(a) Previous Position and a Full Optimization Step



(b) Iteration Using IPC's Scheme



(c) Conservative Bounds


$\mathbf{x_i} + \frac{b_i}{\|\mathbf{\Delta x_i}\|}\mathbf{\Delta x_i}$
$X + \Delta X$

(d) Iteration Using by Our Scheme

Figure 5.7: Comparing different schemes to preserve penetration-free state in a single solver iteration. The black line represents the shape of M, the dashed gray line represents the destination position after taking a full step given by the optimizer in that iteration, the red circle represents an obstacle, and the green dots represent vertices of M. (a) The penetration-free position X^{prev} in the previous iteration, and a position $X + \Delta X$ after taking a full optimization step, which presents penetration; (b) the penetration-free optimization step given using IPC's CCD-aware line search scheme; (c) illustration of out conservative bounds b_i which vary at each vertex; (d) the penetration-free optimization step given by our scheme.

simply combining our trust-region-based optimization scheme with IPC's contact energy will not work, because a small contact radius as IPC uses will result in a near-zero conservative bound.

Furthermore, as we will explain in the next section, there is no need to use a separate function to compute b_v. Instead, this can be seamlessly integrated into the contact detection process with negligible overhead. Additionally, the computation of b_v is fully parallel, and does not require CPU-GPU synchronization when implemented on GPU. This approach offers a significant advantage over the CCD-based line search employed by IPC, which requires multiple computationally expensive continuous collision detections and synchronizations in a single iteration.

5.1.11 Offset Geometry for Mesh with Different Dimensionalities

For volumetric meshes, it is possible to resolve intersections after they occur using the method proposed by [66]. Therefore, for volumetric mesh simulations, we can skip conservative bound culling to accelerate computation. Here we propose a faster method specially tailored for the volumetric mesh simulations.

For a volumetric mesh M, the offset operation should be applied to its surface ∂M to obtain an intersection-aware offset geometry $\mathcal{U}(\partial M)$. Note that in this case, we only offset the geometry outward, in the direction of the surface normal. The penetration depth computed from $\mathcal{U}(\partial M)$ is compatible with the penetration depth provided by [66].

We use pure quadratic contact energy in volumetric simulation, e.g., only using the first stage of Equation 5.17. At the beginning of each step, the simulator performs DCD (discrete collision detection) for each vertex to determine whether they have intersected M. If penetration is detected, the simulator computes the penetration depth d_p using the method proposed by [66]. This penetration depth needs to be adjusted to $d_p + r$ to match the penetration depth of the offset geometry. If no intersection is found, the simulator performs another DCD to detect its intersection with $\mathcal{U}(\partial M)$ and computes the penetration depth in $\mathcal{U}(\partial M)$. This ensures that the penetration depths in M and $\mathcal{U}(\partial M)$ are consistent, resulting in consistent contact forces from both the inside and outside of the mesh. The contact force is greater than 0 at ∂M due to the offset layer. With properly adjusted contact stiffness, most contact will still occur outside the mesh, maintaining an intersection-free state for most parts of the mesh. We use this scheme to handle the cloth-body contact in our cloth simulation experiments.

For 1D meshes immersed in 3D space, such as those used in hair and yarn-level simulations, they can be treated in the same way as the edge-only manifold proposed in Section 5.1.7. Specifically, we employ this contact model to generate the yarn simulation results presented in Figure 5.13 and Figure 5.14.

5.2 Algorithm

We have defined the contact force and energy for the vertex-facet contact and the edge-edge contact. Now we propose the algorithms to practically detect those contacts. Since the intersection-aware offset geometry is composed of many blocks, a trivial implementation will be building a BVH (Bounding Volume Hierarchy) of all those blocks, and looping through all the vertices and edges to detect intersections with those blocks.

However, these blocks correspond to different types of faces, including vertices, edges, and facets. Building a BVH that contains all these blocks would result in an excessively large structure. Instead, we present a method that only requires building a BVH for the faces with the highest dimensionality: facets for vertex-facet contact detection and edges for edge-edge contact detection. Note that such a BVH is constructed based on the bounding boxes of original faces, not the offset ones. Additionally, those algorithms are capable of computing $d_{\min,v}$, $d^E_{\min,v}$, and $d^T_{\min,v}$simultaneously with the contact detection.

5.2.1 Vertex Facet Contact

The algorithm for detecting vertex-facet contact is provided in algorithm 2. As previously mentioned, we only maintain a BVH for all the facets. To detect vertex-facet contact, we do a point query with center $\mathbf{x}(v)$ and radius r_q for each vertex $v \in \mathcal{V}$. r_q is a custom parameter satisfies $r_q \geq r$.

For each facet f within the query radius r_q, the algorithm computes its closest point to v, the face a on the facet where the closest point is located, and the distance $d = dis(v, t)$ (line 4,5,8). Note that a can be either a vertex, or an edge, or t°. Then it updates v's minimal distance to facets, $d_{\min,v}$. We also update f's minimal distance to vertices in parallel, $d_{min,t}$, using an atomic min operation. This is to avoid a race condition since multiple vertex

Algorithm 2: vertexFacetContactDetection

Input: v: a vertex, r: contact radius, r_q: query radius
Output: $\mathcal{F}_{\text{OGC}}(v)$: set of faces that are in contact with v;
$\mathcal{V}_{\text{OGC}}(t)$: set of vertices that are in contact with t;
$d_{\text{min},v}$: the minimal distance from v to another faces;
$d_{min,t}$: the minimal distance from t to all other vertices.

```
1  $d_{\text{min},v} = r_q$
   // sphere query on the facet BVH with center $\mathbf{x}(v)$ and radius $r_q$
2  for each $t \in \mathcal{T}$ s.t. $dis(v,t) < r_q$ do
       // avoid contact with adjacent facet
3      if $v \subset t$ then continue
4      $d = dis(v,t)$
5      $d_{\text{min},v} = min(d, d_{\text{min},v})$
       // multiple vertex query threads may access the same $d_{min,f}$ simultaneously, thus this must be
          an atomic min operation
6      $d_{min,t} = \min(d, d_{min,t})$
7      if $d < r$ then
8          $a = \text{closestFaceFacetToVertex}(v,t)$
           // avoid duplicated contact with $a$ detected from a neighbor facet
9          if $a \in \mathcal{F}_{\text{OGC}}(v)$ then continue
10         if $a \in \mathcal{V}$ then
               // Equation 5.7
11             if $checkVertexFeasibleRegion(\mathbf{x}(v), a)$ then
12                 $\mathcal{F}_{\text{OGC}}(v) = \mathcal{F}_{\text{OGC}}(v) \cup \{a\}$
13                 $\mathcal{V}_{\text{OGC}}(t) = \mathcal{V}_{\text{OGC}}(t) \cup \{v\}$
14         else if $a \in \mathcal{E}$ then
               // Equation 5.8
15             if $checkEdgeFeasibleRegion(\mathbf{x}(v), a)$ then
16                 $\mathcal{F}_{\text{OGC}}(v) = \mathcal{F}_{\text{OGC}}(v) \cup \{a\}$
17                 $\mathcal{V}_{\text{OGC}}(t) = \mathcal{V}_{\text{OGC}}(t) \cup \{v\}$
18         else
               // $v$ must be in the feasible region in this case
19             $\mathcal{F}_{\text{OGC}}(v) = \mathcal{F}_{\text{OGC}}(v) \cup \{t\}$
20             $\mathcal{V}_{\text{OGC}}(t) = \mathcal{V}_{\text{OGC}}(t) \cup \{v\}$
21 end
22 return $\mathcal{F}_{\text{OGC}}(\mathbf{x}_v)$, $d_{\text{min},v}$
```

query threads can access the same $d_{min,t}$ simultaneously.

Since all the vertices whose distance to t is less than r_q will visit t, this ensures that we are computing the correct $d_{min,t}$. Both $d_{\min,v}$ and $d_{min,f}$ are initialized as r_q, because the query does not look beyond that distance. This means that even if there are no active contact pairs detected, $d_{\min,v}$ and $d_{min,f}$ are still upper-bounded by r_q, because we do not know if there is a facet whose distance to v is marginally larger than r_q. That is why we make r_q a separate parameter. Making r_q larger than r will not detect more contacts, but it can potentially improve the conservative bound for each vertex, thereby enhancing convergence.

The next step will be determining whether a is in contact with v, i.e., whether $v \in U_a$. Note that when a is not a facet, it is shared by multiple neighboring facets. In this case, multiple facets can return the same closest face a. To avoid duplicated contacts, we check whether a already exists in the contacting face set $\mathcal{F}_{OGC}(\mathbf{x}_v)$. If $a \notin \mathcal{F}_{OGC}(v)$, we proceed to check $v \in U_a$ using U_a's constructive definition (Equation 5.7, Equation 5.8). Note that if the closest point is located in the interior of t, i.e., $a = t$, $v \in U_a$ is guaranteed. Therefore, no feasible region check is required in this case. For the convenience of contact force computation, we also maintain a list $\mathcal{V}_{\text{OGC}}(t)$ for each $t \in \mathcal{T}$, which is the set of vertices that are in contact with t. After putting a face into $\mathcal{F}_{OGC}(v)$, we also put v into $\mathcal{V}_{\text{OGC}}(t)$ of the corresponding facet using atomic operation.

According to Equation 5.12, if $a \in t$ contacts with v, it must be the closest face on t to v. The local exclusive property guarantees that v can only be in contact with at most one face on t. If v contacts with $a \in t$, it will not contact all other faces of t. Since v will visit all the facets whose distance to v is less than r, this guarantees that algorithm 2 will not miss or duplicate any vertex-facet contact.

5.2.2 Edge Edge Contact

Similarly, we give the algorithm that detects edge-edge contact in algorithm 3. Similar to algorithm 2, it works on the BVH of all the edges, and applies a sphere query centered at $\mathbf{x}_m$ with radius $r_q + \frac{l}{2}$ for each edge, where $\mathbf{x}_m$ and l are the midpoint and length of that edge, respectively. Each query also computes the $d_{\min,e}$. Since every edge has its query thread, no automatic operation is needed here. Note that since each edge detects its own

Algorithm 3: edgeEdgeContactDetection

```
Input: $e$: a edge, $r$: contact radius, $r_q$: query radius
Output: $\mathcal{E}_{\text{OGC}}(e)$: set of faces contacting $e$;
$d_{\text{min},e}$: the minimal distance from $e$ to all other edges.
1  $d_{\text{min},e} = r_q$
2  $\mathbf{x}_m$ = midpoint of $e$
3  $l$ = length of $e$
   // sphere query on the facet BVH with center $\mathbf{x}_m$ and radius $r_q + \frac{l}{2}$
4  for each $e'$ s.t. $dis(e, e') < r_q + \frac{l}{2}$ do
       // avoid contact with adjacent edge
5      if $e \cap e' \neq \emptyset$ then continue
6      $d = dis(e, e')$
7      $d_{\text{min},e} = min(d, d_{\text{min},e})$
8      if $d < r$ then
9          $\mathbf{x}_c = \mathbf{C}(e, e')$
10         $a$ = closestFaceEdgetToEdge$(e, e)$
           // avoid duplicated contact with $a$ detected from a neighbor edge
11         if $a \in \mathcal{E}_e^C$ then continue
12         if $a \in \mathcal{V}$ then
               // Equation 5.14
13             if $checkVertexFeasibleRegionEdgeOffset(\mathbf{x}_c, a)$ then
14                 $\mathcal{E}_{\text{OGC}}(e) = \mathcal{E}_{\text{OGC}(e)} \cup \{e, a\}$
15         else
               // $v$ must be in the feasible region in this case
16             $\mathcal{E}_{\text{OGC}}(e) = \mathcal{E}_{\text{OGC}}(e) \cup \{e, e'\}$
17     return $\mathcal{E}_{\text{OGC}}(e)$, $d_{\text{min},e}$
18 end
```

contacts, each edge-edge contact will be automatically detected exactly twice: one from each side.

5.2.3 Simulation Pipeline

Now that we have provided the contact energy and the algorithms to detect those contacts, the next step would be integrating it into an actual simulation pipeline. Theoretically, the contact force we formulated can be used in a variety of time integrators, including both explicit and implicit ones. Here we provide an algorithm combining Offset Geometric Conact with backward Euler in algorithm 4.

There are 3 major stages of the simulation pipeline: contact detection (line 4 $\sim$ 19), simulation solve (line 20 $\sim$ 22), and conservation bound truncation (line 23 $\sim$ 30). We will introduce each stage correspondingly.

5.2.3.1 Contact Detection

In the contact detection stage, the simulator will apply the previously provided contact detection algorithms to the model. Note that before we apply the vertex-facet contact detection, we need to initialize all the $d_{\min,f}$ to their upper-bound r_q (line 6,7). Then we apply all the vertex-facet contact and edge-edge contact detections in parallel, which computes the contacting faces and each face's minimal distance from other faces. At last, the simulator computes the conservative bounds b_v for all the vertices based on that information (line 17 $\sim$ 19).

5.2.3.2 Simulation Solve

The first step in simulation solving is to apply an initialization that avoids penetration. A trivial approach is to use the positions from the previous step, but a better initialization can improve convergence and reduce damping. Since any guess within conservative bounds will be penetration-free, we can choose an arbitrary initialization scheme and truncate it to stay within these bounds, ensuring a penetration-free start:

$$\mathbf{x}_v^{\text{init}*} = \begin{cases} \mathbf{x}_v^{\text{init}} & \text{if } ||\mathbf{x}_v^{\text{init}} - \mathbf{x}_v^t|| \le b_v \\ \frac{\mathbf{x}_v^{\text{init}} - \mathbf{x}_v^t}{||\mathbf{x}_v^{\text{init}} - \mathbf{x}_v^t||} b_v + \mathbf{x}_v^t & \text{if } ||\mathbf{x}_v^{\text{init}} - \mathbf{x}_v^t|| > b_v \end{cases} \tag{5.27}$$

where $\mathbf{x}_v^{\text{init}*}$ and $\mathbf{x}_v^{\text{init}}$ are the initialization post and pre truncation, respectively, $\mathbf{x}_v^t$ is v's position at the last step.

Algorithm 4: Simulation Step with Offset Geometry Contact

```
Input: $X^t \in \mathbb{R}^{K\times 3}$: stacked positions of vertices from previous step;
$\mathbf{v}^t \in \mathbb{R}^{K\times 3}$: stacked velocities of vertices from previous step;
$\mathbf{a}_{\text{ext}}$: external acceleration;
$\gamma$: a parameter controls when to do a new collision detection;
$M = \{\mathcal{V}, \mathcal{E}, \mathcal{T}\}$;
$r$: contact radius, $r_q$: query radius
Output: $X \in \mathbb{R}^{K\times 3}$: stacked positions of vertices for current step

collisionDetectionRequired = true
$X = X^t$
$Y = X^t + \Delta t \mathbf{v}^t + \Delta t^2 \mathbf{a}_{\text{ext}}$
for each $i$ in $1, 2, \ldots, n_{iter}$ do
    if collisionDetectionRequired then
        // Initialize $d_{\min,t}$ to their upper-bound
        parallel for each $t \in \mathcal{T}$ do
            $d_{\min,t} = r_q$
        end
        parallel for each $v \in \mathcal{V}$ do
            $\mathcal{F}_{\text{OGC}}(v), d_{\min,v} = \text{vertexFacetContactDetection}(v, r, r_q)$
        end
        parallel for each $e \in \mathcal{E}$ do
            $\mathcal{E}_{\text{OGC}}(e), d_{\min,e} = \text{edgeEdgeContatDetection}(e, r, r_q)$
        end
        $X^{\text{prev}} = X$
        collisionDetectionRequired=false
    parallel for each $v \in \mathcal{V}$ do
        // Equation 5.20
        $b_v = \text{computeSconvertativeBounds}(v)$
    end
    if $i == 1$ then
        $X = \text{applyInitialGuess}(X^t, \mathbf{v}^t, \mathbf{a}_{\text{ext}})$
    $X = \text{simulationIteration}(\{\mathcal{F}_{\text{OGC}}\}, \{\mathcal{V}_{\text{OGC}}\}, \{\mathcal{E}_{\text{OGC}}\}, X, X^t, Y, \mathbf{v}^t, \mathbf{a}_{\text{ext}}, M)$
    numVerticesExceedBound = 0
    // Truncated the vertex displacements to be within $b_v$
    parallel for each $v \in \mathcal{V}$ do
        if $||\mathbf{x}_v - \mathbf{x}_v^{prev}|| > b_v$ then
            $\mathbf{x}_v = \frac{\mathbf{x}_v - \mathbf{x}_v^{\text{prev}}}{||\mathbf{x}_v - \mathbf{x}_v^{\text{prev}}||} + \mathbf{x}_v^{\text{prev}}$
            // Atomic increment
            numVerticesExceedBound++
    end
    if $numVerticesExceedBound >= \gamma_e K$ then
        // If a certain amount of vertices move out of their conservative bounds, do a new collision
           detection
        collisionDetectionRequired = true
    // Optional Convergence Evaluation
    if $evaluateConvergence(\{\mathcal{F}_{OGC}\}, \{\mathcal{V}_{OGC}\}, \{\mathcal{E}_{OGC}\}, X, X^t, \mathbf{v}^t, \mathbf{a}_{ext}, M)$ then
        break
end
return $X$
```

Algorithm 5: VBD Iteration with Contact

Input: X: the initialization value; X^t: the positions of the previous step; Y: the inertia;
Output: This step's position $\mathbf{x}^{t+1}$ and velocity $\mathbf{v}^{t+1}$.

```
1  for each color $c$ do
       // Block-level parallelization
2      parallel for each vertex $v$ in color $c$ do
3          $\mathbf{f}_v = -\frac{m_v}{h^2}(\mathbf{x}_v - \mathbf{y}_v)$, $\mathbf{H}_v = \frac{m_v}{h^2}\mathbf{I}$
           // Thread-level parallelization
4          parallel for each $t \in \mathcal{T}_v$ do
               // Variables in shared memory
5              $\mathbf{f}_{v,t} = -\frac{\partial E_t}{\partial \mathbf{x}_v}$, $\mathbf{H}_{v,t} = \frac{\partial^2 E_t}{\partial \mathbf{x}_v \partial \mathbf{x}_v}$
6          end
           // Local reduction sums
7          $\mathbf{f}_v$+=$\sum_{t\in\mathcal{T}_v} \mathbf{f}_{v,t}$, $\mathbf{H}_v$+=$\sum_{t\in\mathcal{T}_v} \mathbf{H}_{v,t}$
           // Thread-level parallelization
8          parallel for each $e \in \mathcal{E}_v$ do
               // Variables in shared memory
9              $\mathbf{f}_{v,e} = -\frac{\partial E_e}{\partial \mathbf{x}_v}$, $\mathbf{H}_{v,e} = \frac{\partial^2 E_e}{\partial \mathbf{x}_v \partial \mathbf{x}_v}$
10         end
           // Local reduction sums
11         $\mathbf{f}_v$+=$\sum_{e\in\mathcal{E}_v} \mathbf{f}_{v,e}$, $\mathbf{H}_v$+=$\sum_{t\in\mathcal{E}_v} \mathbf{H}_{v,e}$
           // Accumulate the force and Hessian of the vertex-facets contact of the vertex side
           // Thread-level parallelization
12         parallel for each $a \in \mathcal{F}_{OGC}(v)$ do
               // Variables in shared memory
13             $\mathbf{f}_{v,a} = -\frac{\partial E_c^{v,f}(v,a)}{\partial \mathbf{x}_v}$, $\mathbf{H}_{v,a} = \frac{\partial^2 E_c^{v,f}(v,a)}{\partial \mathbf{x}_v \partial \mathbf{x}_v}$
14         end
           // Local reduction sums
15         $\mathbf{f}_v$+=$\sum_{a\in\mathcal{F}_{\mathrm{OGC}}(v)} \mathbf{f}_{v,a}$, $\mathbf{H}_v$+=$\sum_{a\in\mathcal{F}_{\mathrm{OGC}}(v)} \mathbf{H}_{v,a}$
           // Accumulate the force and Hessian of the vertex-facets contact of the neighbor facets
16         for each $t \in \mathcal{T}_v$ do
               // Thread-level parallelization
17             parallel for each $v' \in \mathcal{V}_{OGC}(t)$ do
                   // Variables in shared memory
18                 $\mathbf{f}_{t,v'} = -\frac{\partial E_c^{v,f}(v',t)}{\partial \mathbf{x}_v}$, $\mathbf{H}_{t,v'} = \frac{\partial^2 E_c^{v,f}(v',t)}{\partial \mathbf{x}_v \partial \mathbf{x}_v}$
19             end
               // Local reduction sums
20             $\mathbf{f}_v$+=$\sum_{v'\in\mathcal{V}_{\mathrm{OGC}}(t)} \mathbf{f}_{t,v'}$, $\mathbf{H}_v$+=$\sum_{v'\in\mathcal{V}_{\mathrm{OGC}}(t)} \mathbf{H}_{t,v'}$;
21         end
           // Accumulate the force and Hessian of the edge-edge contact of neighbor edges
22         for each $e \in \mathcal{E}_v$ do
               // Thread-level parallelization
23             parallel for each $e' \in \mathcal{E}_{OGC}(e)$ do
                   // Variables in shared memory
24                 $\mathbf{f}_{e,e'} = -\frac{\partial E_c^{e,e}(e,e')}{\partial \mathbf{x}_v}$, $\mathbf{H}_{e,e'} = \frac{\partial^2 E_c^{e,e}(e,e')}{\partial \mathbf{x}_v \partial \mathbf{x}_v}$
25             end
               // Local reduction sums
26             $\mathbf{f}_v$+=$\sum_{e'\in\mathcal{E}_{\mathrm{OGC}}(e)} \mathbf{f}_{e,e'}$, $\mathbf{H}_v$+=$\sum_{e'\in\mathcal{E}_{\mathrm{OGC}}(e)} \mathbf{H}_{e,e'}$;
27         end
28         $\mathbf{x}_v \leftarrow \mathbf{x}_v + \mathbf{H}_v^{-1}\mathbf{f}_v$
29     end
30 end
31 return $X$
```

The second step is solving the non-linear equation of the backward Euler time integration. OGC is compatible with various solvers, such as Newton's method, gradient descent, and block coordinate descent, provided they work with the energy formulation of OGC. These solvers can be seen as functions that yield a displacement from the previous position to reduce the energy. To prevent penetration, we need to post-process the displacements by truncating them within the conservative bounds.

Here we present an efficient GPU implementation of a VBD [1] solver, as shown in algorithm 5. In this algorithm, m_v denotes the mass of vertex v, E_t the elastic energy of facet t, E_e the bending energy of edge e, and $E_c^{v,f}$ and $E_c^{e,e}$ represent the contact energies (including both normal and frictional components) for vertex-facet and edge-edge contacts, respectively. We employ a two-level parallelism scheme similar to that in [1], except we use thread-level parallelism to accumulate contact forces and Hessians for each vertex, as well as for its neighboring facets and edges.

5.2.3.3 Conservative Bound Truncation

According to Equation 5.26, starting from a penetration-free state X^{prev}, as long as the displacement of each vertex satisfies $||\Delta \mathbf{x}_v|| < b_v$, it is guaranteed that the model will not create any penetration. Note that $\Delta \mathbf{x}_v$ may not be the displacement of a single iteration of simulation but can be the *accumulated* displacement from multiple iterations. Therefore, collision detection is not needed in every iteration to guarantee a penetration-free state. Only when some vertices have exceeded their conservative bounds, new collision detection is needed to refresh the conservative bounds and recalculate the contacts.

This property is particularly beneficial for first-order or locally second-order methods, such as gradient descent or vertex block descent, since these methods create relatively small displacements at each iteration, and each iteration is very fast. For these methods, new collision detection is typically needed only after a fair amount of iterations.

During the collision detection stage, the simulator records the position where the contact detection is conducted as X^{prev} (line 15). After each simulation iteration, the simulator computes the displacement of each vertex from X^{prev}, and truncates the displacement to be within the bounds (line 25 $\sim$ 27). Instead of redoing contact detection every time one vertex moves out of its bound, a threshold γ_e is used to control when to apply a new contact detection. A new contact detection is performed only after the number of vertices moving out of their bounds exceeds $\gamma_e K$ (line $29, 30$). Before this threshold is reached, those vertices can be truncated multiple times and cannot move any further, though they can still adjust the direction of their displacement.

5.3 Results

We implemented our algorithm on both CPU and GPU platforms. The CPU implementation, written in C++, was executed on an AMD Ryzen 7950X with 64 GB of memory. We implement the GPU version using NVIDIA Warp [191], and run it on a NVIDIA RTX 4090. For the simulation parameters, we set $\gamma_p = 0.45$ and $\gamma_e = 0.01$. Details of other experiment-specific parameters and performance metrics are provided in Table 5.1. We plan to open-source both the C++ and Warp versions of our code, ensuring they are user-friendly and ready for out-of-the-box use.

5.3.1 Cloth Simulation

5.3.1.1 Large Scale Test

To evaluate the stability, efficiency, and scalability of our method, we present a simulation of colliding 50 layers of cloth. Those clothes are dropped onto a cylinder, collide with each other, and then slide to the ground. They form a pile on the ground and eventually rest in contact. The dropping process is visualized in Figure 5.8. Despite the complicated contacts, the simulation remains penetration-free the entire time.

5.3.1.2 Stress Tests

We present two experiments to demonstrate the stability and performance of our method in scenarios involving numerous complex self-collisions, extreme normal contact forces, and frictions. Both of those experiments maintain penetration-free states the entire time.

The first experiment, illustrated in Figure 5.9, simulates the formation of a complex tight knot. We initialize the knot in a loose form, then tighten it by pulling its two ends. As the knot tightens, small sub-knots form and collide with each other. Eventually, these small knots merge into a tight, multi-layered complex knot.

The second experiment, illustrated in Figure 5.10, involves twisting a square-shaped cloth's two ends for eight complete turns. This example features extreme deformations,

Table 5.1: Peformance results and simulation

Experiment Name	Number of		Contact &Fiction			Simulation Parameters		Time per step (avg./max)	
	Vert.	Primitives	k_c	μ_c, ϵ_v	$r(mm)$	Time Step (sec.)	Iterations	CPU VBD	GPU VBD
50 Layers of Cloth (Figure 5.8)	1M	1.96M	1e5	0.2, 1e-2	2	1/1200	40	0.21/0.55s	6.3/11.5ms
Tightening a knot (Figure 5.9)	48K	92K	1e5	0.4, 1e-2	2	1/300	50	122/180ms	4.4/6.8ms
Twisting Cloth (Figure 5.10)	10K	19.6K	1e5	0.2, 1e-2	2	1/300	10	21/30ms	0.9/1.5ms
Cloth on Body (Figure 5.12)	15.6K	29K	1e5	0.5, 1e-2	2	1/200	20	30/42ms	1.2/1.4ms
Yarn Stretch (Figure 5.13)	65K	65K	2e-3	0.1, 1e-3	1.5	3e-4	4	NA	0.48/0.56ms
Yarn Twist (Figure 5.14)	65K	65K	2e-3	0.1, 1e-3	1.5	3e-4	4	NA	0.52/0.66ms
3 Layers of Cloth on Sphere (Figure 5.15)	14.7K	28.6K	1e4	0.5, 1e-2	2	1/100	NA	See figure	NA
1 Layer of Cloth on Sphere (Figure 5.16)	4.9K	9.5K	1e4	0.5, 1e-2	5	1/100	40	62/75ms	1.2/3.2ms
Twisting Volumetric Mat (Figure 5.17)	15K	46.8K	1e5	0.2, 1e-2	2	1/240	20	NA	5.5/8.5ms

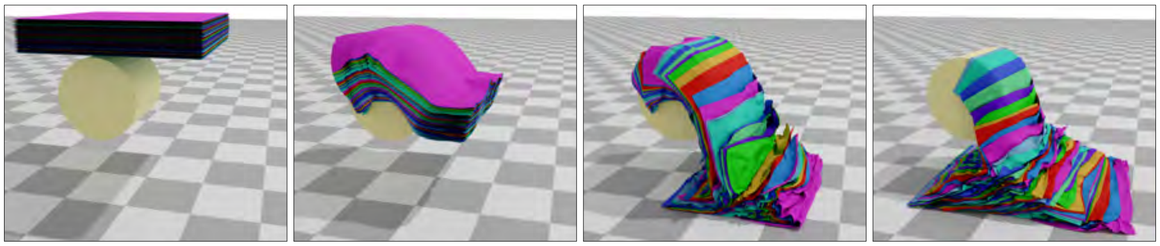


Figure 5.8: Fifty layers of cloth are dropped onto a cylinder, then slide to the ground. This simulation has 246K vertices and 475K triangles. We use $r = 3$mm, a time step of 1/1200s, and 40 iterations per step. The average/maximum computation time per time step is 0.21/0.55s on the CPU and 6.3/11.5ms on the GPU.

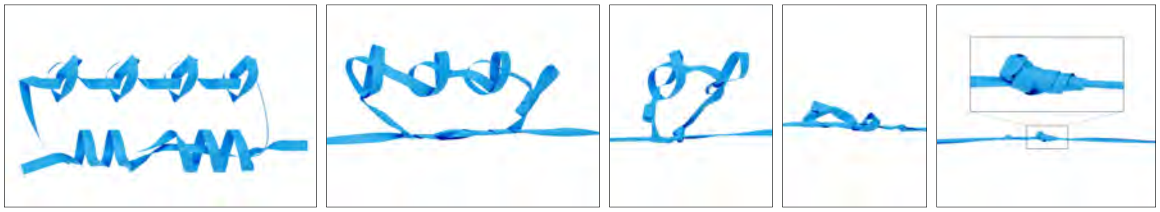


Figure 5.9: Tightening a complex knot with 48K vertices and 91K faces by pulling its two ends. At the end of the simulation, the mesh forms a multi-layered, very tight knot. We use $r = 2$mm, a time step of 1/300s, and 50 iterations per step. The average/maximum computation time per time step is 122/180ms on the CPU and 6.3/11.5ms on the GPU.

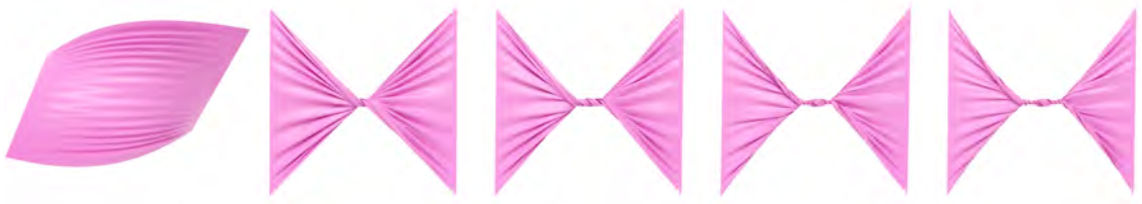


Figure 5.10: Twisting a square cloth for 8 circles, showcasing complicated self-collision with extreme contact force and friction. The model has 10K vertices and 19.6K faces. We use $r = 2$mm, a time step of 1/300s, and 10 iterations per step. The average/maximum computation time per time step is 21/30ms on the CPU and 0.9/1.5ms on the GPU.

generating strong material forces that compete with self-collisions. Our contact model effectively handles these strong deformations with frictional contact.

5.3.1.3 Coupled Cloth Simulation

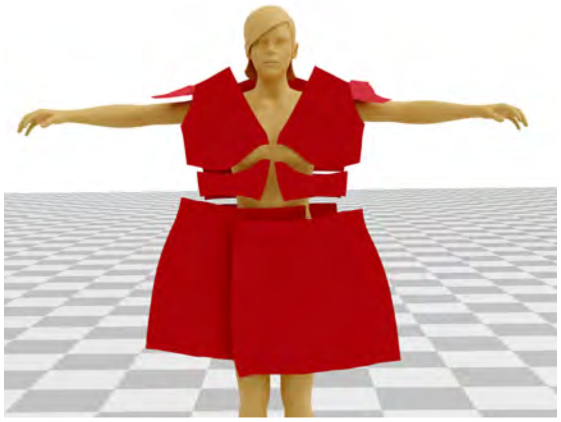
To test our method in a practical cloth simulation scenario, we conduct the same cloth simulation experiment as the one presented in the C-IPC paper [67], as shown in Figure 5.12. The cloth consists of 14 separate pieces stitched together using stiff zero-length spring constraints, see Figure 5.11a. We filter out collisions between the stitched primitives to ensure seamless stitching. For body-cloth contact, we use volumetric contact energy since the body motion is driven by skeleton animation rather than simulation, making it challenging to prevent penetration after updating the body's position, see Figure 5.11b. As a result, body-cloth penetration might occur during the simulation, but cloth-cloth penetration is prevented. In the C-IPC paper, the average computational time for a 0.04-second frame is reported as 24 seconds. In our tests, the same frame takes only 0.24 seconds on the CPU and 9.6 milliseconds on the GPU.

We also showcase a scenario of a robot manipulating a T-shirt. The robot's trajectory is pre-computed and we use the same scheme as in Figure 5.12 to handle the collision between the cloth and the robot. This experiment runs in real time and stably simulates the contacts between the robot and multiple layers of cloth.

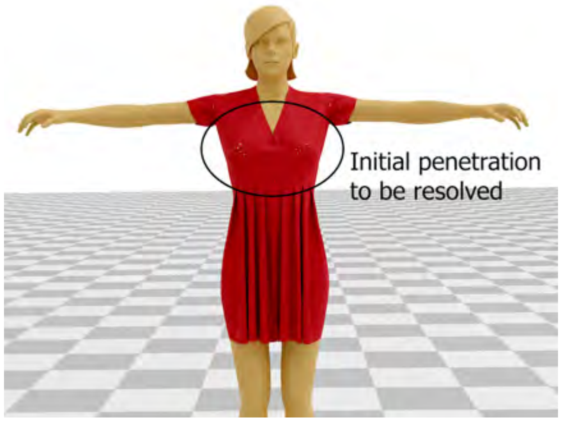
5.3.2 Yarn Level Simulation

We perform further stress tests of our method with the use of yarn level cloth simulations. Instead of simulating cloth as thin shells, we individually simulate each constituent yarn thread as codimensional rods. The behavior of the cloth is then the sum of contributions from yarn bending, twisting, stretching, contacts, and friction. This is traditionally difficult as even minor penetrations (pull-throughs) can cause significant unraveling of the yarn.

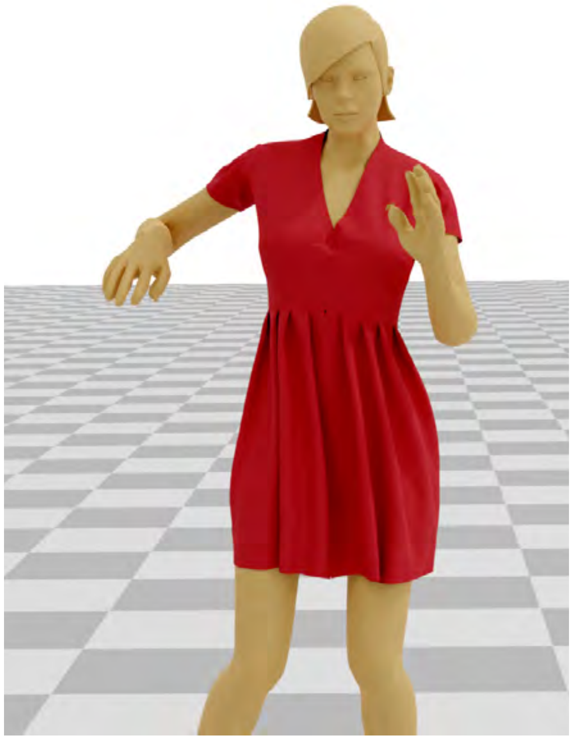
We model rod bending, twisting, and stretching with the use of Cosserat Rods similar to that proposed by kugelstadt16. To demonstrate the effect of our conservative bounds, we implement a penalty-energy-based collision handling method and compare it against our method. This method models contact as a quadratic energy and always takes the full step given by Newton's method. In the figure, we label this method as ”Newton”.



(a) Template



(b) Initial State



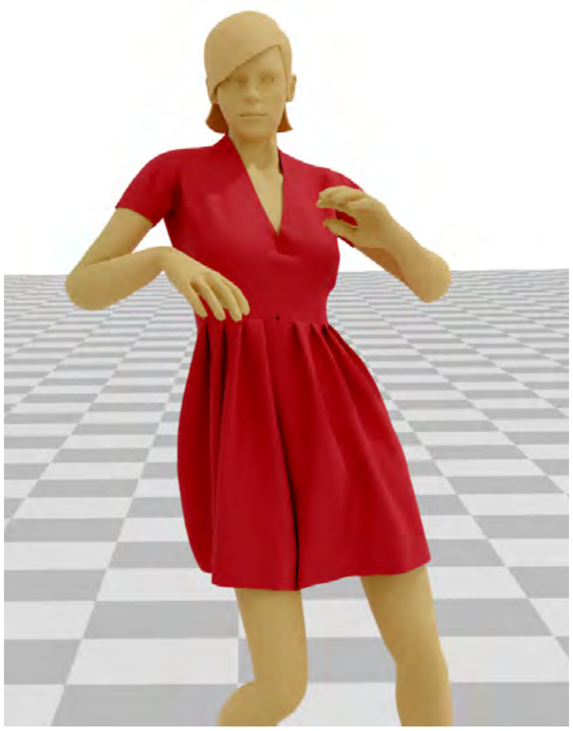


Figure 5.11: Simulating a dress on a moving human body. The character is driven by skeletal animation, with 12.8K vertices and 25.4K triangles. The dress model consists of 14 separate pieces as shown in (a), with 15.7K vertices and 29.4K triangles. We use $r = 2\text{mm}$, a time step of 1/200s, and 20 iterations per step. The average/maximum computation time per time step is 30/42ms on the CPU and 1.2/1.4ms on the GPU.

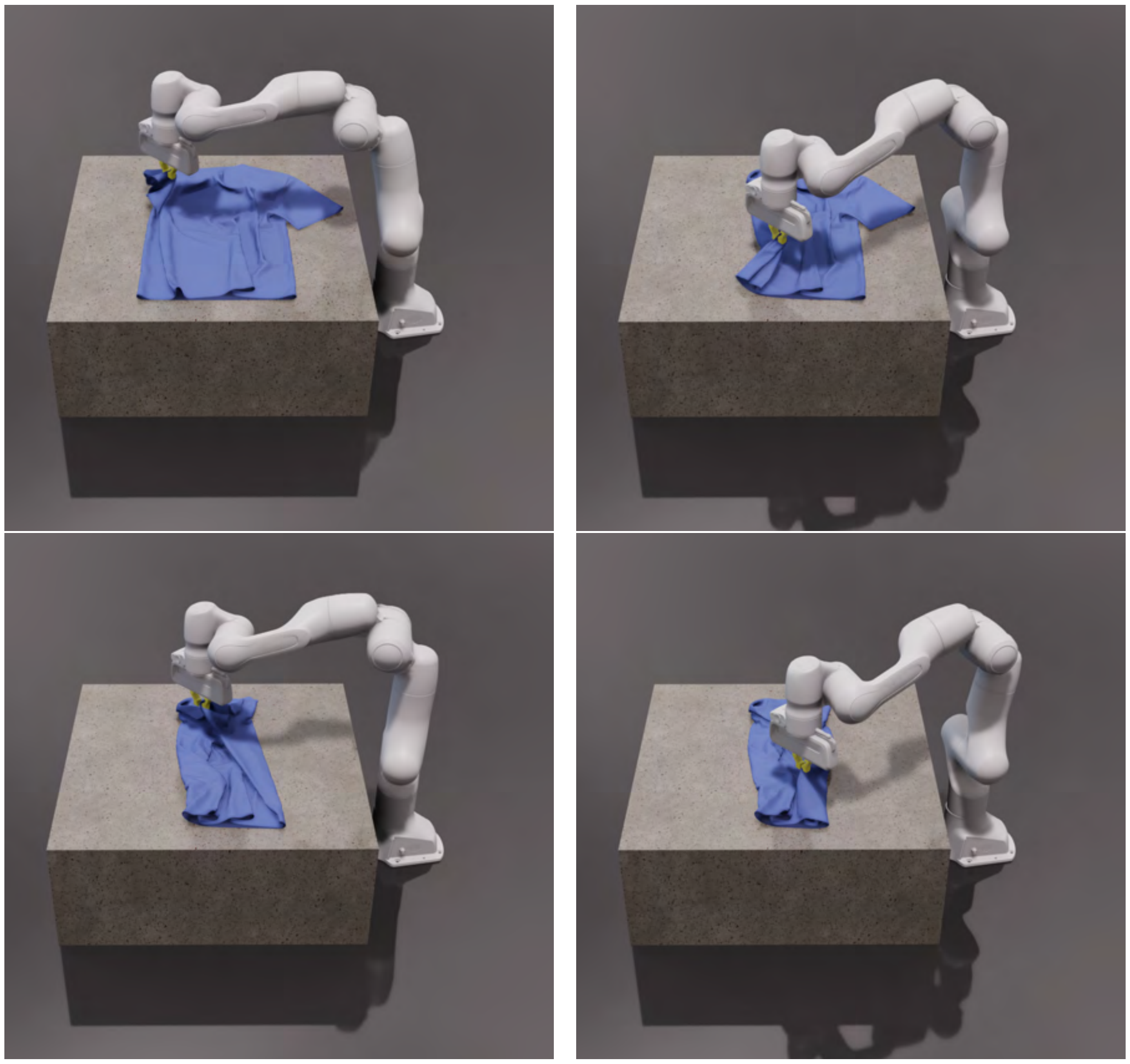


Figure 5.12: Simulating a robot manipulating a T-shirt. The robot's trajectory is pre-computed. The T-shirt mesh has 13.8K vertices and 27.4K triangles. We use a collision radius of 2mm and a time step of 1/600s for the simulation. The average/maximum computation time per time step is 1.8/2.2ms on the GPU.

In Figure 5.13, we pull and stretch the yarn cloth on two ends until it is taut with tension. Using penalty-energy-based collisions, the yarn threads phase through each other as they ultimately unravel catastrophically. Our method successfully preserves the yarn geometry even under extreme tension. Despite the yarn being taut enough to remain flat against gravity, no pull-through occurs.

In Figure 5.14, we clamp the square yarn cloth on two ends and twist one end in five full rotations. Penalty-energy-based collisions fail to prevent yarn penetration as the yarn threads crush and entangle into a knot. In contrast, our method is able to successfully return to the original state once the cloth is let go with no change to the yarn structure.

In both examples, we use a yarn cloth that is 40cm by 40cm with 3mm thick yarn under normal gravity. The yarn threads have a density of 1 gram per meter and a friction coefficient of 0.1. Both examples can run in real time on our setup.

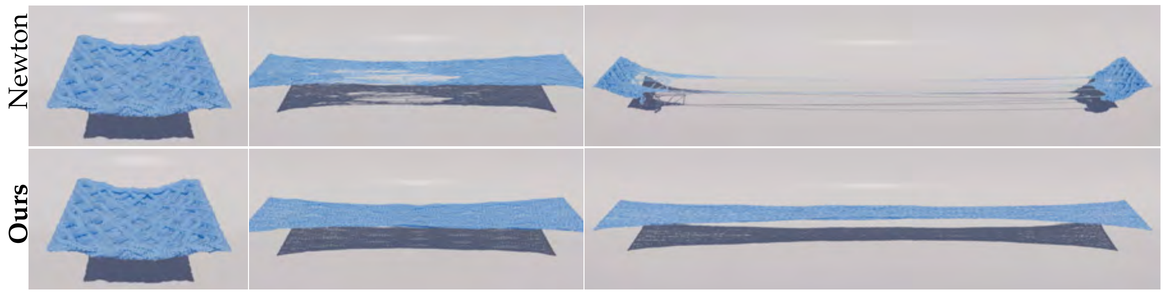


Figure 5.13: The yarn cloth is slowly pulled apart. Pure Newton is unable to prevent penetrations which cause catastrophic unraveling. In contrast, our method is able to maintain a penetration free state through-out.

5.3.3 Convergence

To evaluate OGC's ability to converge with different solvers, we plot the change in relative force residuals over iterations and computation time in Figure 5.15. The relative force residual is defined as:

$$e^{(i)} = \frac{\text{mean}(||\mathbf{f}_v^{(i)}||)}{\text{mean}(||\mathbf{f}_v^{*}||)} \tag{5.28}$$

where $\mathbf{f}_v^*$ is the initial force residual on vertex v and $\mathbf{f}_v^{(i)}$ is the force residual on vertex after iteration i.

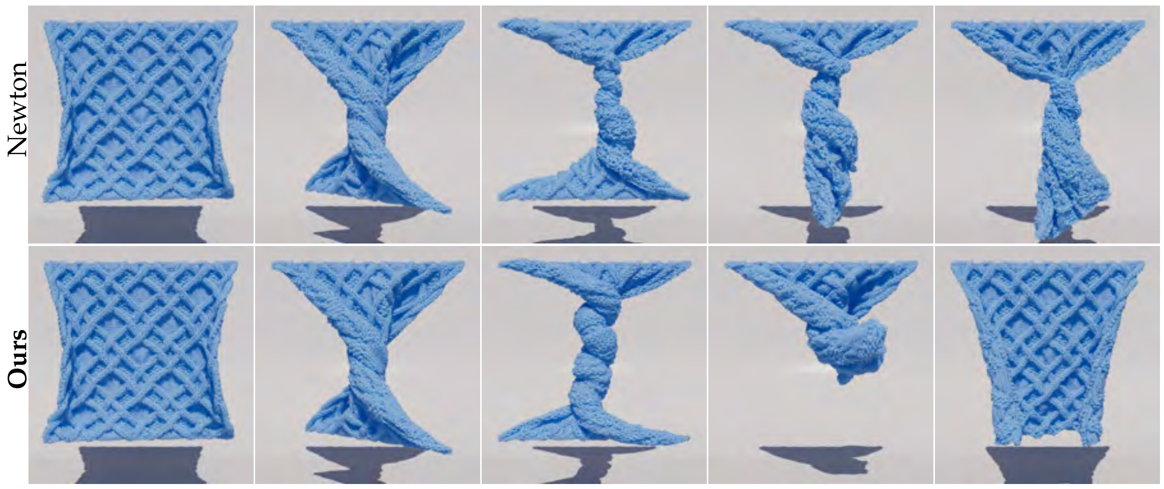


Figure 5.14: A square yarn cloth is clamped on its edges and twisted 5 full rotations. Pure Newton is unable to keep the yarn threads separate as they tangle. Our method is able to preserve the yarn structure despite the extreme deformation.

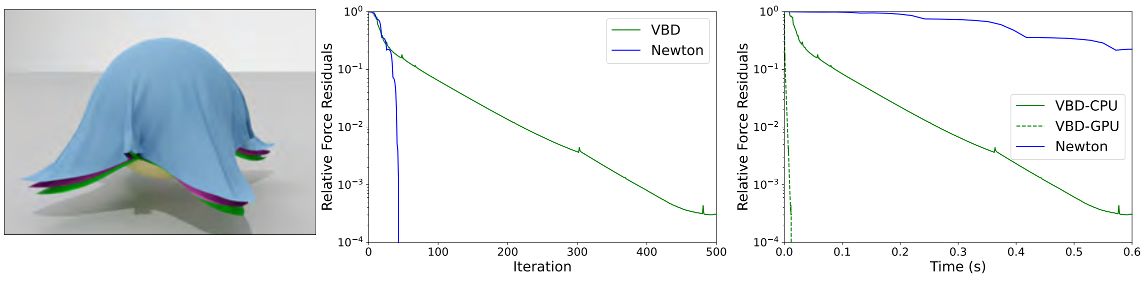


Figure 5.15: Convergence plot of Newton's method and VBD-based OGC for simulating three clothes dropping on a sphere at the given step, with 14.7K vertices, 28.6K triangles, and a time step of $1/100s$. The graphs show relative force residuals change over iterations and computation time.

Both VBD and Newton's method can reduce the mean force residuals to less than 1e-4, which is the lowest error achievable with single precision. We run VBD for 500 iterations and Newton's method for 50 iterations. The spike in VBD's curve corresponds to the application of contact detection. Since we are not performing a line search for VBD, the force residuals experience a brief spike after updating the contact set through a new DCD. However, VBD quickly recovers from this with just a few iterations and continues to reduce the error.

In terms of convergence speed, Newton's method converges faster in terms of iterations, reaching numerical convergence at the 46th iteration. However, since each iteration of Newton's method is much more computationally expensive and requires a line search to ensure stability, it lags far behind VBD in terms of computational time. Collision detection accounts for approximately 3% of the computational time when using Newton's method and 10% when using VBD. This experiment demonstrates that the contact force defined by OGC can converge very efficiently with various solvers, using minimal contact detections.

5.3.4 Quantitative Comparison to Incremental Potential Contact

We compare Newton's method based OGC and VBD based OGC with IPC with incremental potential contact (IPC). The results are visualized in Figure 5.16. We used the open-sourced implementation of Codimensional-IPC (C-IPC) to generate results. We evaluate the computational time and the total number of collisions at each step.

We can see that IPC's use significantly more collision detections because IPC require more than two collision detections at each iteration: one CCD to cull the global step size and one DCD per energy evaluation in the line search process. This is not the case for OGC-based methods, because OGC does not require any CCD, and DCD is only necessary when points reach their conservative bounds, which does not happen at every iteration, especially in the later stages of the optimization process. In these later stages, the optimizer (both Newton's method and VBD) provides very small step sizes, so it usually takes several iterations for the accumulated displacements to exceed the conservative bounds. As a result, OGC-based methods require significantly fewer collision detections, leading to much lower computational time compared to IPC.

This efficiency is particularly advantageous for VBD-based OGC. Since VBD tends to generate smaller steps than Newton's method, it is less likely to exceed the conservative bounds, allowing the simulation to fully leverage VBD's output. Newton's method, on the other hand, tends to provide larger optimization steps, which supposedly can lead to a more significant reduction in error. However, much of this potential gain can be lost due to conservative bound culling, resulting in wasted computation. Overall, on the CPU, VBD-based OGC is more than about 128 times faster than IPC, while Newton's method-based OGC is 9.2 times faster than IPC on average.

VBD-based OGC is more advantageous over IPC on GPU. We compare the GPU implementation of VBD-based OGC with GIPC [192], the state-of-the-art GPU variant of IPC.

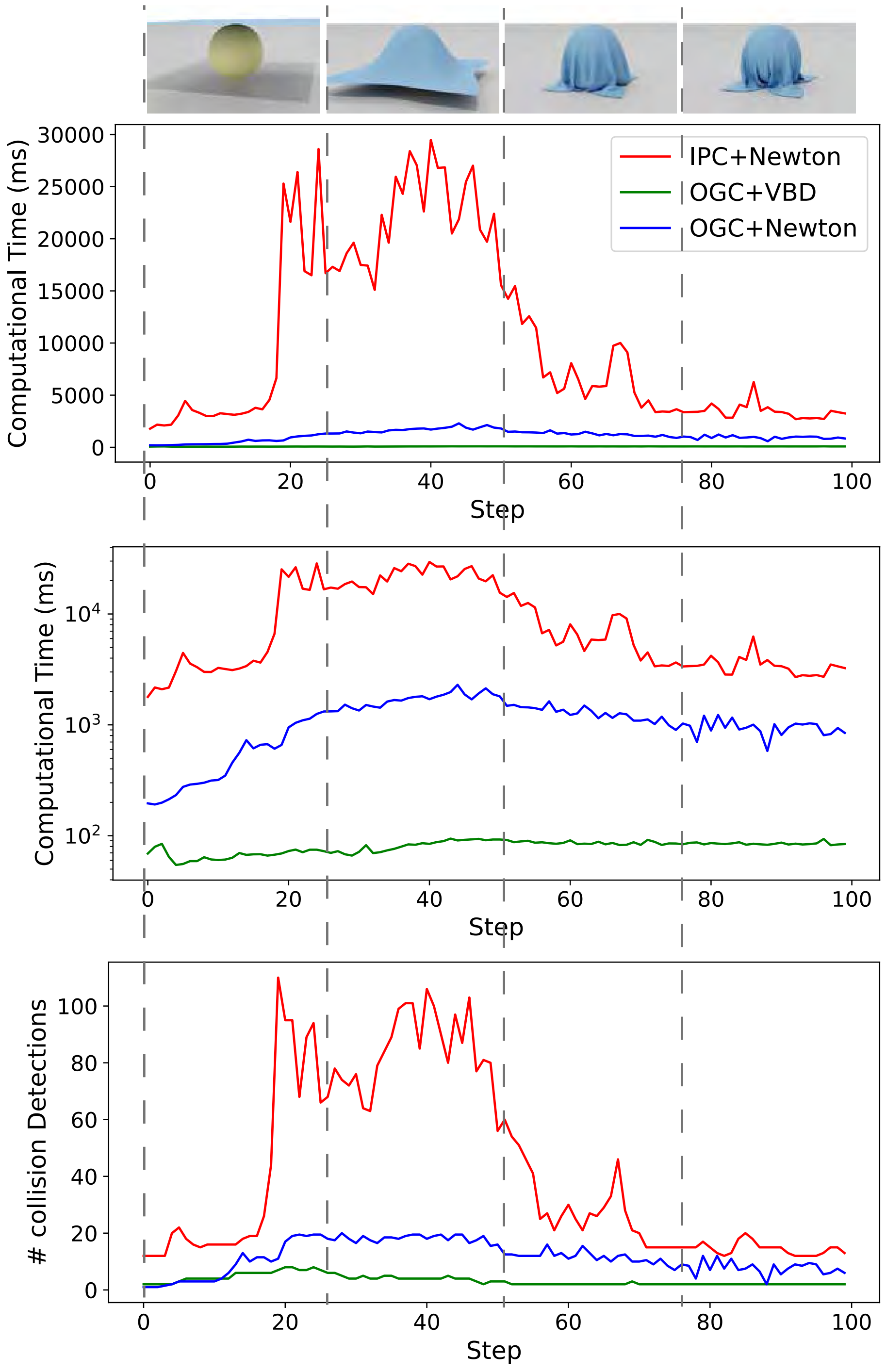


Figure 5.16: We compare Newton's method-based OGC and VBD-based OGC with IPC by simulating a cloth dropping onto a fixed sphere. The cloth has 4.9K vertices and 9.5K triangles. The simulation is run with a time step of 1/100s for 100 steps. The four images in the first row show the state of the simulation using IPC at steps 0, 25, 50, and 75, respectively. We use a contact radius of 5mm for OGC and allow IPC to automatically control the contact radius. From top to bottom, the first figure illustrates the computational time at each step for each method, the second one is the same as the first one but in a logarithmic scale, and the third chart shows the number of collision detections used at each step.

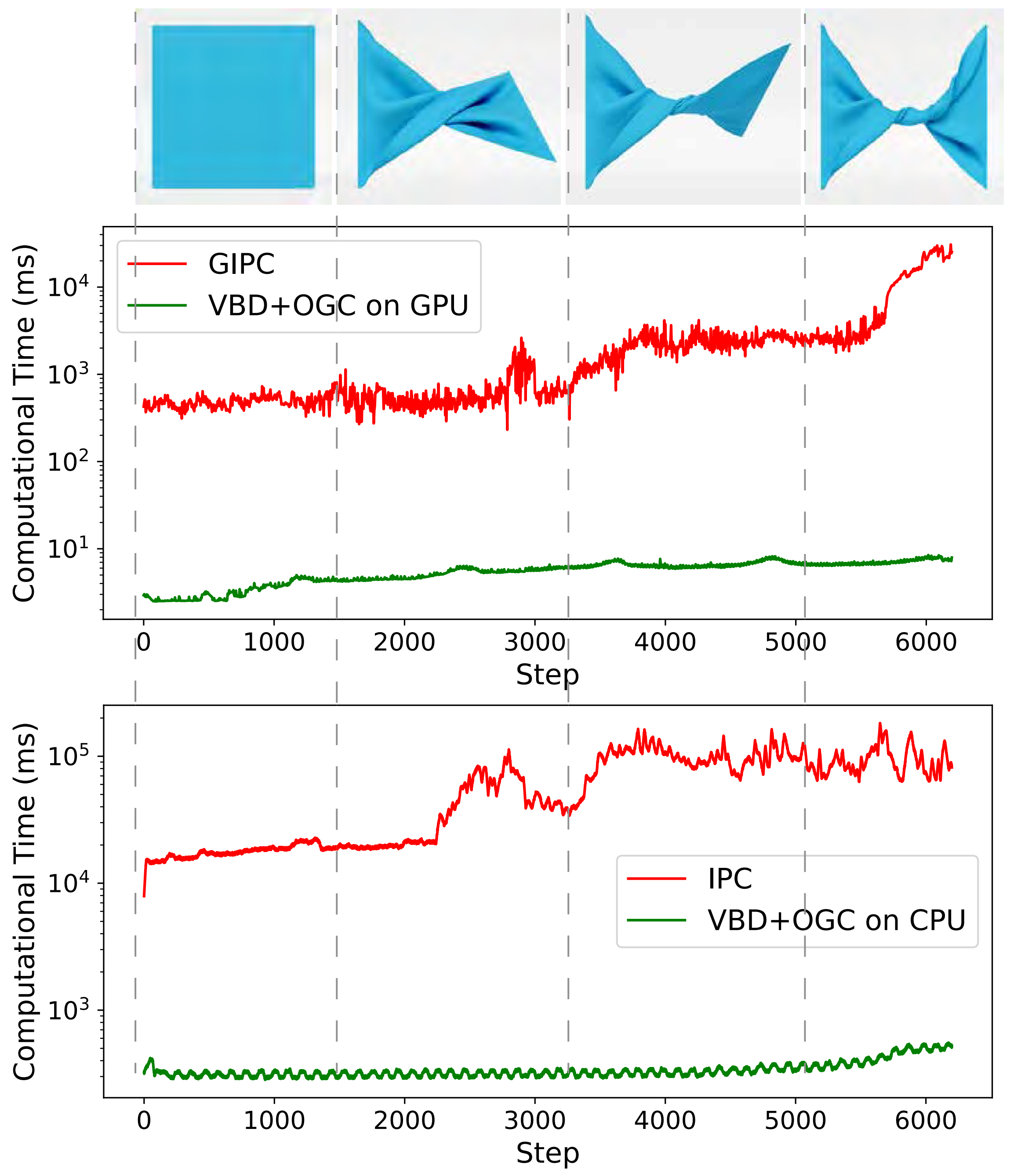


Figure 5.17: We compare the GPU implementation of VBD-OGC with GIPC [192], and the CPU implementation of VBD-OGC with IPC, by replicating the volumetric mat twisting experiment presented in the IPC [2] and GIPC [192] papers. The mesh has 15.3K vertices and 46.8K tets. The simulation is run with a time step of 1/240s. The four images in the first row show the state of the simulation using IPC at 0s, 4s, 8s, and 12s, respectively. The middle row compares the runtime of each step in the GPU implementation of VBD-OGC against GIPC, while the bottom row compares the CPU implementation of VBD-OGC with IPC. We use a contact radius of 2 mm for OGC and allow GIPC and IPC to automatically control the contact radius. We plot the time consumption at each step in the chart.

We used the open-sourced implementation of GIPC to generate results. For testing, we simulate the twisting of a volumetric mat at an angular velocity $\frac{\pi}{2}$ by 16 seconds and evaluate the computational time at each step. The results are visualized in Figure 5.17.

The average frame time for GIPC is 1893ms, while VBD-OGC requires only 5.51ms per step on average, making it 343 times faster and capable of achieving near real-time performance, even under intensive collisions and deformations. Furthermore, VBD-OGC demonstrates significantly more stable performance, with the maximum step time reaching only 8.5 ms. In contrast, GIPC's maximum frame time exceeds 20 seconds, occurring at the end of the simulation when the object experiences extensive self-contact, leading to minimal optimization progress in each iteration. This comparison highlights OGC's suitability for real-time simulations due to its consistent and efficient time consumption.

We also present the results of the same experiment using the CPU implementations of VBD-OGC and IPC in the bottom row of Figure 5.17. For IPC, we use its officially released implementation. While IPC takes an average of 61.18 seconds per time step, the CPU version of VBD-OGC completes each step in just 0.540 seconds on average, achieving a 133× speedup. This demonstrates that our method's advantages do not only come from better parallelism.

5.3.5 Qualitative Comparison to Incremental Potential Contact

5.3.5.1 Work with Large Contact Radius

We compare the compatibility of the IPC and OGC contact models with a large contact radius by simulating a cloth twisted by half a circle using both methods. The final states of the simulations using the IPC and OGC are shown in Figure 5.1a and Figure 5.1b, respectively. The cloth consists of a 200×200 regular grid with each side measuring 1 meter, resulting in a 0.5mm minimal distance between vertices. As shown in Figure 5.1a, the IPC model produces severe artifacts caused by non-orthogonal forces from neighbors and other points, including vertex bulging and oscillations. In contrast, the OGC model handles the large contact radius robustly, producing stable and natural contact results. Please see the supplementary video for a more thorough side-by-side comparison.

5.3.5.2 Numerical Damping

IPC is known to exhibit severe numerical damping artifacts when convergence is insufficient. This issue is demonstrated in Figure 5.18, where a square cloth is simulated dropping onto a small sphere, causing self-contact. In this experiment, we use a time step of dt=1/500 but limit the solver to only one iteration per step. Once self-contact occurs, IPC quickly loses nearly all momentum, resulting in a slow-motion effect. This happens because the self-contact restricts the optimization step size, allowing only minimal movement per step and effectively dissipating velocity.

In contrast, the OGC model, with its conservative initialization scheme and per-vertex-based displacement bounds, preserves the momentum for most vertices, producing simulations with significantly more dynamics. While neither IPC nor OGC achieves full numerical convergence under such limited iterations, the OGC model generates results that are far more visually plausible.

5.4 Discussions

Offset Geometric Contact is a contact model intended to achieve orthogonality of normal contact force. However, on a discrete surface, orthogonality and continuity of contact force cannot both be achieved. As illustrated in Figure 5.19a, when a point moves along the black trajectory, it is subject to discontinuous contact forces, particularly upon entering the facet's block from the open boundary. At that moment, it suddenly experiences a non-zero contact force from the facet. Note that this discontinuity only occurs at the open boundary on the concave side of the faces, not at the closed boundary, where the contact force is zero. The more concave the area is, the more likely this issue is to arise, because it occurs in the overlapping area of two adjacent face's blocks. However, the more concave the area is, the less likely it is for a point to enter the narrow space between faces, which helps mitigate the problem. In theory, this discontinuity can lead to instability or slow convergence, though we did not observe any such issues in our experiments.

Our technique for achieving penetration-free simulation is significantly more efficient in scenarios with intensive collisions. However, in cases with few collisions and large velocities, it may lag behind the CCD-aware line search employed by IPC. As visualized in Figure 5.19b two mass points falling freely under gravity in parallel: IPC's technique

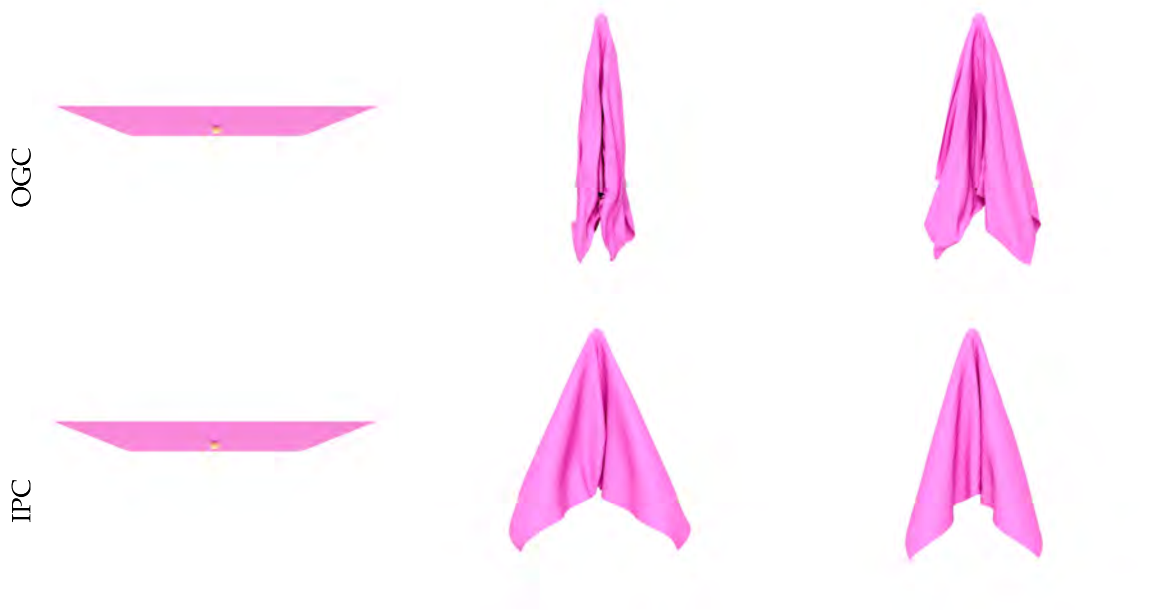
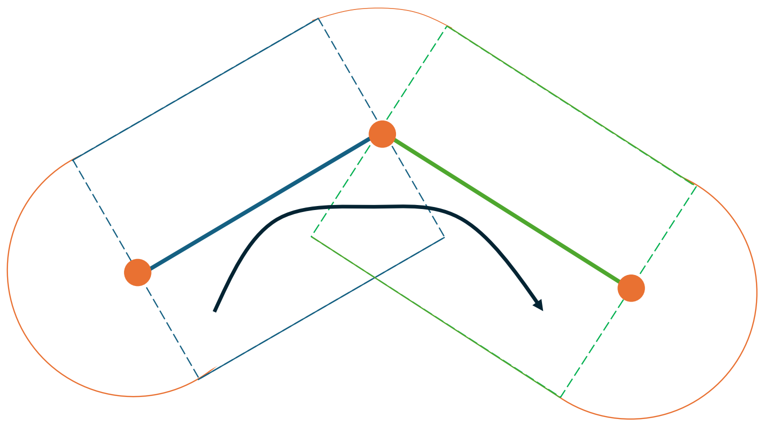
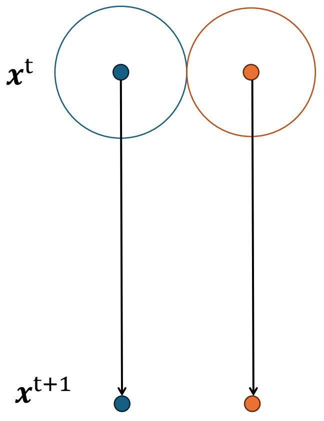


Figure 5.18: Comparing our method (OGC+VBD) with IPC in the low iteration count setup. In both of those experiments we use a time step of 1/500s and only 1 iteration per step. The cloth has 4.9K vertices and 9.5K triangles. The top row is the results of OGC+VBD and the bottom row is the results of IPC. From left to right, each column visualizes the simulation state at frame 0, 50 and 80.



(a) Discontinuity



(b) High velocity

Figure 5.19: Illustration of our method's limitations. (a) A point's trajectory that results in a discontinuous contact force. The thickened line and dots represent the face, and the colored lines indicate the boundaries of the face's block with corresponding color. Solid lines denote closed boundaries, while dashed lines denote open boundaries. The solid black line visualizes the trajectory. (b) Two mass points dropping with high velocity from current position $\mathbf{x}^t$ to the next time step's position $\mathbf{x}^{t+1}$, the circle visualizes the conservative bound given by our method and the black arrow visualizes their trajectory within this time step.

can apply the full step in a single iteration because their trajectories do not intersect. In contrast, our approach limits their motion to less than $\frac{d}{2}$, where d is the distance between the two points. This restriction can necessitate more iterations for convergence, and the issue worsens as velocity increases.

Nevertheless, this limitation also suggests a promising direction for future improvement. Potential strategies include intelligently switching among various penetration-free techniques or incorporating vertex displacement directions to establish tighter bounds.

We have presented offset geometric contact, an efficient contact model that allows for penetration-free simulation of codimensional objects, significantly reducing the stiffness of contact forces and increasing the efficiency of the simulation pipeline. By offsetting each face in its normal directions, our formulation ensures normal contact forces remain orthogonal, allowing for a larger contact distance and thus mitigating the stiffness problem. Instead of continuous collision detection (CCD), we compute a local maximum displacement bound for each vertex in parallel, adding negligible overhead. This local approach, combined with a fully parallel solver like Vertex Block Descent, enables real-time, large-scale simulations on GPUs. Our experiments show that this method can be more than two orders of magnitude faster than IPC-based simulations and maintain near-constant computational cost by using a fixed iteration count, making penetration-free simulation feasible for a broader range of applications.

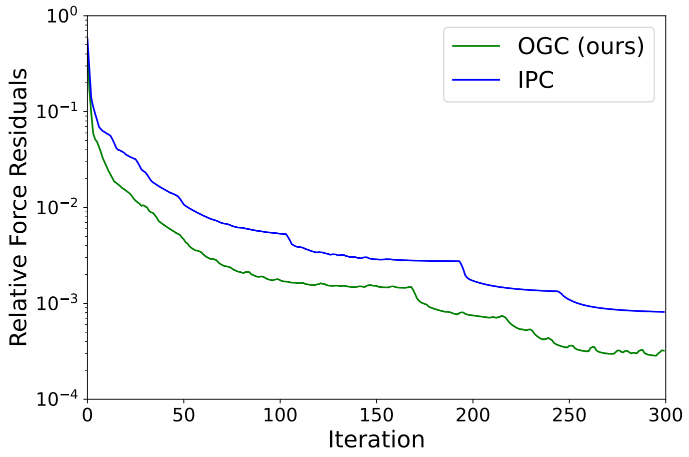
Our results demonstrate that the proposed method effectively handles highly complex simulation scenarios (Figure 5.8), maintains stability under extreme stress tests (Figure 5.9, 5.10, 5.13, and 5.14), and exhibits fast convergence (Figure 5.15). In addition, we present an efficient implementation of our contact model integrated with the VBD integrator, leveraging block-level operations to maximize parallelism and efficiency.

5.4.0.1 Comparing Activation Functions

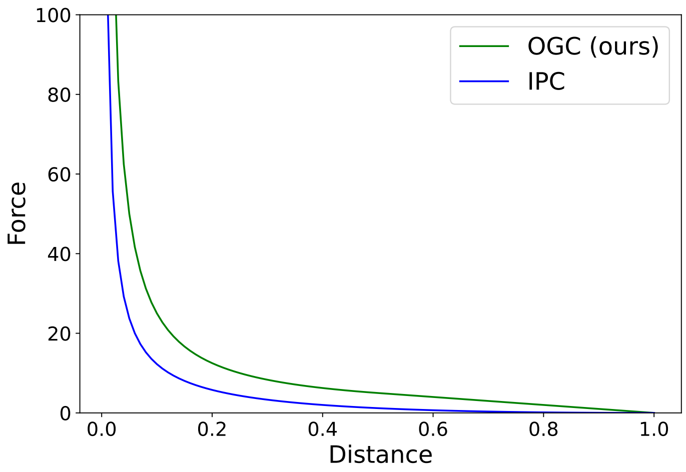
To demonstrate the effectiveness of our activation function, we conduct an ablation test, with results visualized in Figure 5.20. In this experiment, we run two simulations with our and IPC's activation function, simulating a piece of cloth dropped onto a sphere on the ground, with a time step of 1/100s, collision stiffness $k_c = 1e4$, and VBD solver. We first simulate 40 time steps, ensuring each step reaches numerical convergence. The

(a) OGC (ours)

(b) IPC



(c) Convergence Plot by Iterations



(d) Force-Distance Relationship for OGC (ours) and IPC

Figure 5.20: Comparing our activation function Equation 5.17 with IPC's activation function. The results are measured on the 40th time step of simulating a piece of cloth falling onto a sphere. The cloth has 4.9K vertices and 9.5K triangles, and we use a time step of 1/100s. The state of simulation at the selected step using OGC and IPC's activation function is visualized in (a) and (b), respectively. Panel (c) plots the convergence of relative force residuals by iteration, and panel (d) shows the force-distance relationship of the two functions.

state of the simulation at the 40th step is visualized in Figure 5.20a and Figure 5.20b. We can see that the simulations with two activation functions provide visually identical results. Furthermore, we plot the change in relative force residuals (Equation 5.28) over iterations of the 40th step in Figure 5.20c, where the simulation using our activation converges approximately 2x faster than the one using IPC's activation. At last, we plot the force-distance relationship of two functions in Figure 5.20d, where we set the contact radius and collision stiffness k_c of both of those activations to be 1. Our activation shows a smoother transition from 0 to infinity and exhibits less stiff behavior. In fact, at the state visualized in Figure 5.20a and Figure 5.20b, the condition number of the system Hessian of the simulation with our activation is 5 times smaller than that using IPC's activation.

CHAPTER 6

CAPTURING DETAILED DEFORMATIONS OF NON-RIGID OBJECTS

In this section, we propose a system capable of capturing the detailed deformation of non-rigid bodies directly, without the need for a registration process. The pipeline of the system is visualized in Figure 6.1. This system captures temporally consistent data that describes the trajectories of points on real-world objects, which can be used to optimize physics parameters using a differentiable simulation framework. Additionally, we introduce a technique to directly drive the simulation with the captured data by incorporating it into Equation 2.8 as bilateral constraints. This technique is particularly useful for filling in missing observations in our data. We demonstrate our framework in the context of human body capture, but this capture system can be easily extended to support other non-rigid bodies, such as cloth or elastic materials.

6.1 System Summary

In most real-world images, the human body is occluded by clothing, making precise body measurements difficult or impossible. A significant amount of previous work focuses on approximate but robust pose estimation in the wild [124, 129]. However, a small muscle twitch or the speed of breathing may contain signals that are critical in certain contexts, e.g., in the context of social interactions, minute body shape motion may reveal important information about the person's emotional state or intent [136]. Detailed human body measurements are highly relevant also in orthopedics and rehabilitation [193], virtual cloth try on [194, 195], or building realistic avatars for telepresence and AR/VR [196, 197].

When precise measurements are needed, prior work utilized either 1) reflective markers attached to a motion capture suit or glued to the skin [96], or 2) painting colored patterns on the skin [167]. The traditional reflective ("mocap") markers present certain limitations. Because all of the markers look alike (Figure 6.2a), marker labeling relies strongly on temporal tracking and high frame-rate cameras. However, robust marker labeling is a hard problem [98] which often requires manual corrections, especially for markers that have been occluded for too long. The difficulty of this problem grows with the number of markers [96], thus sparse marker sets are most common in the industry. Sparse marker sets are sufficient for fitting a low-dimensional skeletal body model, but not for capturing the details of flesh deformation or motion due to breathing.

To capture moving bodies with high detail, the DFAUST approach [167] starts by geometrically registering a template body model to 3D scans [165, 131] and then uses colored patterns on the skin to obtain high-accuracy temporal correspondences via optical flow. These colored patterns serve a similar purpose as the checkerboard-like corners on our suit, i.e., they enable precise localization of points on the surface of the body. The key difference of our approach is that our suit contains also unique two-letter codes adjacent to each corner, allowing us to label the corners directly by recognizing the codes. This is not possible with the DFAUST's patterns, because they are self-similar, created by applying colored stamps to the skin. Instead, the DFAUST approach relies on the initial geometric registration and temporal tracking, which can suffer from error accumulation and may lead to incorrect local minima in more challenging poses or fast motions. The DFAUST dataset contains a variety of high-detailed human body animations, but is restricted to upright standing-type motions. In contrast, we demonstrate captures of a wider variety of motions, including gymnastics exercises, yoga poses or rolling on the ground.

Our new motion capture method was enabled by recent advances in deep learning and high-resolution camera sensors. The key idea is to use a new type of motion capture suit with special fiducial markers, consisting of checkerboard-like corners for precise localization and two-letter codes for unique labeling. Our localization and labeling process is very robust, because it does not rely on temporal tracking or any type of body model; in fact, our approach succeeds even if only a small part of the body is visible in the image; see Figure 6.1. A similar advantage exists also in the temporal domain. Because our localization and labeling approach can process each image independently, there are no issues due to occlusions and dis-occlusions which complicate traditional temporal tracking in both marker-based as well as marker-less methods.

Even though the automatic localization and labeling are very robust, achieving this functionality is non-trivial, because our methods need to be robust against significant stretching of the tight-fitting suit as well as projective distortion. Fortunately, checkerboard-like corners remain to be checkerboard-like even despite significant stretching of the suit. We apply three convolutional neural networks combined with geometric algorithms. The first of our convolutional neural networks (CNNs) is a corner detector which localizes all of the corners in an input image (4000×2160). To rectify the distortion of the two-letter codes inside the white squares, we connect candidate four-tuples of corners into quadrilaterals (quads) and apply homography transformation which rectifies both suit stretching as well as projective distortion. Another CNN called *RejectorNet* then performs quality control and legible and upright-oriented codes. The remaining codes are passed to *RecogNet* which reads the characters in the two-letter code. Because the orientation of our codes is unique (we avoided symmetric symbols such as "O" or "I" as well as ambiguous pairs like "6" and "9"), recognition of the code allows us to uniquely label each of the adjacent corners; see Figure 6.2c.

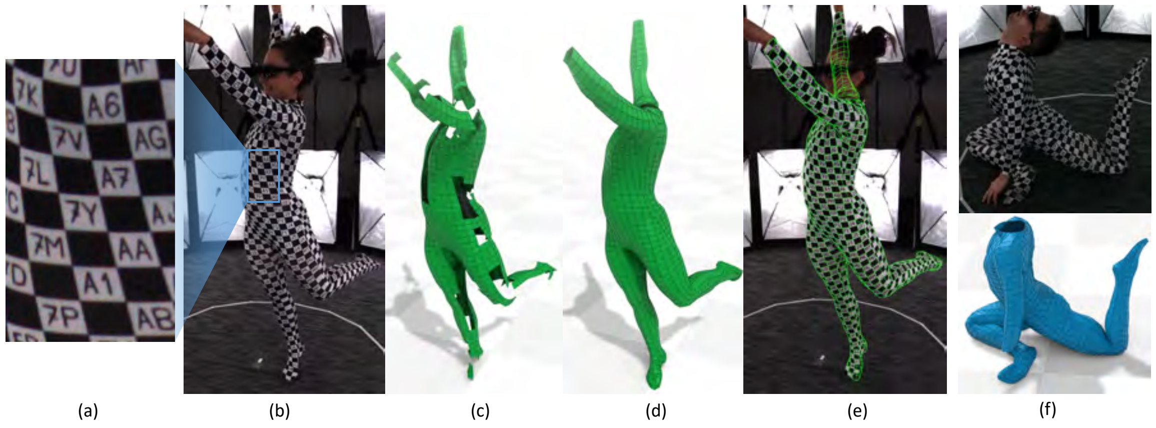


Figure 6.1: Visualization of our whole pipeline. (a) Our novel motion capture suit with a special pattern; (b) An example input image from our multi-camera capture system; (c) Raw 3D reconstruction of **labeled** corners from the suit (raw data, no body model was used); (d) The result after interpolating missing observations (here, a body model is used); (e) Our reconstructed mesh aligns very closely with the original input images; (f) Our method works even in uncommon poses with many self-occlusions, such as this yoga pose.

The labels of our corners establish correspondences both in time and in space, i.e., between individual cameras in our multi-view system, which means that we can easily triangulate the 2D corner locations into 3D points. However, the 3D reconstructed (triangulated) points will inevitably miss observations due to self-occlusions and the limited number of our cameras; see Figure 6.1c. To fill (interpolate) these missing observations, we start by fitting the STAR model [135] and then refine it for each of our actors using a point trajectory from a calisthenics-type motion sequence captured using our method. This refinement ensures that we obtain the best possible low-dimensional model for each of our actors, since high quality is our main objective. We use this refined body model to interpolate the missing corners in the rest pose, resulting in the final mesh without any holes; see Figure 6.1d.

Our goal was to make each two-letter code as small as possible, so we can recover the highest possible number of points on the body. We created our special capture suits in two sizes, one "medium" (with 1487 corners) and one "small" (with 1119 corners) and we captured three actors: one male and two females (two of the actors used the "medium" suit).

We evaluated both the geometric accuracy through reprojection error of 3D reconstruction, and the quality of the temporal correspondences by computing the optical flow between the synthetic image and the real image. Results show 99% of our reconstructed points has a reprojection error of less than 1.01 pixels, and 95% of the pixels on the optical flow have a optical flow norm of less than 1.2 pixels. In our camera setup, 1 pixel approximately converts to 1mm on a person 2 meters away from the camera.

6.2 3D Reconstruction of Labeled Points

6.2.1 Suit

To create our special motion capture suit, we started by purchasing a tight-fitting unitard, originally intended for dance or performing arts. Fortuitously, one of the manufacturer-provided patterns was precisely the black-and-white checkerboard texture reminiscent of computer vision calibration boards (in fact this provided some of the original inspiration for this project). We purchased two suits, one "medium" and one "small" and augmented them by writing codes into the white squares using a marker pen. The medium suit contains 1487 corners and 625 two-letter codes; the small suit has 1119 corners and 456 codes. For our two-letter codes, we only used symbols whose upright orientation is unique and non-ambiguous, specifically: "1234567ABCDEFGJKLMPQRTUVY".

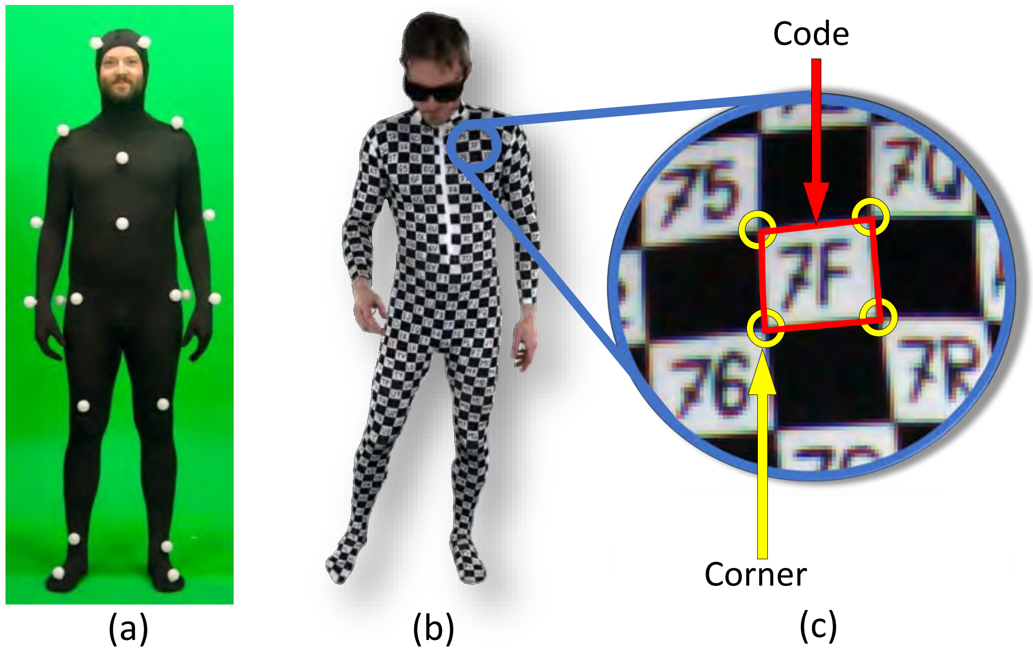


Figure 6.2: Comparison between traditional mocap markers and our markers. (a) A classical motion capture suit with reflective markers. (b) The design of our motion capture suit with fiducial markers. (c) Each or our markers consists of checkerboard-like corners and codes.

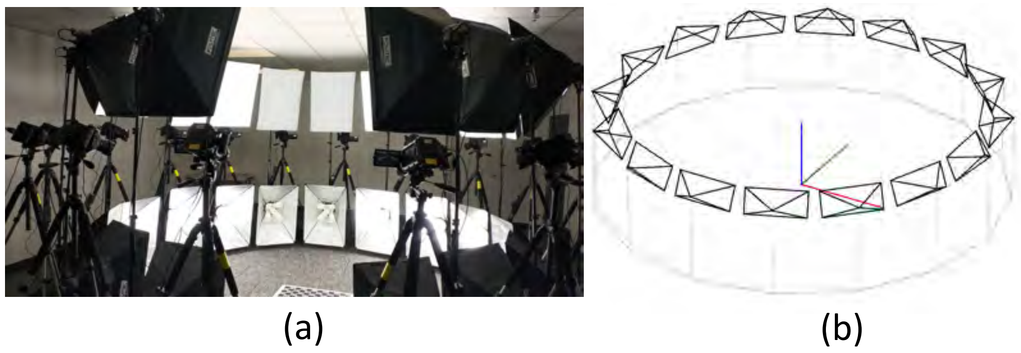


Figure 6.3: Our camera setup. (a) A photo of our 16 camera setup. (b) The cameras form a circle surrounding the capture volume.

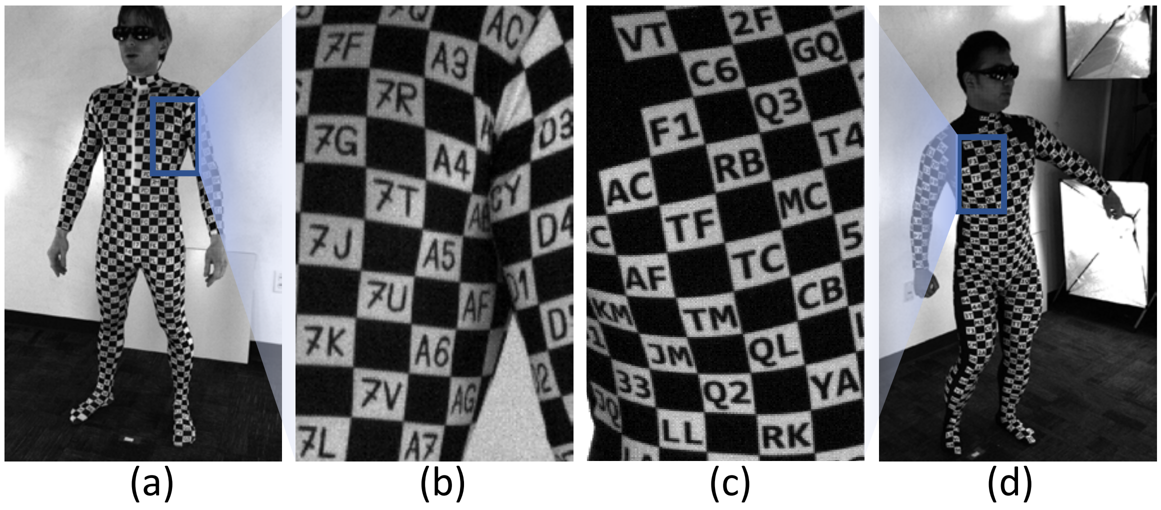


Figure 6.4: Comparison of (a, b) hand-drawn and (c, d) printed suits.

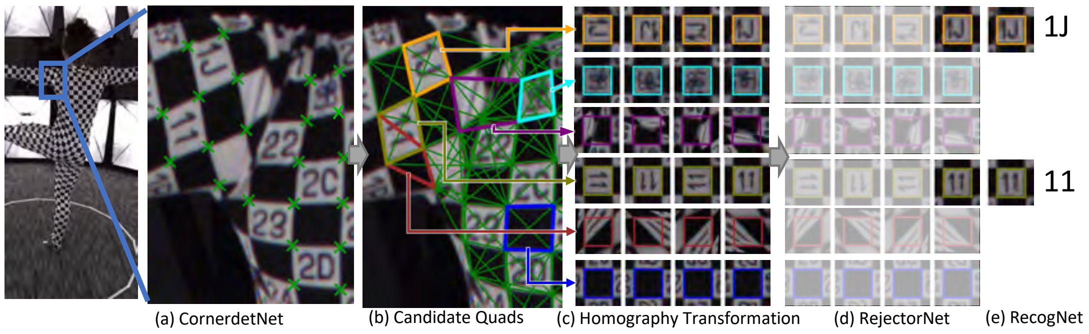


Figure 6.5: Corner localization and labeling pipeline: (a) *CornerdetNet* CNN detects and localizes our suit corners; (b) Four-tuples of corners are connected into candidate quads; (c) homography transformations using all four possible orientations (we selected a few quads and outlined them with different colors for illustration purposes); (d) *RejectorNet* CNN: a binary classifier accepting only valid white squares with upright oriented two-letter codes; (e) *RecogNet* CNN recognizes the two characters in the valid codes.

An alternative to hand-drawn suits is to use a cloth printing service, capable of printing an arbitrary high-resolution image on textile. However, this approach is more complex because it requires us to design sewing patterns, ship them to the printing service, then cut and sew the printed textile into a suit. We have explored this approach, using a computer font that is quite different from our handwriting; see Figure 6.4c, d. This experimental suit contains 1473 corners and 618 two-letter codes, but it contains large black areas due to suboptimal sewing patterns.

6.2.2 Camera System

Our multicamera setup contains 16 standard (RGB) cameras arranged into a circle surrounding the capture volume (Figure 6.3b). Each camera captures 4000×2160 images at 30 FPS in RAW format. The camera shutters are synchronized via genlock with negligible synchronization error, which means that human motion is captured as if "frozen" in time. Surrounding the capture volume, we positioned 32 softboxes that generate uniform diffuse light. The bright light allows us to use a small aperture and very fast $0.5ms$ shutter speeds, guaranteeing sharp images even with the fastest human motions. The cameras are calibrated by waving a traditional calibration checkerboard in from of them. The intrinsic and extrinsic camera parameters are calibrated using the well-established method [198] for which we use OpenCV's checkerboard corner detector for rigid calibration boards [199]. Next, the camera parameters and the 3D checkerboard corner positions in the world coordinates are further refined using bundle adjustment [200]. We use the Levenberg-Marquardt algorithm and the Ceres library [201].

6.2.3 Image Processing Pipeline

The calibrated cameras generate sequences of images, which are processed by our pipeline outlined in Figure 6.5. We start by detecting checkerboard-like corners in the input image with sub-pixel accuracy (Figure 6.5a, Section 6.2.4). Next, we need to uniquely label the detected corners by recognizing the adjacent two-letter codes. Because the codes are written in the white squares surrounded by four corners, we generate candidate quadrilaterals (quads) by connecting four-tuples of corners. Only a few four-tuples of corners correspond to the white squares, but it is okay to generate a quad that does not correspond to a white square, because it will be discarded later; hence we use the term "candidate quads"; see Figure 6.5b. Since the quads are generated by connecting four corners, we naturally have correspondences between the corners and the quads. The candidate quads are rectified by mapping them into a regular square using homography (Figure 6.5c, Section 6.2.5) to remove suit stretching and perspective distortion, and then passed as input to *RejectorNet* which performs quality control and checks whether the quad actually corresponds to a white square with a code (Figure 6.5d). The *RejectorNet* also ensures the correct upright orientation of the code. The images accepted by *RejectorNet* are then passed to *RecogNet*, which reads the two-letter code that finally enables us to uniquely label each corner (Figure 6.5e).

We would like to point out that our method is local by design, i.e., each stage of the pipeline works with small patches of the input image. This gives us a several advantages: *a*) Our method is capable of extracting reliable geometric information of the human body and - crucially - correspondences even from a small patch of the suit. This makes our method very robust to occlusions or partial views of the human body e.g., due to zoomed-in cameras. *b*) By decomposing the suit into small quads and undistorting them using homography, we can counteract much of the projective distortions and suit stretching (see Figure 6.5c), simplifying the learning task. *c*) The CNN quad classifier includes a quality control mechanism, rejecting white squares with dubious quality and further improving the robustness of our method.

6.2.4 Corner Detector

The corner detector's task is to detect and localize all checkerboard-like corners in the input image. This task is non-trivial because there are corner-like features in the background, the suit stretches along with the skin and there are significant lighting variations.

Our corners have two key properties: *a*) The corners are sparsely and approximately uniformly distributed on the suit; *b*) The corners are defined locally, i.e., a small image patch is enough to identify and localize a corner. We divide our input image into a grid of 8×8 cells with the assumption that there could be at most one checkerboard-like corner in each 8×8 cell, and apply *CornerdetNet* CNN to detect and localize a checkerboard-like corner from each cell separately. The design of *CornerdetNet* is inspired by single shot detectors [202, 203], which perform prediction and localization simultaneously. Please see

Supplementary Material A.1 for more details about *CornerdetNet*.

6.2.5 Corner Labeling and 3D Reconstruction

At this point we have detected typically several hundreds of corners in each input image. The next step is to read the codes and link them to the corners, which will give us a unique label for each corner. The deformations of the suit and its codes can be significant, including not only projective transformations, but also stretching and shearing because the tight-fitting suit is highly elastic. First we have to detect the white squares with two letter codes. We know that each such white square is surrounded by four corners. Therefore, we generate quadrilaterals (quads) by connecting four-tuples of corners. In theory we could connect any four-tuples of corners into a quad, but in practice we can immediately discard concave quads (which do not correspond to correct sequences of corners) or quads that would cover too few or too many pixels (which would make it impossible to contain a legible code). We call the resulting quads "candidate quads", because they may - but are not guaranteed to - contain a correct two-letter code. We transform the four corners of each candidate quad to a standardized square using a homography transformation to simplify subsequent processing. The standardized square is a 64×64 pixel image with 20 pixel margin on each side, i.e., 104×104 total; see Figure 6.6. The 20 pixel margin allows the *RejectorNet* to detect errors stemming from incorrect corner detections. Since we do not know the correct upright orientation of our two-letter code yet, we generate all four possible orientations; see Figure 6.6c, d. Figure 6.6c shows an example of an invalid quad and Figure 6.6d demonstrates a valid one.

6.2.5.1 Quad Classifiers

We trained two quad classifiers, *RejectorNet* and *RecogNet*. *RejectorNet* is a binary classifier predicting whether a candidate quad is valid, i.e., whether the four corners are at the correct locations and their order is correct relative to the upright code orientation; see Figure 6.6b. Also, the white square surrounded by a valid quad needs to contain a clearly legible code. Invalid quads are discarded, and the valid ones are passed to *RecogNet* which reads the codes, such as "2L" in Figure 6.6b. *RecogNet* is a multi-class classifier with two heads, one for each character of the two letter code. The details of candidate quad generation and quad classifiers can be found in Supplementary Material A.2. We

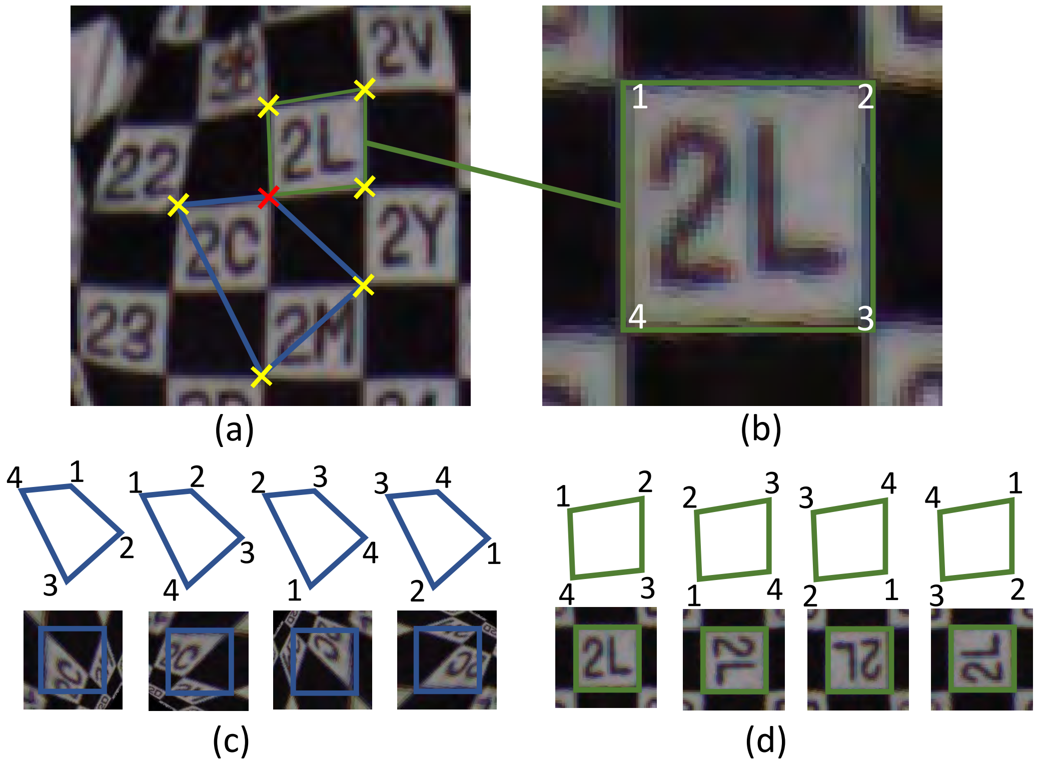


Figure 6.6: Quad generation: (a) For a given corner (red), we find three other corners (yellow) within a bounding box and compute their convex hull to generate a candidate quad. Here we visualize two example candidate quads (green and blue); (b) Our corner numbering convention with respect to upright code orientation; (c) and (d): The four possible orientations of the candidate quads, and their corresponding homography transformations. (c) is an invalid candidate, but the first orientation in (d) is valid.

use standard cross entropy losses to train those classifiers. The training of our CNNs is discussed in Section 6.5.

6.2.5.2 Corner Labeling and 3D Reconstruction

At this point, the two-letter codes of the valid quads have been recognized, including their upright orientation. The next step is to uniquely label each corner. We define a labeling function $l(code, i_q)$ which maps a two-letter *code* and corner index $i_q \in \{1, 2, 3, 4\}$ (see Figure 6.6b) to an integer which represents a unique corner ID. The unique corner IDs are defined for each suit. Many corners have *two* two-letter codes adjacent to them. If both of the two-letter codes are visible, we can leverage this fact as a redundancy check, detecting potential errors of *RecogNet*. Given unique corner IDs, we can convert corresponding 2D corners in more than two views into labeled 3D points. Let $\mathcal{C}_i$ be the set of cameras that see corner i, $k \in \mathcal{C}_i$ is a camera that sees corner i, $\mathbf{c}_i^k \in \mathbb{R}^2$ be the corner i's location in image coordinate system of camera k, and $f^k : \mathbb{R}^3 \to \mathbb{R}^2$ is the projection function of camera k. We computed 3D reconstructed corner $\mathbf{p}_i$ by minimizing the reprojection error:

$$\mathbf{p}_i = \operatorname*{arg\,min}_{\mathbf{p}_i \in \mathbb{R}^3} \sum_{k \in \mathcal{C}_i} ||f^k(\mathbf{p}_i) - \mathbf{c}_i^k||_2^2 \,. \tag{6.1}$$

This is a non-linear least square optimization problem; we compute an initial guess of $\mathbf{p}_i$ using the Linear-LS method [204] and optimize it using non-linear least squares solver [201].

6.2.5.3 Error Filtering

The label consistency check discussed above works only if two adjacent two-letter codes are present in the suit and visible in the images. If this is not the case, a corner can be assigned the wrong label if *RejectorNet* or *RecogNet* make a mistake. This kind of labeling errors will typically result in non-sensical correspondences with large reprojection errors, which we detect and correct by a RANSAC-type method discussed below. Specifically, for a corner with label i, let $\mathcal{C}_i$ be the set of the cameras that claim to see this corner. We assume that outliers in $\mathcal{C}_i$, i.e., the cameras that mislabeled corner i, should only be a minority. We iterate over all pairs of cameras (j, k) in $\mathcal{C}_i$, and 3D reconstruct the corner i from each pair. Among all of the pairs, we pick the 3D reconstruction that has the lowest reprojection error averaged over all cameras in $\mathcal{C}_i$ and assume this is the correct 3D location $\mathbf{p}_i$. Next,

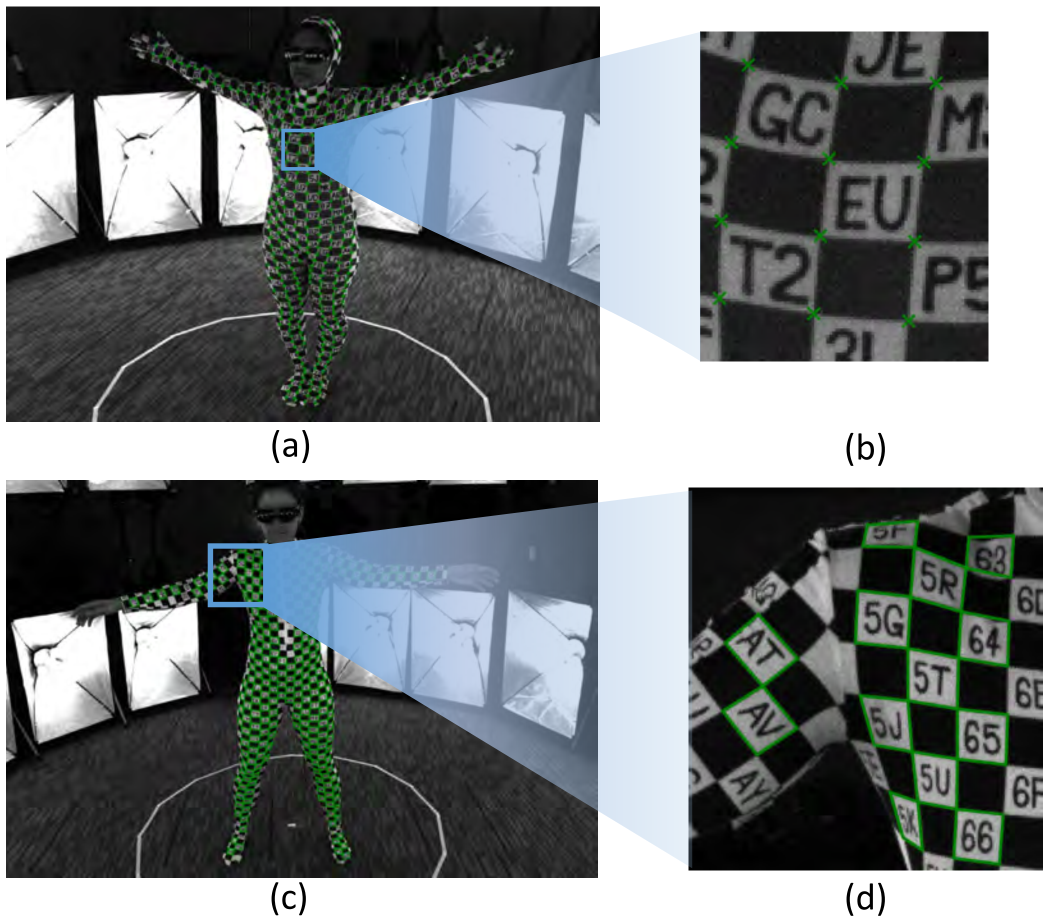


Figure 6.7: Two types of annotations we applied: (a, b): corner annoatation; (c, d): quad annotation.

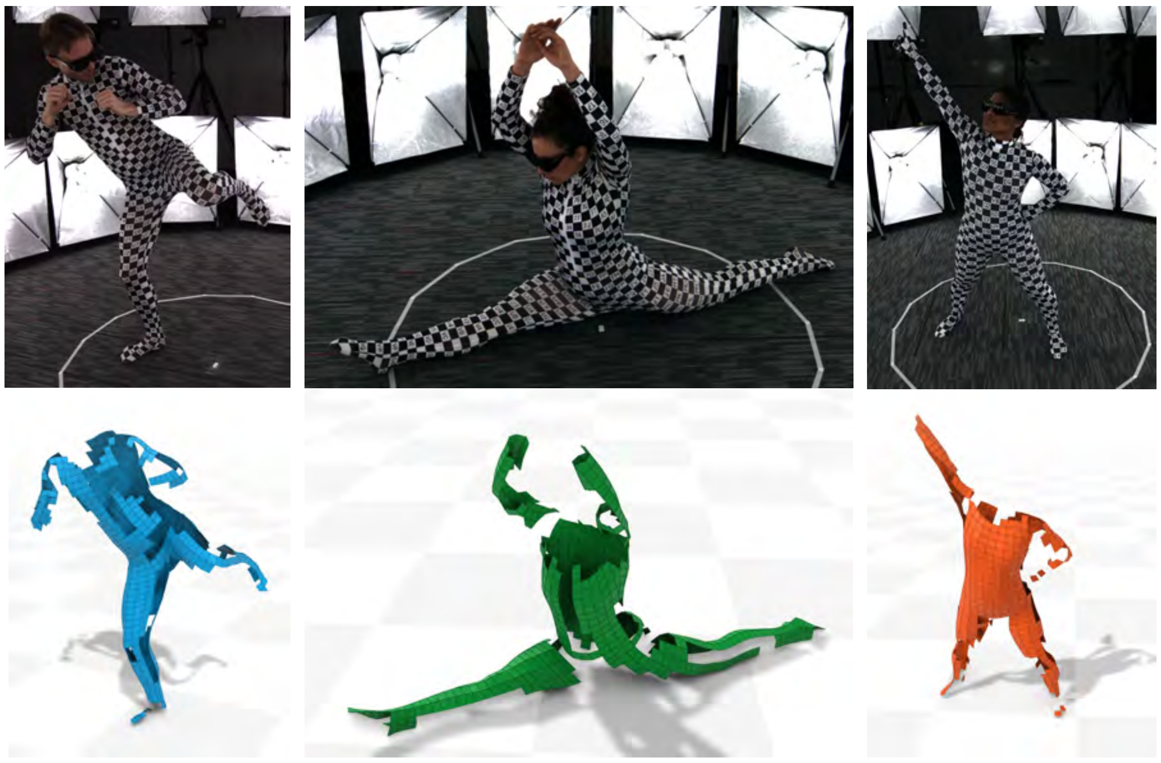


Figure 6.8: Example input images (top) from our multi-camera system and the corresponding raw 3D reconstructions (bottom). The reconstructed 3D points are meshed according to the patterns in our suits.

we analyze the reprojection errors of $\mathbf{p}_i$ into all of the cameras $\mathcal{C}_i$. The reprojection error should be low in cameras with correct labeling, but high if there was a labeling error. We use the $1.5 \times IQR$ (interquartile range) rule [205] to detect the outliers in terms of reprojections errors. We re-compute the triangulation of $\mathbf{p}_i$ after removing the outliers from the cameras $\mathcal{C}_i$. This RANSAC-type outlier filter does not work when there are only two cameras that see one corner. Therefore, we additionally discard reconstructed corners with an average reprojection error larger than 1.5 pixels. These tests are designed to be conservative, because mistakenly discarded points are not a major problem, just missing observations which can be inpainted as discussed in Section 6.4.

6.3 Data Acquisition and Neural Network Training

A key feature of our approach is that all of our networks are trained only on small image patches, e.g., see Figure 6.6c, d. This allows our trained model to generalize to different suits, capture environments, camera configurations and body poses that are not in the training set, because our local fiducial markers exhibit significantly less variability than images of full human poses. This is quite different from deep-learning-based methods that perform global pose-prediction, looking for the body as a whole. The training of our networks does not require large training sets. We have prepared our training data ourselves, without the use of any external annotation services or existing data sets.

Our dataset contains 28 manually annotated images, 24 of them are randomly selected from captures of our three actors wearing the hand-drawn suits. We also captured a different (fourth) actor wearing the printed suit and annotated four images of it in order to evaluate our approach on printed characters as opposed to hand-drawn. For each image, we apply two types of annotations: corner annotation (Figure 6.7a, b) and quad annotation (Figure 6.7c, d). In the corner annotation, we manually annotate all of our checkerboard-like corners on the suit with sub-pixel accuracy. In the quad annotation, we manually connect the corners annotated in the previous step into quads. Specifically, we create quads that correspond to valid white squares with two-letter codes in the suit and the annotators also write down the code of each annotated quad. We ensure the quad vertices are in a clockwise order and start from the top-left corner, defined by the upright orientation of the code (see Figure 6.6b). It takes about two person-hours to annotate one image. These annotations are then automatically converted into training data for our networks using techniques described in Appendix C.

We apply data augmentation to our training data through geometric and color space operations. We also generate synthetic training data by rendering textured human body models. The details can be found in Appendix C.

6.4 Filling Missing Observationswith Refined Body Model

Our method for 3D reconstruction of labeled points will inevitably result in missing observations because the human body often self-occludes itself and is observed only by a limited number of cameras; see Figure 6.8. In this section, we propose a method to interpolate (inpaint) the missing corners. Even though we could use any existing multi-person human body model [134, 135] for this purpose, we can achieve higher quality, because our pipeline gives us highly accurate measurements of the actor's body and its deformations. Therefore, instead of relying on previous statistical body shape models, we capture example motions of a given actor using our method and use this data to create a more precise *refined body model*, i.e., a model with parameters refined for a specific person.

Our body model has two types of parameters: shape parameters that are invariant in time, and pose parameters that change from frame to frame as the body moves. The shape parameters are only optimized during the model refinement process. After the body model refinement process is done, we fix the shape parameters and only allow the pose parameters to change. However, even after the refinement, the low-dimensional body model will not fit the 3D reconstructed corners exactly. We call the remaining residuals "non-articulated displacements", because they correspond to motion that is not well explained by the articulated body model. The non-articulated displacements arise due to breathing, muscle activations, flesh deformation, etc. Therefore, in addition to our refined body model we also interpolate the non-articulated displacements mapped to the rest pose via inverse skinning. The combination of the refined body model with the non-articualted displacement interpolation enables us to achieve high quality inpainting.

6.4.1 Actor-Specific Model Optimization

Our body model is based on linear blend skinning (LBS) [206]. Let $\mathbf{v}_i \in \mathbb{R}^4$ for $i = 1, 2, \ldots, N$ be the deformed vertices in homogeneous coordinates, where N is the number of all the vertices on our body model. We denote the skinning model as: $\mathbf{v}_i = D_i(\tilde{\mathbf{v}}_i, \mathbf{J}, \mathbf{W}, \grave{})$, where $\tilde{\mathbf{v}}_i \in \tilde{\mathbf{V}} = (\tilde{\mathbf{v}}_\mathbf{1}, \ldots, \tilde{\mathbf{v}}_N)$, $\tilde{\mathbf{V}}$ are rest pose vertex positions, $\mathbf{J} = (\mathbf{j}_\mathbf{1}, \ldots, \mathbf{j}_M)$ are joint positions and M is the number of joints in our model; $\mathbf{W} \in \mathbb{R}^{N \times M}$ is the matrix of skinning weights and $\theta \in \mathbb{R}^{M \times 4}$ are joint rotations represented by quaternions. In summary, $\tilde{\mathbf{V}}$, $\mathbf{J}$ and $\mathbf{W}$ are shape parameters (constant in time) and θ are pose parameters (varying in time). The deformed vertices can be computed as:

$$\mathbf{v}_i = D_i(\tilde{\mathbf{v}}_i, \mathbf{J}, \mathbf{W}, \theta) = \sum_{j=1}^{M} w_{i,j} T_j(\theta, \mathbf{J}) \tilde{\mathbf{v}}_i \,. \tag{6.2}$$

Note that here $\mathbf{v}_i, \tilde{\mathbf{v}}_i \in \mathbb{R}^{4 \times 1}$ are the deformed and rest pose vertex in homogeneous coordinates and $T_j(\theta, \mathbf{J}) \in \mathbb{R}^{4 \times 4}$ represents the transformation matrix of joint j. In the following we will use homogeneous coordinates interchangably with their 3D Cartesian counterparts.

6.4.1.1 Initialization

We initialize our body model by registering our corners to the STAR model [135]. We start by selecting a frame f_{init} in a rest-like pose where most corners are visible, and fit the STAR model to our labeled 3D points in f_{init} using a non-rigid ICP scheme which finds correspondences between our suit corners and the STAR model's mesh. The non-rigid ICP process is initialized by 10 to 20 hand picked correspondences between the STAR model and the 3D reconstructed corners. During the ICP procedure, We optimize both pose and shape parameters of the STAR model and iteratively update correspondences by projecting each of our 3D reconstructed points to the closest triangle of the STAR model (the actual closest point is represented using barycentric coordinates).

In this stage, we have registered most of our corners to the STAR model, but we still need to add corners that were unobserved in frame f_{init}. We can fit the STAR model to subsequent frames of our training motion using non-rigid ICP initialized by the registered corners instead of hand picked correspondences. These subsequent frames will reveal corners unobserved in the initial frame, which we register against the STAR mesh by closest-point projection as before. We use the corners registered to the STAR model's rest pose as the initial rest pose shape $\tilde{\mathbf{V}}^0$, and use barycentric interpolation to generate the initial skinning weights $\mathbf{W}^0$. Note that the number of vertices and mesh connectivity of our body model is different from the STAR model's mesh. We use each corner on the suit as a vertex of our model, and the rest pose vertex $\tilde{\mathbf{v}}_i$ corresponds to corner i in our suit. The meshing of our body model is discussed below. We use the STAR model's joints as the initial joint location $\mathbf{J}^0$. We removed the joints that controls head, neck, toes and palm from the STAR model, resulting in $M = 16$ joints. We call this model our *initial body model*.

6.4.1.2 Model Refinement

After the initialization, we further optimize the shape parameters to obtain our refined body model that more accurately fits a specific actor. Specifically, we optimize the skinning weights $\mathbf{W}$, the joint locations $\mathbf{J}$ and the rest pose vertex positions $\tilde{\mathbf{V}}$. Unlike *SMPL or STAR, we do not use pose-corrective blend shapes and instead correct the shape by interpolating non-articulated displacements, discussed in Section 6.4.2.

If $P^k, k = 1, 2, \ldots, K$ is the set of 3D points that were reconstructed from frame k and K is the number of frames in the training set, we refine the body model by minimizing:

$$\mathcal{L}_A(\tilde{\mathbf{V}}, \mathbf{J}, \mathbf{W}, \Theta) = \mathcal{L}_f(\tilde{\mathbf{V}}, \mathbf{J}, \mathbf{W}, \Theta) + \lambda_g \mathcal{L}_g(\mathbf{W}) + \lambda_J \mathcal{L}_J(\mathbf{J}) \tag{6.3}$$

where $\Theta = (\theta_1, \theta_2, \ldots, \theta_K)$ are the pose parameters of all the frames in the training set and $\mathcal{L}_f(\tilde{\mathbf{V}}, \mathbf{J}, \mathbf{W}, \Theta)$ is the fitting error term:

$$\mathcal{L}_f(\tilde{\mathbf{V}}, \mathbf{J}, \mathbf{W}, \Theta) = \frac{1}{\sum_{k=1}^{K} |P^k|} \sum_{k=1}^{K} \sum_{\mathbf{p}_i^k \in P^k} \| D_i(\tilde{\mathbf{v}}_i, \mathbf{J}, \mathbf{W}, \theta_k) - \mathbf{p}_i^k \|_2^2 . \tag{6.4}$$

$\mathcal{L}_J(\mathbf{J})$ is an l_2 loss penalizing joint locations moving too far away from their initial positions and $\mathcal{L}_g(W)$ is a regularization term encouraging sparsity of the skinning weights:

$$\mathcal{L}_g(\mathbf{W}) = \sum_{i=1}^{N} \sum_{j=1}^{M} g_{i,j} w_{i,j}^2 \tag{6.5}$$

where $g_{i,j}$ is the geodesic distance from corner i to the closest vertex that has non-zero initial weight for joint $\mathbf{j}_j$ in the STAR model. The regularization weights were empirically set to $\lambda_g = 1000$ and $\lambda_J = 1$ when our spatial units are millimeters.

We optimize $\mathcal{L}_A$ with an alternating optimization scheme. Starting with the initial LBS model, we first calculate pose parameters θ_k for each frame. Then we optimize $\mathbf{W}$, $\mathbf{J}$, and $\tilde{\mathbf{V}}$

one by one, while keeping the other parameters fixed. We iterate this procedure until the error decrease becomes negligible; in our results we needed between 50-100 iterations.

After the optimization is finished, we mesh the rest pose vertices $\tilde{\mathbf{V}}$. From the unique ID of each corner, we know how they were connected into quads in the suit. We manually add vertices to close the holes which come from areas of the suit such as the zipper and the seams (see Figure 6.9a). The result is a quad-dominant mesh (Figure 6.9b).

6.4.2 Point Cloud Completion

After the optimization, the fitting error (Equation 6.4) will drop from 13.5mm to 7.1mm on the test set; further results are reported in Section 6.5.4. The refined LBS body model is good for representing articulated skeletal motion of the actor's body, but it does not represent well effects such as breathing or flesh deformation. However, the non-articulated component of the motion that cannot be represented by LBS is relatively small. Therefore, we start by applying inverse skinning transformations (also known as "unposing") to our observed 3D reconstructed points $\mathbf{p_i}$; see Figure 6.10d. We denote the inverse skinning of point i at pose k as $D_i^{-1}(\mathbf{p}_i, \mathbf{J}, \mathbf{W}, \theta_k)$. As can be seen in Figure 6.10d, the $D_i^{-1}(\mathbf{p}_i, \mathbf{J}, \mathbf{W}, \theta_k)$ will not exactly match $\tilde{\mathbf{v}}_i$ due to the non-articulated residuals. Formally, the non-articulated displacements $\Delta\tilde{\mathbf{v}}_i^k$ are defined as:

$$\mathbf{v}_i^k = D(\tilde{\mathbf{v}}_i + \Delta\tilde{\mathbf{v}}_i^k, \mathbf{J}, \mathbf{W}, \theta_k)\,. \tag{6.6}$$

The key problem of our inpainting consists in interpolating the values of $\Delta\tilde{\mathbf{v}}_i^k$ from the observed points to the unobserved ones, in other words, predicting the unobserved non-articulated displacements; see Figure 6.10e. The modified rest pose is then mapped back by (forward) skinning to produce the final mesh; see Figure 6.10f.

Our method for predicting the unobserved non-articulated displacements in the rest pose is based on the assumption of spatio-temporal smoothness. We stack all of the rest pose displacements into a $KN \times 3$ matrix $\mathbf{X}$, where K is the number of frames and N the number of vertices (all vertices, both unobserved and observed ones).

We find $\mathbf{X}$ by solving the following constrained optimization problem:

$$\begin{aligned} &\min \mathcal{L}_{\text{spat}}(\mathbf{X}) + w_T \mathcal{L}_{\text{temp}}(\mathbf{X}) \\ &\text{s.t. } \mathbf{CX} = \mathbf{D} \end{aligned} \tag{6.7}$$

where $\mathcal{L}_{\text{spat}}$ is a spatial Laplacian term that penalizes non-smooth deformations of the mesh and $\mathcal{L}_{\text{temp}}$ is a temporal Laplacian term that penalizes non-smooth trajectories of the vertices. Both of the terms are positive semi-definite quadratic forms. The parameter w_T is a weight balancing these two terms which we empirically set to 100. The sparse selector matrix $\mathbf{C}$ represents the observed points (constraints) and $\mathbf{D}$ their unposed 3D positions for each frame (each frame may have a different set of observed points). Specifically, we define $\mathcal{L}_{\text{spat}}$ as:

$$\mathcal{L}_{\text{spat}} = \sum_{i=1}^{3} \mathbf{X}_i^T \mathbf{L} \mathbf{X}_i \tag{6.8}$$

where $\mathbf{L}$ is cotangent-weighted Laplacian of the rest pose and $\mathbf{X}_i$ is the i-th column of $\mathbf{X}$. We found this quadratic deformation energy to be sufficient because our non-articulated displacements in the rest pose are small, though in future work it would be possible to explore non-linear thin shell deformation energies. For $\mathcal{L}_{\text{temp}}$, we use 1D temporal Laplacian which corresponds to acceleration: $\|\Delta\tilde{\mathbf{v}}_i^{k-1} - 2\Delta\tilde{\mathbf{v}}_i^{k} + \Delta\tilde{\mathbf{v}}_i^{k+1}\|_2^2$. The $\mathcal{L}_{\text{spat}}$ operator is applied to all frames independently, and $\mathcal{L}_{\text{temp}}$ is applied to all vertices independently. However, their weighted combination in Equation 6.7 introduces spatio-temporal coupling, allowing one observed point to affect unobserved points through both space and time.

Note that Equation 6.7 is equivalent to the Variational form of Back Euler Equation 2.8 with linear bilateral constraints. The temporal Laplacian L_{temp} corresponds to the inertia term with external acceleration set to 0, and L_{spat} is a quadratic bending energy [207].

The optimization problem Eq. 6.7 is a convex quadratic problem subject to equality constraints, which we transform to a linear system (KKT matrix) and solve. The only complication is that when processing too many frames, the KKT system can become too large. For example, with $K = 5000$ frames, the KKT matrix becomes approximately $10^7 \times 10^7$. Even though the KKT matrix is sparse, the linear solve becomes costly. To avoid this problem, we observe that smoothing over too many frames is not necessary and introduce a windowing scheme, decomposing longer sequence into 150-frame windows and solve them independently. To avoid any non-smoothness when transitioning from one window to another, the 150-frame windows overlap by 50 frames. After solving the problem in Equation 6.7 for each window separately, we smoothly blend the overlapping 50 frames to ensure smoothness when transitioning from one window to the next one.

In this section we considered only off-line hole filling, where we can infer information from future frames. This approach would not be applicable to real-time settings where future frames are not available.

6.5 Results

We first discuss the details of our CNN training process and the results. To prepare the training data, we manually annotated 24 randomly selected images (4000×2160) of our three actors wearing the handwritten suits. Out of the 24, we withheld 4 images as test set A. To evaluate our system's ability to generalize to a different suit and person, we also captured a short sequence of a fourth actor wearing the printed suit and annotated 4 images from this sequence as test set B. Note that our trained process has never seen the font in the printed suit. Table 6.1 shows the total numbers of images used for training our CNNs. As shown in the first row of Table 6.1, the original training set (without data augmentation) for *RecogNet* is much smaller compared to *CornerdetNet* and *RejectorNet*, because of the limited number of valid quads in each of our annotated images. To improve the classification performance of *RecogNet*, we used synthetically generated images to complement the real data, as discussed in Supplementary Material B.2. The synthetic data contain 214471 crops (104×104), which significantly improved the robustness of *RecogNet*; see Section 6.5.1.

We train our CNNs using Tensorflow [208] using a single NVIDIA Titan RTX; for each of our CNNs, an overnight run is typically enough to converge to good results using the Adam optimizer. After our CNNs have been trained, we run inference on a PC with an i7-9700K CPU and an NVIDIA GTX 1080 GPU. With a 4000×2160 input image, an inference pass of *CornerdetNet* takes 300ms, generating candidate quads 10ms, *RejectorNet* takes 1-2s to classify all of the candidate quads (104×104) and *RecogNet* takes 5ms to recognize the valid quads. The computational bottleneck is the *RejectorNet* due to the large number of candidate quads; this could be improved in the future by a more aggressive culling of candidate quads. For each frame, the time for 3D reconstruction is negligible, taking less than 1ms for all points. Even though we used only one computer and processed our image sequences off-line, we would like to point out that our method for extracting 3D labeled points from multi-view images is embarrassingly parallel, because each frame and even each input crop for our CNNs can be processed independently. Coupling through time is introduced only in the final hole-filling step (Section 6.4.2). The time for solving the sparse linear system (Eq. 6.7) for a 150 frames window is about 10s.

We captured motion sequences of three actors, one male and two females. One of the female actor wears the small suit and the other two actors wear the medium suit. For each actor, we captured about 12,000 frames (at 30FPS) of raw image data consisting of 1) camera calibration, 2) 6000 frames of calisthenics-type sequence intended for body model refinement (also serving as a warm-up for the actor), 3) the main performance. Each frame consists of 16 images from our multicamera setup. It took about 300 hours to process all of the 576,000 images (4.6 TB) using one computer.

6.5.1 Evaluation of CNNs

6.5.1.1 Cornerdet

There are two parts of the *CornerdetNet*'s output: 1) classification response that predicts whether there is a valid corner in the center 8×8 window of the input 20×20 image and 2) its subpixel coordinates (or arbitrary values of a corner is not present). We summarize the results for both classification and localization errors. The localization error is measured by the distance in pixels between the predicted corner location and the manually annotated corner location. The overall classification accuracy for *CornerdetNet* is 99.393% on the training set and 99.510% on the test set A. The fact that *CornerdetNet* works better on the test set A supports our hypothesis that more aggressive data augmentation results in worse performance on the training set but better performance on the test set. On the test set A, the average localization error is 0.21 pixels and 99% of the corner localizations achieve error 0.6361 pixels or less, which is remarkably low. Similar accuracy was also achieved on the test set B, which is from the printed suit, which means *CornerdetNet* can generalize to a suit that it has not been trained on. With our camera setup, 1 pixel error corresponds to approximately 1mm of 3D error for an actor 2 meters away from the camera. In practice, this means that our 3D reconstructed points are highly accurate, allowing us to capture minute motions such as muscle twitches or flesh jiggling.

6.5.1.2 *RejectorNet*

The confusion matrices of a trained *RejectorNet* network are reported in Table 6.2a - Table 6.2e. The overall classification accuracy for *RejectorNet* is 99.723% on the training set, 99.704% on the test set A, and 99.31% on the test set B. From the confusion matrix, we can observe that we have more false positives than false negatives. The reason is that we intentionally annotated the training data conservatively. As shown in Figure 6.11a, quads with even slight imperfections were labeled as negative examples. This results in *RecogNet* reporting more false positives, but the *RecogNet* actually inherits the conservative nature of the annotations; in practice, *RecogNet* only rarely accepts a low-quality quad image. This nature is well demonstrated by its performance on test set B. Because the *RejectorNet* have never seen the font used on the printed suit, due to its conservative nature it tends to reject a considerable number of valid quads (around 23% of valid quads are rejected). On the other hand, the *RejectorNet* returns no false positives, i.e., no invalid quads were falsely accepted.

6.5.1.3 *RecogNet*

We compare the *RecogNet* trained with/without the synthetic training set in Table 6.3 - Table 6.6. Without using the synthetic training set, the *RecogNet* had prediction accuracy of 99.522% on the test set A. This accuracy was low and it was the main source of errors in our pipeline. Enhanced with the synthetic training set, the prediction accuracy on the test set A increased to 99.919% and significantly improved our results. Moreover, without using the synthetic training set, the *RecogNet* cannot generalize well to the test set B: the prediction accuracy on it is only 85.16%. That is because *RecogNet* has never seen this new font on the printed suit. However, by enhancing training set with synthetic data which uses various fonts, the generalizability of *RecogNet* has improved significantly, achieving an accuracy of 100% on the test set B.

6.5.1.4 Overall Performance

In the previous sections we reported the results of each individual CNN. To evaluate our complete corner localization and labeling pipeline (Figure 6.5), we use our 2 test sets of 8 manually annotated images (4000×2160) where we know the ground truth positions and labels of all corners. The images in the test set A and test set B contain 1702 and 1296 manually labeled corners, respectively. Our 2D pipeline detected 92% (1566 of 1702) of the ground truth corners from the test set A and 69% (896 of 1296) of the ground truth corners from the test set B. The discarded corners corresponded to low quality quads or quads with printed fonts that are significantly different from the hand-drawn ones, which were rejected by *RejectorNet*. Note that we intentionally trained the *RejectorNet* to be conservative, i.e., to reject all borderline cases. Missing observations on the test set A do not represent a big problem because they can be fixed by inpainting (Section 6.4.2); also, we observed that low-quality quads are often associated with inaccurate corner localization, increasing the noise in the 3D reconstruction. The missing observations on the test set B could be improved by enhancing the training of *RejectorNet* with a few more annotated images from the new suit, or training *RejectorNet* with synthetic data. The mean corner localization error in our is 0.4607 pixels test set A and 0.4678 on the test set B, and the maximum localization error is 1.854 on the test set A and 1.031 on the test set B. Due to our conservative rejection approach, the final CNN, *RecogNet*, made *zero* mistakes on the test sets, i.e., all of the labeled corners in the two test sets were assigned the correct label.

6.5.2 Comparison to Previous Methods

6.5.2.1 CornerdetNet

We compared our *CornerdetNet* with three alternative corner detectors: Shi-Tomasi [209], Harris [210], and deep-learning-based detector “SuperPoint” [144]. For Shi-Tomasi [209] and Harris [210], we use OpenCV’s [211] implementation and the recommended parameters. We used the method cv::cornerSubPix() [212] to refine the detected corner locations to subpixel accuracy.

A qualitative comparison of corner detection results is shown in Figure 6.12. We also quantitatively compare those corner detectors using the test set B. To evaluate the classification accuracy, we match the detected corners to the annotated corners if the distance between them is less than 1.5 pixels. The classification accuracy comparison is shown (Table 6.7). As we can see from both Figure 6.12 and Table 6.7, Shi-Tomasi, Harris and SuperPoints all have the same problem: they detect a lot of non-checkerboard corners (false positives), many of which are from the text on the suit. This is not surprising because those corner detectors are designed to detect general features that exists in nature, not checkerboard corners specifically. These wrong detections can slow down the processing of our pipeline because they lead to more invalid candidate quads. Also, they can confuse *RejectorNet*. On the other hand, alternative corner detectors are missing many valid corners in comparison to *CornerdetNet*, which will results in missing observations. From Table 6.8, we can see that those corner detectors are also inferior to *CornerdetNet* in terms of localization accuracy.

6.5.2.2 *RecogNet*

We compared our *RecogNet* with Tesseract-OCR [213]. The results are shown in Table 6.9. Tesseract-OCR produces significantly worse results than *RecogNet*, especially on the hand-drawn test set A. In practice, we find that Tesseract-OCR is very sensitive to stretching, distortion and noise. This is probably due to the fact that Tesseract-OCR was designed to recognize printed text with standard format rather than text on a suit worn by human which can have significant distortion due to stretching. Our dedicated *RecogNet* performs much better in this case.

6.5.3 3D Reconstruction Accuracy

6.5.3.1 Metrics

Evaluating 3D reconstruction accuracy is hard, because we do not have any ground truth measurements of a moving human body. To evaluate the accuracy of our 3D reconstructed corners, we compute their reprojection errors and we compare them to the reprojection errors obtained in our camera calibration process (Section 6.2.5). Using f^k to denote the projection function of camera k, if a reconstructed 3D point $\mathbf{p}_i$ is seen by camera k and $\mathbf{c}_i^k$ is the pixel location of the corresponding 2D corner i in camera k, the reprojection error for corner i in camera k is defined as:

$$\mathbf{e}_{i,k} = ||f^k(\mathbf{p}_i) - \mathbf{c}_i^k|| \,. \tag{6.9}$$

The reprojection error for camera calibration is defined analogously, except that we use 3D calibration boards with perfect, rigid checkerboard corners and a standard OpenCV corner detector. In contrast, our corners are painted on an elastic suit worn by an actor.

6.5.3.2 Quantitative Evaluation

We report the histograms of reprojection errors of 3D reconstruction and camera calibration in Figure 6.13. The 3D reconstruction reprojection error is computed per camera for all the reconstructed points in a consecutive sequence of 10000 frames. The calibration reprojection error was computed on 448 frames that we use to calibrate the cameras, where we wave a 9×12 calibration board in front of our cameras. In Figure 6.13, we can see the two error distributions look very similar, which means the reprojection errors of our 3D reconstruction results have similar statistics as the reprojection errors in camera calibration. We cannot expect to obtain lower reprojection errors than camera calibration.

Table 6.10 shows the percentiles of all the reprojections errors in 10000 frames that we use to evaluate the 3D reconstruction. 99% of the reprojection errors is less than 1.009 pixels, which is remarkably accurate given the high resolution of our images (4000×2160).

6.5.3.3 Qualitative Evaluation

Figure 6.14 shows challenging cases where there are significant self occlusions. We mesh the reconstructed point cloud using the rest pose mesh structure introduced in Section 6.4.1 by preserving the observed faces in the rest pose mesh (see Figure 6.9b). Then we project the reconstructed mesh back to the image using the camera parameters, which gives us the green wireframe in Figure 6.14. We can see that the mesh wireframe aligns very closely with the checkerboard pattern on the suit. Another important observation is that even despite large occlusions, our method can still obtain correctly labeled corners as long as the entire two-letter code is visible see, e.g., the foot and calf in Figure 6.14a, b. In Figure 6.14c, we can see that the conservative *RejectorNet* correctly rejects the wrinkled quads in the belly region, since reading the codes would be difficult or impossible.

6.5.4 Evaluation of Body Model Refinement

To refine our actor model, we record a 6000 frames training sequence. After the body model refinement we select another 3000 frames corresponding to motions different from the ones in the training set. The fitting error is defined as the distance between the vertices of the deformed body model and the actual 3D reconstructed corners. We compare the fitting errors between the initial model, which is just a remeshing of the STAR model (see Section 6.4.1), and the refined body model, which was optimized on the training set (see

Section 6.4.2). Figure 6.15 shows the distribution of fitting errors per vertex of the initial body model and the refined body model on the training set and the test set. We can see in both data sets, the refined body model is much more accurate. Specifically, the body model refinement reduces the average fitting error from 13.6mm to 5.2mm on the training set and from 13.5mm to 7.1mm on the test set. Figure 6.16 visualizes the fitting errors on the body model before and after body model refinement in one example frame using a heat map.

6.5.5 Evaluation with Optical Flow

To quantify the accuracy of the 3D reconstruction of the entire body, we compare renderings of a textured mesh with the original images using optical flow [166, 167]. First, we need to create a suit-like texture for our body mesh (Figure 6.9b). We create a standard UV parametrization for our mesh and generate the texture from 10 hand picked frames using a differentiable renderer [214]; though this is just one possible way to generate the texture [167]. We render the textured body mesh with back face culling enabled and overlay it over clean plates (i.e., images of the capture volume without any actor). The virtual camera parameters are set to our calibration of the real cameras. The optical flow is computed from the synthetic images to the undistorted real images using FlowNet2 [215] with the default settings. Because our mesh does not have the hands and the head, we first render a foreground mask of our body mesh (Figure 6.17c). We only evaluate the optical flow on the region covered by the foreground mask to exclude the hands, the head and the background. The foreground mask cannot exclude the hands and the head when they are occluding the body (as in Figure 6.17a, b) but, fortunately, the optical flow is robust to missing parts (see Figure 6.17d). We use optical flow to compare the original images with two types of renders: 1) our low-dimensional refined body model (the gray mesh in Figure 6.10c, which does not fit the reconstructed corners exactly), 2) our final result after adding non-articulated displacements (Figure 6.10f). Figure 6.18a plots the average optical flow norm for each frame for consecutive 2000 frames, including various challenging poses and fast motions. We can see that the result with non-articulated displacements is much more accurate than only our low-dimensional refined body model. This is mainly due to flesh deformation which is not well explained by the refined body model, especially in more extreme poses which correspond to the spikes in the blue curve in Figure 6.18a.

The red curve corresponds to our final result which exhibits consistently low optical flow errors.

We also plot the distribution of the optical flow norm for each pixel in the foreground mask in Figure 6.18b. With our final animated mesh, 95% of pixels have optical flow norm less than 1.20 and 99% of pixels have optical flow norm less than 2.46.

6.6 Discussions

We prove some qualitative results in Figure 6.19. An obvious limitation of our method is the necessity of wearing a special motion capture suit. A suit can in principle slide over the skin, but we did not observe any significant sliding in our experiments because our suits are tightly fitting. If this became a problem in the future, we could increase adhesion with internal silicone patches as in sportswear, or even apply spirit gum or medical adhesives. The suit needs to be made in various sizes and fit may be a challenge for obese people. The holy grail of full-body capture is to get rid of suits and instead rely only on skin features such as the pores, similarly to facial performance capture. We tried imaging the bare skin, but with our current camera resolution (4000×2160) we were unable to get sufficient detail from the skin. We could obtain more detail with narrower fields of view and more cameras to cover the capture volume, but then there are issues with the depth of field and hardware budgets. Additional complications of imaging bare skin are body hair and privacy concerns; our suit certainly has its disadvantages, but mitigates these issues. A significant advantage of our suit compared to traditional motion capture suits is that we do not need to attach any markers (reflective spheres; see Figure 6.2a). Traditional motion capture markers can impede motion or even fall off, e.g., when the actor is rolling on the ground. An intriguing direction for future work would be to enhance our suit with additional sensors, in particular EMG, IMU or pressure sensors on the feet.

In this paper we focused on the body and ignored the motion of the face and the hands. Our actors are wearing sunglasses because our continuous passive lights are too bright; the perceived brightness could be reduced by lights which strobe in sync with camera shutters, but this would require significant investments in hardware. In future work, our method could be directly combined with modern methods that capture the motion of the face and the hands [136, 137, 127, 128]. We note that our current system captures the motion of the feet, but not the individual toes.

Our current data processing is off-line only. We believe it should be possible to create a real-time version of our system. This would require machine vision cameras tightly integrated with dedicated GPUs or tensor processors for real-time neural network inference. Each such hardware unit could emit small amounts of data: only information about the corner locations and their labels, avoiding the high bandwidth requirements typically associated with high-resolution video streams.

Another avenue for future work involves research of different types of fiducial markers that can be printed on the suit. In fact, we made initial experiments with printing on textiles and sewing our own suits, which gives us much more flexibility than handwritten two-letter codes discussed in this paper. Our pipeline for reconstructing labeled 3D points does not make any assumptions about the human body, which means that we could apply our method also for capturing the motion of clothing or even loose textiles such as a curtain.

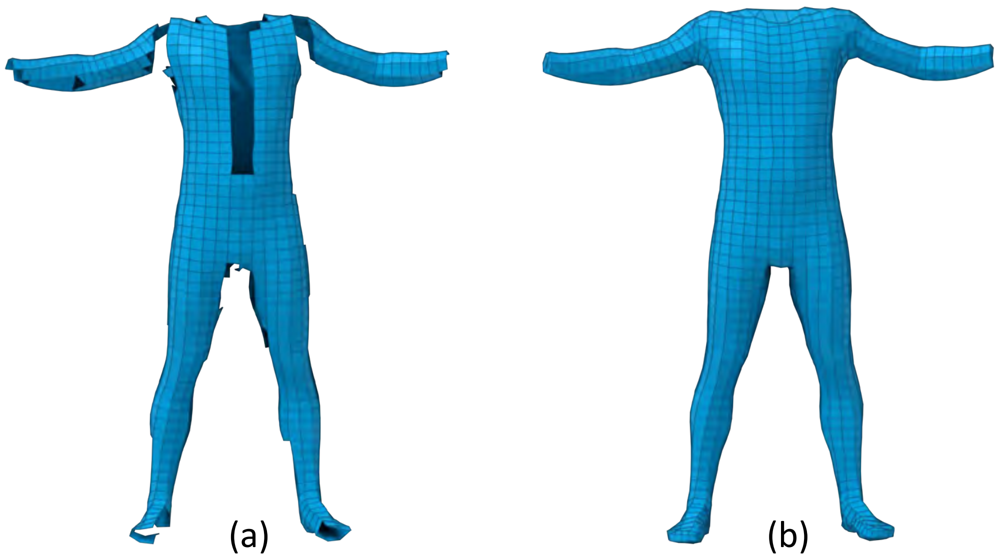


Figure 6.9: Visualization of the hole filling. (a) The quad structure corresponding to our suit has holes due to the zipper and the seams; (b) The completed rest pose mesh.

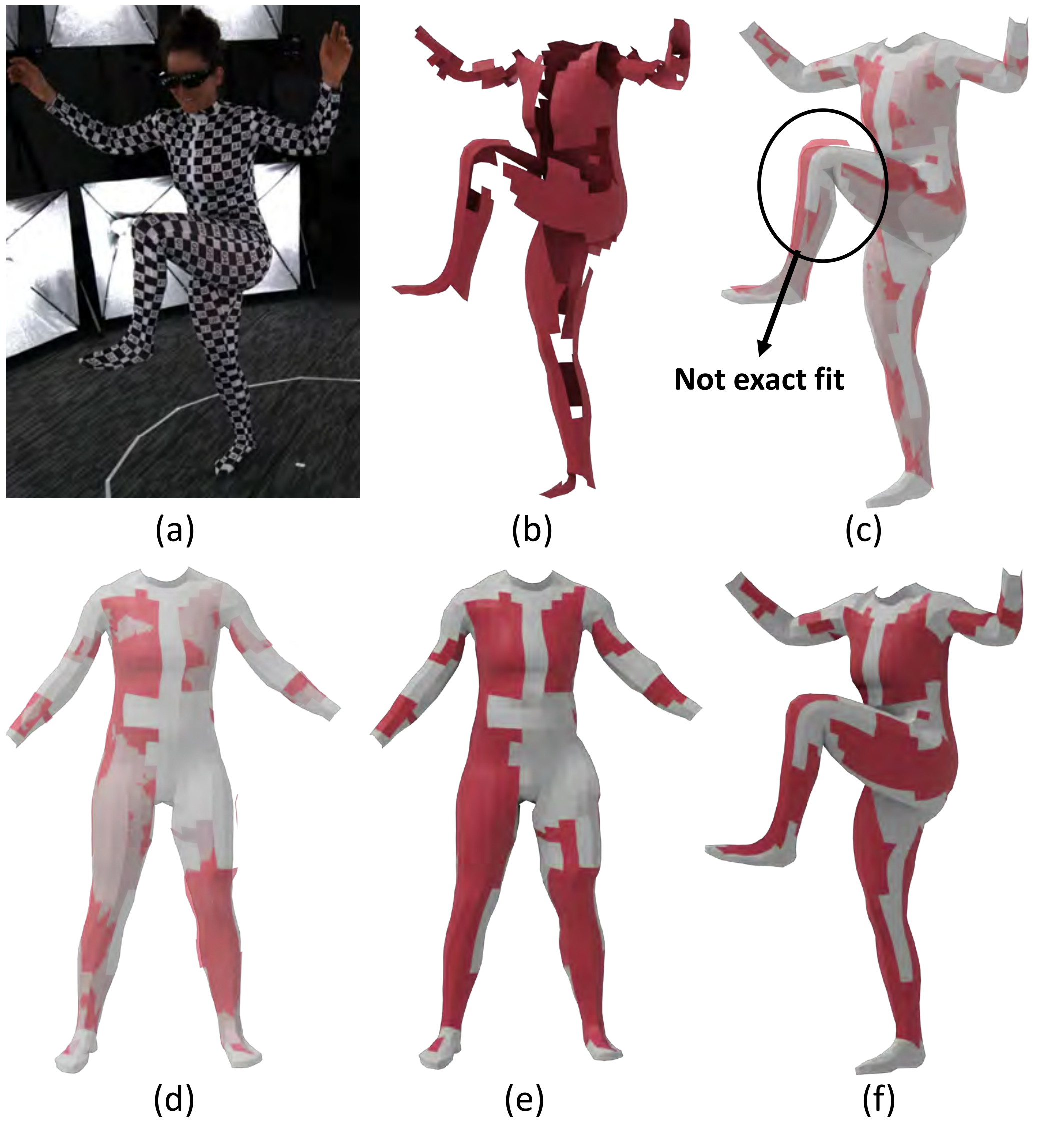


Figure 6.10: Our hole-filling pipeline: (a) an input image; (b) 3D points P reconstructed from input images and connected with quads; (c) our refined body model (gray) does not fit P exactly (transparent rendering shows discrepancies); (d) inverse skinning, mapping P to the rest pose; (e) rest pose mesh interpolation, matching the inverse-skinned P exactly; (f) the final result obtained by forward skinning of the interpolated rest pose mesh.

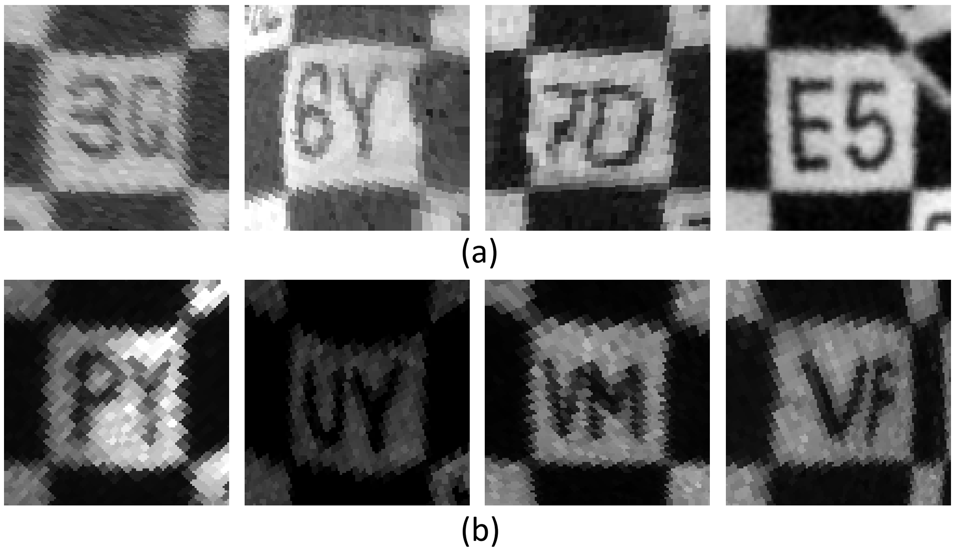


Figure 6.11: Examples of errors made by *RejectorNet*. (a) False positives; even though the codes are legible, these samples were labeled negative due to slight image imperfections. (b) False negatives, labeled positive but close to the decision boundary.

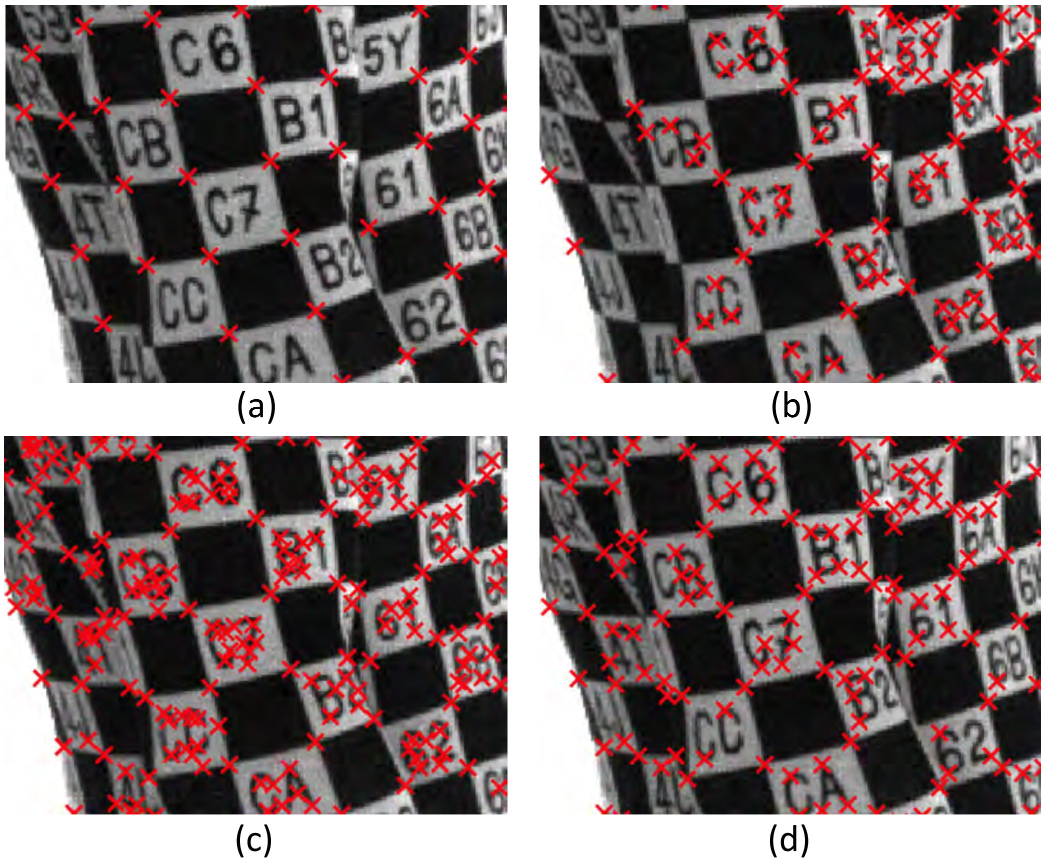


Figure 6.12: Comparison between different corner detectors. (a) Our *CornerdetNet*; (b) Shi-Tomasi; (c) Harris; (d) SuperPoint.

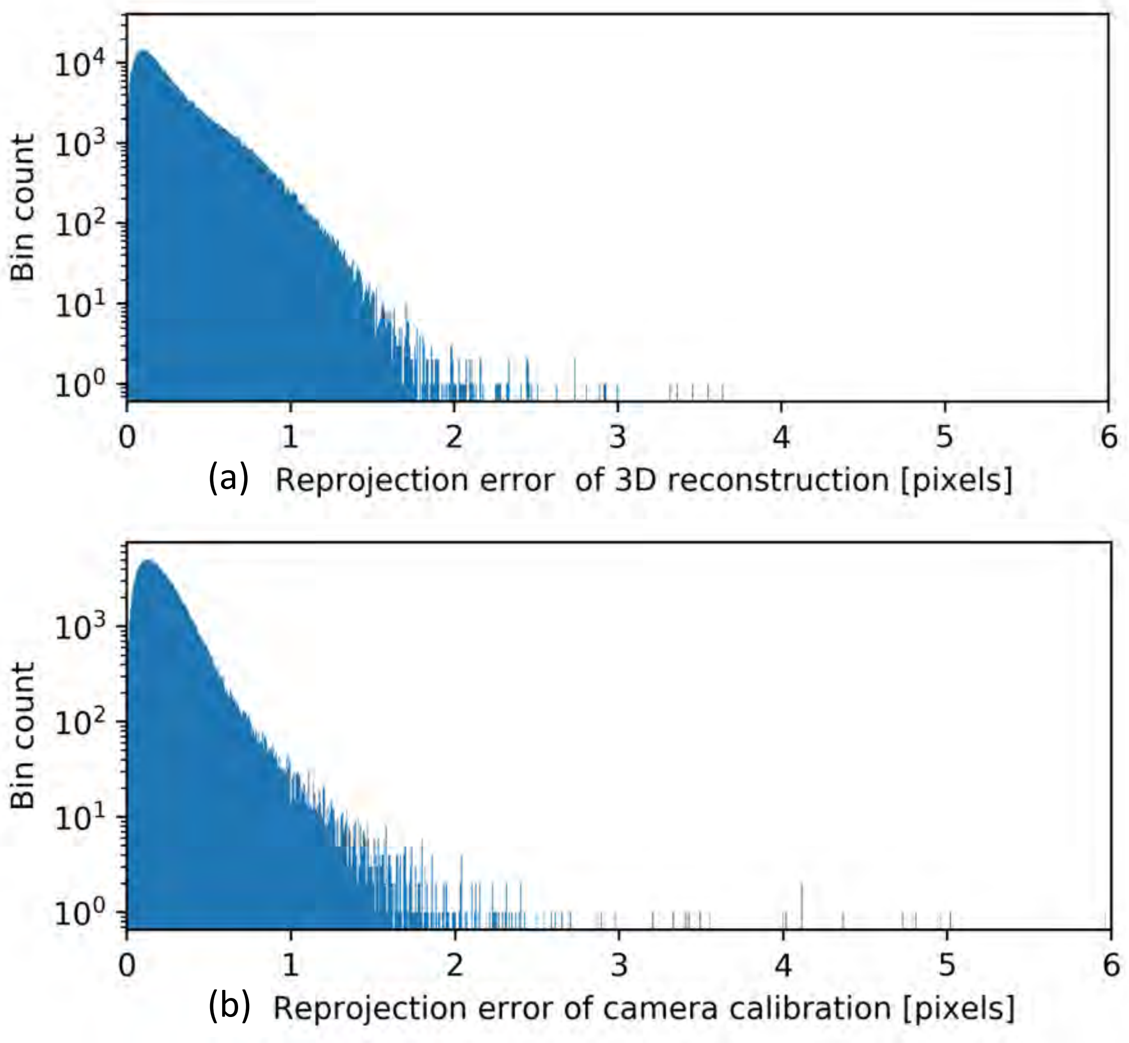


Figure 6.13: Comparison of histograms of reprojection errors in 3D reconstruction and camera calibration. (a) Distribution of reprojection errors computed per camera for all the 3D reconstructed corners in 10000 consecutive frames. (b) The distribution of reprojection errors of camera calibration. We can see those two histograms look very similar.

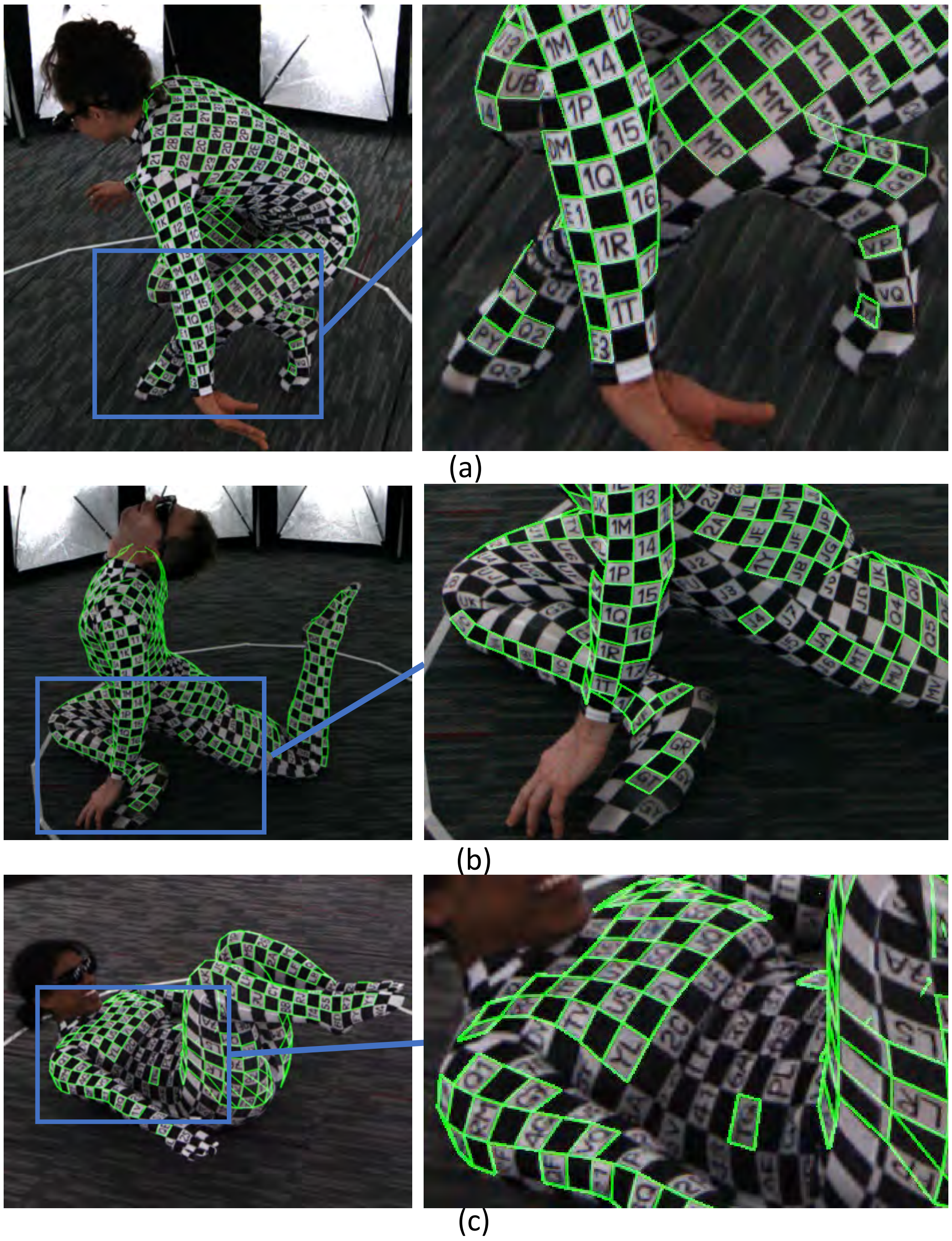


Figure 6.14: Our results in challenging poses. Our reconstructed mesh (visualized as green wireframe) closely matches the checkerboard pattern in the original input images (background). Note the successful isolated code recognitions on the feet in (a) and (b).

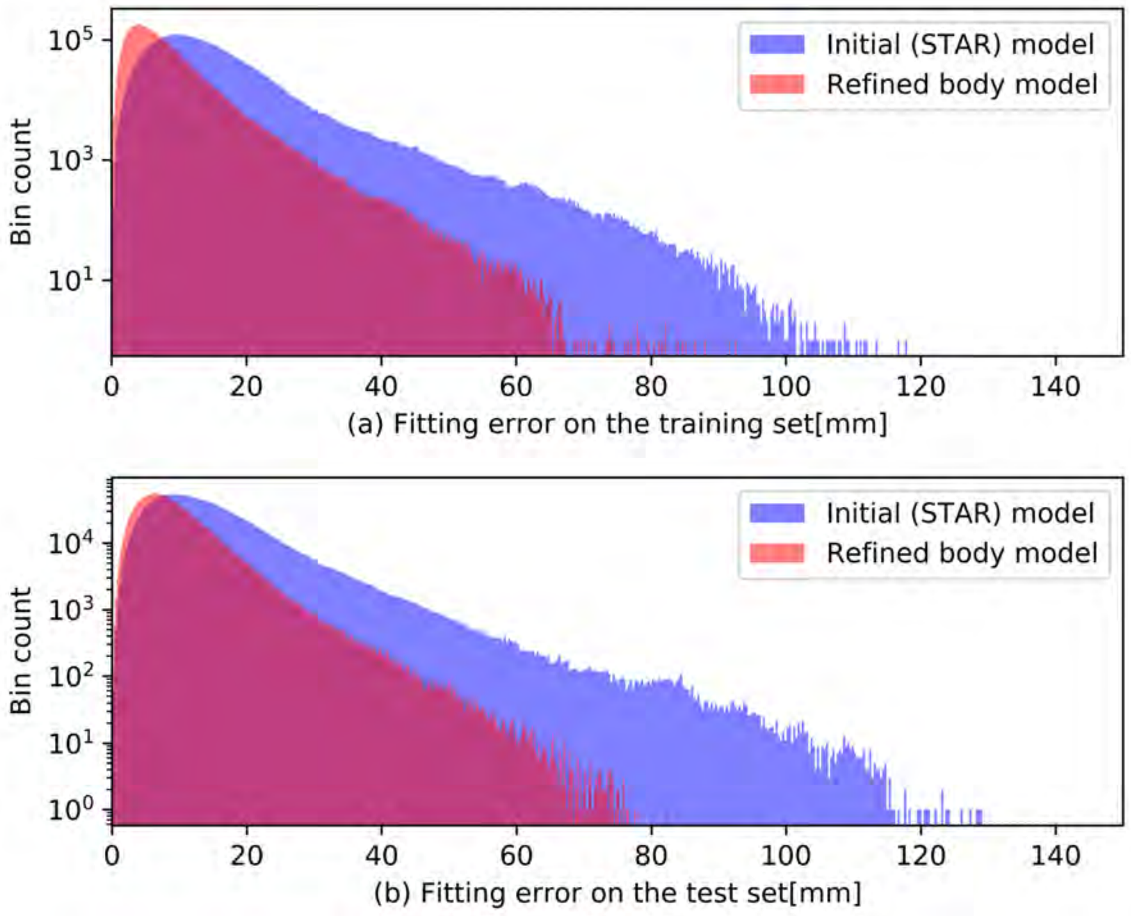


Figure 6.15: Histograms of fitting errors of the initial body model and the refined body model on the training set and the test set.

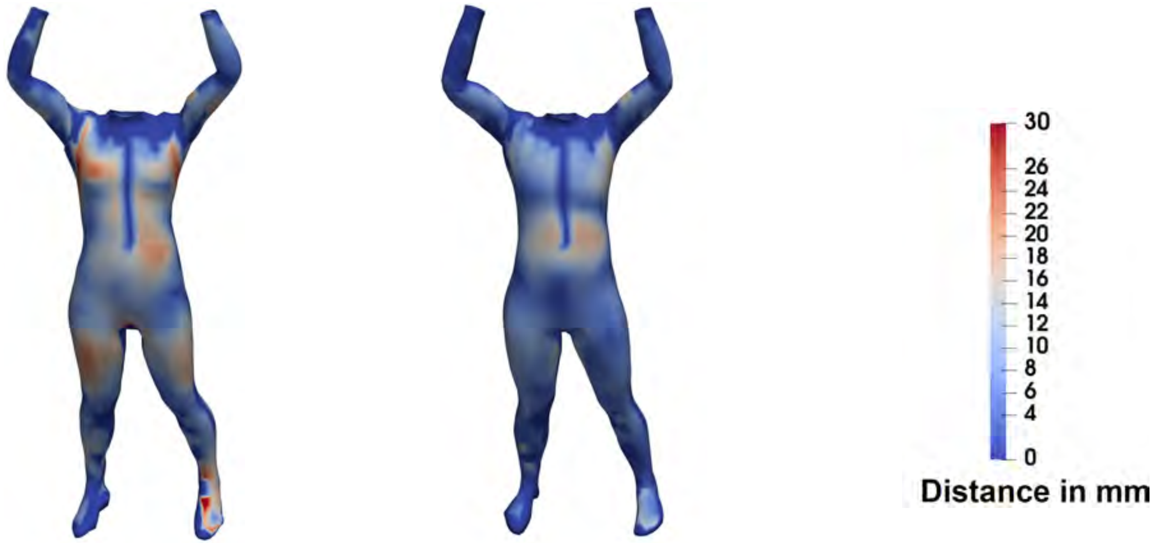


Figure 6.16: The fitting errors on the body model; left: an initial body model which is just a re-meshing of the STAR model; right: our final refined body model.

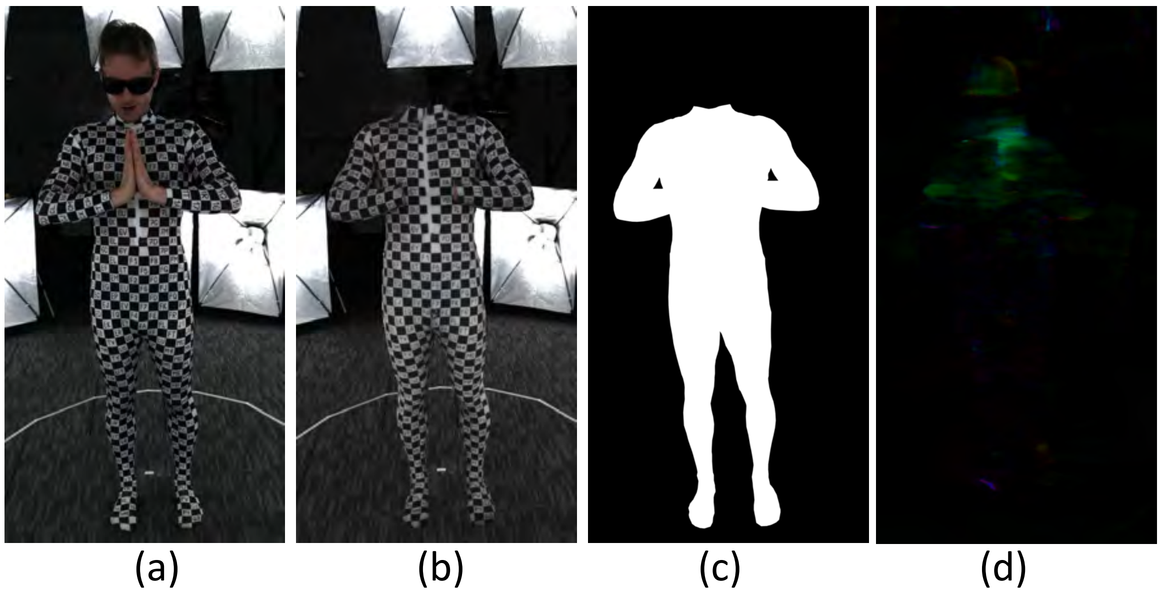
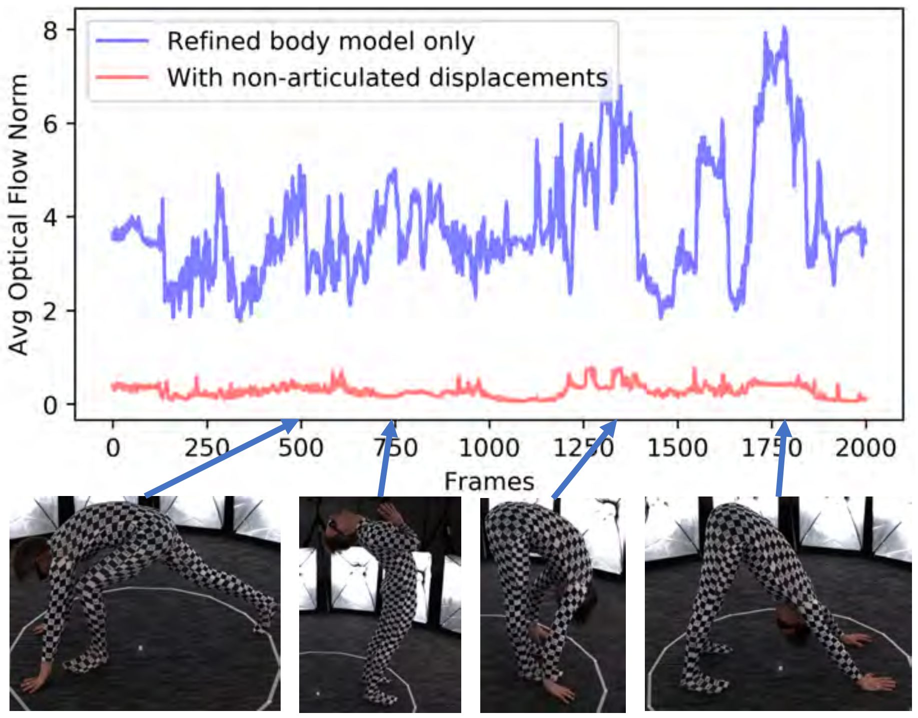
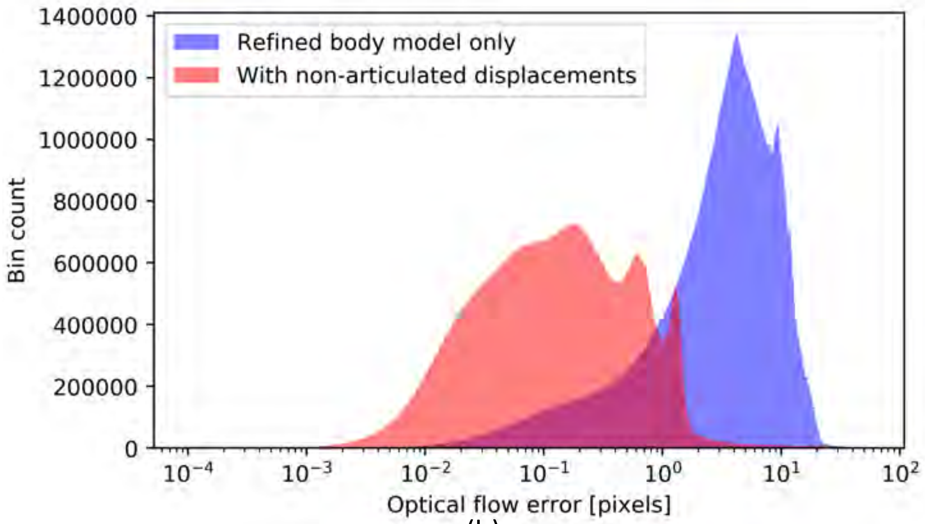


Figure 6.17: Evaluation the accuracy using optical flow. (a) The input image. (b) The synthetic image rendered from our body mesh. (c) The foreground mask of our body mesh. (d) The optical flow between the synthetic image (b) and the real one (a). The angle of flow is visualized by hue and the magnitude of flow by value in HSV color model.



(a)



(b)

Figure 6.18: Optical flow errors. The blue curves correspond to a low-dimensional refined body model only; the red curves correspond to our final results including non-articulated displacements. (a) The plot of average optical flow norm in a motion sequence of 2000 frames (we show four example frames below the graph). (b) Histograms of per-pixel optical flow norms for the same sequence.

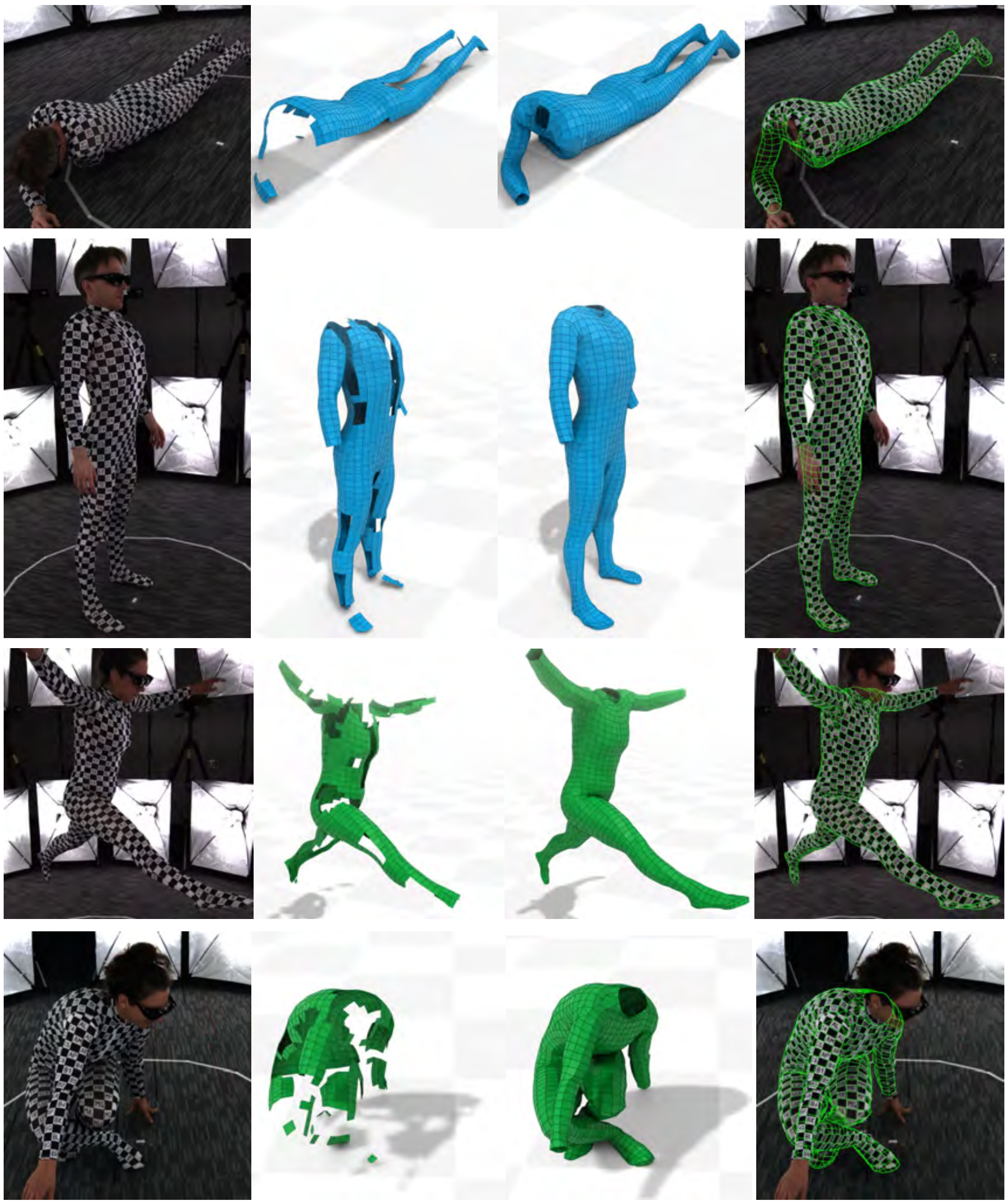


Figure 6.19: Results in challenging poses. From the left to right: 1) input images, 2) raw 3D reconstructions with holes, 3) our final meshes after interpolating non-articulated displacements, and 4) wireframe rendering of the final meshes overlaid with the original images.

Table 6.1: Composition of our training data. The original training set is the training set before data augmentation from 20 annotated images. The test set A is from 4 annotated images of our 3 actors wearing the hand-drawn suit who appeared in the training set. The test set B contains images of fourth actor wearing a suit with printed font.

Type of Data	*CornerdetNet*	*RejectorNet*	*RecogNet*
Original Training Set	667320	21257	7402
Augmented Training Set	5678934	121060	118432
Synthetic Training Set	*NA*	*NA*	214471
Test set A	136500	3372	1245
Test set B (printed suit)	134641	3523	900

(a) Con- fu- sion ma- trix on train- ing set.

n=5678934	Actual True	Actual False
Prediction True	2.533%	0.587%
Prediction False	0.02%	96.86%

(b) Con- fu- sion ma- trix on test set A.

n=136500	Actual True	Actual False
Prediction True	1.7%	0.44%
Prediction False	0.051%	97.81%

(c) Con- fu- sion ma- trix on test set B (printed suit).

n=134641	Actual True	Actual False
Prediction True	1.214%	0.34%
Prediction False	0.021%	98.432%

(d) Max/mean/median of cor- ner lo- cal-

Table 6.3: Confusion matrix on training set.

n=121060	Actual True	Actual False
Prediction True	13.133%	0.243%
Prediction False	0.034%	86.59%

Table 6.4: Confusion matrix on the test set A.

n=3372	Actual True	Actual False
Prediction True	1.305%	0.297%
Prediction False	0.0%	98.399%

Table 6.5: Confusion matrix on the test set B (printed suit)).

n=3523	Actual True	Actual False
Prediction True	2.286%	0%
Prediction False	0.6812%	97.03%

Table 6.6: Classification accuracy for *RecogNet* trained with/without synthetic training data.

With synthetic training set	Classification Accuracy			
	Real Training	Synthetic Training	Test A	Test B
No	99.940%	NA	99.522%	85.16%
Yes	99.967%	94.849%	99.919%	100%

Table 6.7: Comparison of classification accuracies.

	CornerdetNet	Shi-Tomasi	Harris	SuperPoint
True Positives	928	400	867	901
False Negatives	11	540	73	39
False Positives	30	779	12413	3094

Table 6.8: Comparison of mean localization errors.

	CornerdetNet	Shi-Tomasi	Harris	SuperPoint
Localization Error	0.2321	0.3135	0.3081	0.4937

Table 6.9: Comparison of text recognizers' prediction accuracies.

	RecogNet	Tesseract-OCR
test set A	99.919%	19.26%
test set B	100%	61.93%

Table 6.10: Percentiles of reprojection errors computed per camera for all the reconstructed points in a consecutive sequence of 10000 frames.

95%	99%	99.9%	99.99%
0.6979	1.009	1.409	3.376

CHAPTER 7

CONCLUSION

In conclusion, our research has introduced several innovative methods and techniques for various simulation and motion capture challenges, demonstrating significant advancements in efficiency, robustness, and applicability.

- We presented Vertex Block Descent (VBD), an efficient iterative descent-based solution for optimization problems, particularly beneficial for physics-based simulations with implicit Euler integration defined through a variational formulation. We elaborated on the intricacies of elastic body dynamics using VBD, including damping, constraints, collisions, and friction. Notably, we introduced an adaptive initialization technique and momentum-based acceleration to enhance convergence. We also discussed the effective parallelization of VBD, highlighting its vertex-level computation which improves the parallelization of Gauss-Seidel iterations. Our results confirmed VBD's ability to manage highly complex simulation cases, remain stable under extreme conditions, and offer fast convergence.
- We developed a formal definition and algorithm for finding the shortest path to the boundary in the context of self-intersections. This method efficiently handles self-collisions and inter-object collisions by combining DCD with CCD. The algorithm has been tested in highly complex simulation scenarios involving collisions and rest-in-contact conditions, showing robust performance with minimal computational overhead.
- We extended the shortest path algorithm to effectively handle codimensional objects in simulations where penetration must be strictly avoided. Our novel approach involves offsetting the entire surface to ensure that contact forces remain orthogonal, preventing stretching artifacts and allowing for a larger contact distance. This reduces the stiffness of the contact energy, which is a significant issue in existing

methods like Incremental Potential Contact (IPC). Additionally, we introduced a variable max displacement bound for each vertex, allowing those far from contact to fully utilize the solver's descent step and thereby significantly improving convergence. Our techniques are designed as local operations, enabling massive parallelism without the need for global synchronization. This allows for efficient implementation on GPUs, facilitating real-time, large-scale, penetration-free simulations. Both the contact force computation and penetration-prevention technique are local operations, enabling massive parallelism without global synchronization. This, combined with a fully parallel solver, allows for efficient implementation on GPUs, achieving real-time, large-scale, penetration-free simulations.

- We introduced a novel motion capture suit capable of capturing over 1000 uniquely labeled points on a moving human body. This was achieved using a checkerboard-type corner design with two-letter codes for unique labeling, supported by a multi-camera system built from off-the-shelf components. Our approach demonstrated superior cost-efficiency compared to traditional full-body 3DMD setups and handled a wider variety of motions, including gymnastics and yoga poses. The method's independence from temporal coherence enhances its robustness to dis-occlusions and facilitates parallel processing. We have made our code and data publicly available, with plans to release an optimized version as open-source software.

Together, these contributions provide substantial improvements and future directions for physics-based simulations, collision detection & handling, and inverse physics problems.

APPENDIX A

PROOF OF THEOREMS FOR SHORTEST PATH TO SURFACES

A.1 Proof of Theorems

A.1.1 Theorem 1

We first prove this lemma:

Lemma 1. *For any point $\mathbf{p} \in M$, its shortest path to the boundary as $\overline{\mathbf{p}}$ is a line segment.*

Proof. Let's assume that $\mathbf{p}$'s shortest path to boundary as $\overline{\mathbf{p}}$ is not a line segment, and $\mathbf{p}$'s closest boundary point as as $\overline{\mathbf{p}}$ is $\mathbf{s}$ (as $\overline{\mathbf{s}}$). Then there must be a curve on the undeformed pose: $\overline{\mathbf{c}}(t) : I \mapsto \overline{M}, \overline{\mathbf{c}}(0) = \overline{\mathbf{p}}, \overline{\mathbf{c}}(1) = \overline{\mathbf{s}}$, s.t., $\mathbf{c}(t) = \Psi(\overline{\mathbf{c}}(t))$ is $\mathbf{p}$'s shortest path to boundary.

Since $\nabla\Psi > 0$, we know that Ψ is locally bijective. Thus, according to Heine–Borel theorem [216], we can have a limited number of open sets $\mathcal{A}$ covering $\overline{M}$, such that Ψ is bijective on each open set in $\mathcal{A}$. We can select a subset $\mathcal{A}_0 \subseteq \mathcal{A}$, such that $\overline{\mathbf{c}}(I)$ is totally contained by $\overline{A}_0$= the union of $\mathcal{A}_0$; see Figure A.1a.

We then construct a cluster of curves: $\mathbf{h}(u,t) = u\mathbf{c}(t) + (1-u)\mathbf{l}(t) : I \times I \mapsto \overline{M}$, such that, $\mathbf{h}(0,t) = \mathbf{c}(t), \mathbf{h}(1,t) = \mathbf{l}(t), \forall t \in I$, where $\mathbf{l}(t)$ is the line segment from $\mathbf{p}$ to $\mathbf{s}$; see Figure A.1b. $\mathbf{h}(u,t)$ will smoothly deformed from $\mathbf{c}(t)$ to $\mathbf{l}(t)$ as u changes from 0 to 1. Note that any moment u, the length of the curve: $\mathbf{c}_u(t) = \mathbf{h}(u,\cdot)$ must be shorter than $\mathbf{c}(t)$.

Since $\mathbf{c}(I) \in A_0 = \Psi(\overline{A}_0)$, there must exist a $u_0 > 0$, such that, $\mathbf{h}([0,u_0], I) \in A_0$. Additionaly, because Ψ is bijective on A_0, we can define a undeformed pose curve cluster: $\overline{\mathbf{h}}(u,t) = \Psi^{-1}(\mathbf{h}(u,t)) : [0,u_0] \times I \mapsto \overline{A}_0$; as shown in Figure A.1a.

We can then select another group open set $\mathcal{A}_1$ containing $\overline{\mathbf{h}}(u_0, I)$, and repeat the above procedure. This will give us another cluster of curves: $\overline{\mathbf{h}}(u,t) = \Psi^{-1}(\mathbf{h}(u,t)) : [u_0, u_1] \times I \mapsto \overline{A}_0$. During such process, the curve will not touch the boundary of the model, otherwise, there will be a shorter curve connecting $\overline{\mathbf{p}}$ and the boundary, which violates our assumption. Since $\mathcal{A}$ is a limited set, we can eventually obtain a $\overline{\mathbf{h}}(u,t) : [u_k, 1] \times I \mapsto \overline{M}$ within a limited $k+1$ steps, such that $\Psi(\overline{\mathbf{h}}(1,t)) = \mathbf{l}(t), \forall t \in I, \overline{\mathbf{h}}(1,0) = \overline{\mathbf{p}}$ and $\overline{\mathbf{h}}(1,1) = \overline{\mathbf{s}}$.

Here we have proved that $\mathbf{l}(t)$ is a valid path, which must be shorter than $\mathbf{c}(t)$ due to Euclidean metrics. Hence creating a contradiction. □

With Lemma 1, the proof of Theorem 1 becomes trivial. Since the shortest path to the boundary must be a valid path, of course it should be the *shortest* valid line segment to boundary.

A.1.2 Theorem 2

The proof of Theorem 2 is similar to Theorem 1. Say the closest boundary point is $\mathbf{s}' \in f$ (as $\overline{\mathbf{s}}'$) and the Euclidean closest boundary point on f is $\mathbf{s}$ (as $\overline{\mathbf{s}}$), where f is a boundary face. We also construct a cluster of curves: $\mathbf{h}(u,t) = u\mathbf{l}(t) + (1-u)\mathbf{l}'(t) I \times I \mapsto \overline{M}$, where $\mathbf{l}'(t)$ and $\mathbf{l}(t)$ are the line segment from $\mathbf{p}$ to $\mathbf{s}'$ and $\mathbf{s}$, respectively. This $\mathbf{h}(u,t)$ also holds the property that at any given moment u, the length of the curve: $\mathbf{c}_u(t) = \mathbf{h}(u,\cdot)$ must be shorter than $\mathbf{l}'(t)$.

Note that instead of fixing two ends, we only fix one end of $\mathbf{h}(u,t)$, as it deforms from $\mathbf{l}'(t)$ to $\mathbf{l}(t)$. Because Ψ is bijective on each boundary face, we can explicitly construct the line segment on the undeformed pose that goes from $\overline{\mathbf{s}}'$ and $\overline{\mathbf{s}}$, this allows us to move the position of the end point.

Similar to Lemma 1, we can induce a cluster of curves on the undeformed pose: $\overline{\mathbf{h}}(u,t)$, which will give us the pre-image of $\mathbf{l}(t)$ as $\overline{\mathbf{h}}(1,t)$, connecting $\overline{\mathbf{p}}$ and $\overline{\mathbf{s}}$. Hence we have proven that $\mathbf{l}(t)$ is also a valid path from $\overline{\mathbf{p}}$ to $\overline{\mathbf{s}}$. This contradicts the assumption that $\mathbf{s}'$ is the closest boundary point.

A.1.3 Element Traverse and Valid Path

We can give an equivalent definition of a curve being a valid path in the discrete case.

Theorem 3. *A line segment connecting $\mathbf{a} \in e_a$ and $\mathbf{b} \in e_b$ in a mesh, is a valid path if and only it is included by element traverse from e_a to e_b.*

Proof. Sufficiency. If there exists such a element traverse $\mathcal{T}(\mathbf{a}, \mathbf{b}) = (e_1, e_2, e_3, \ldots, e_{k-1}, e_k)$, s.t., $e_0 = e_a$ and $e_k = e_b$, we can explicitly construct a continuous piece-wise linear curve

$\bar{\mathbf{c}}(t)$ defined on them, whose image is $\mathbf{c}(t)$. We do this by making a division of I: $I = [t_0, t_1] \cup [t_1, t_2] \cup [t_2, t_3] \cup \cdots \cup [t_{k-1}, t_k]$, where $t_0 = 0$, $t_k = 1, t_0 \le t_1 \le t_2 \le \cdots \le t_k$. The division can be obtained by making $\frac{t_i}{t_{i+1}} = \frac{|\bar{\mathbf{r}}_i - \bar{\mathbf{r}}_{\bar{\mathbf{r}}+1}|}{|\bar{\mathbf{r}}_{i+1} - \bar{\mathbf{r}}_{i+2}|}, \forall i = 1, 2, \ldots, k+1$, where $\bar{\mathbf{r}}_i = \Psi|_{e_i}^{-1}(\mathbf{r}_i)$ is the preimage of the line segment's exit point from e_i. The curve can be constructed as:

$$\bar{\mathbf{c}}(t) = \frac{t - t_i}{t_{i+1} - t_i}\bar{\mathbf{r}}_i + (1 - \frac{t - t_i}{t_{i+1} - t_i})\bar{\mathbf{r}}_{i+1}, \text{if } t \in [t_i, t_{i+1}] \tag{A.1}$$

Necessity. Suppose we have a curve on the undeformed pose $\bar{\mathbf{c}}(t)$ connecting $\bar{\mathbf{a}}, \bar{\mathbf{b}}$, whose image under Ψ is a line segment. If $\bar{\mathbf{c}}(t)$ passes no vertex of $\overline{M}$, we directly obtain an element traversal by enumerating the elements that $\bar{\mathbf{c}}(t)$ passes by as t continuously changes from 0 to 1.

When $\bar{\mathbf{c}}(t)$ passes a vertex $\bar{\mathbf{v}}$ of $\overline{M}$, assume it goes from e_i to e_{i+1} at that point. According to the definition of a manifold, we can search around that $\bar{\mathbf{v}}$ and guarantee to have an element traversal from e_i to e_{i+1} formed by elements adjacent to $\bar{\mathbf{v}}$.

The case of $\bar{\mathbf{c}}(t)$ passing an edge can be proved similarly.

□

A.1.4 Further Discussion on Inverted Elements

The line segment **sp** is only a subset of such a path constructed by our algorithm. In fact, in this case, the length of path $\mathbf{c}(t)$ is evaluated by this formula:

$$\int_0^1 sign((\mathbf{s} - \mathbf{p})r'(t)|r'(t)|dt \tag{A.2}$$

which means, in the presence of inverted tetrahedra, that the length of $\mathbf{c}(t)$ grows as it goes in the direction of $\mathbf{s} - \mathbf{p}$, and decreases if it goes in the opposite direction, which happens when it passes through the inverted tetrahedron. This is understandable because when solving the self-intersection, the penetrated point does not need to go back and forth, it only needs to pass through the overlapping part once. Thus the length of the overlapping part should only count once. With inverted tetrahedra, our algorithm is actually constructing the shortest line segment connecting **p** and a surface point under the metrics introduced by Eq.A.2.

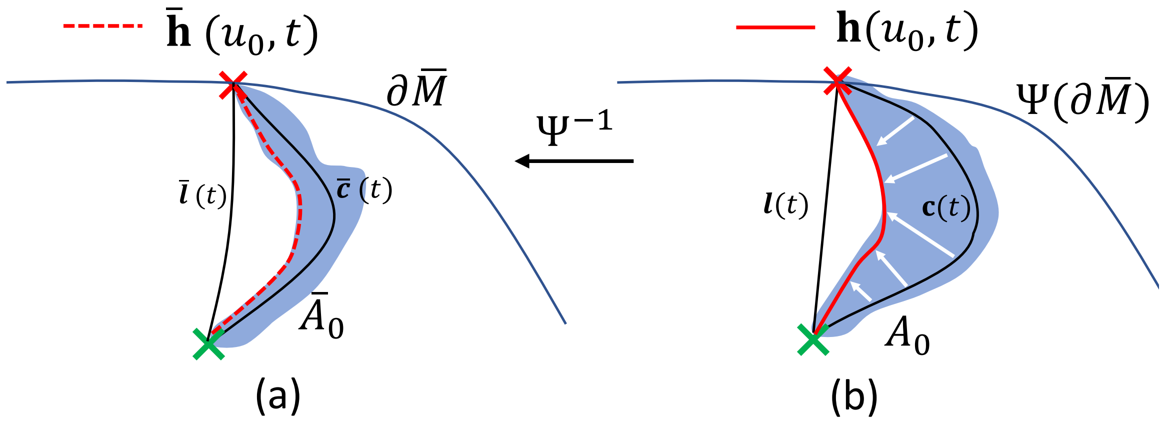


Figure A.1: Constructing the undeformed pose curve. (a) The undeformed pose. (b) The deformed pose.

APPENDIX B

DATA GENERATION AND ANNOTATION FOR OUR CAPTURE SYSTEM

B.1 CNN Structures

B.1.1 *CornerdetNet*

The input to *CornerdetNet* is the 8×8 cell where a corner is being sought (Figure B.1), including a 6-pixel margins added to each side (Figure B.2b), making the input crop size 20×20. These margins allow us reliably detect even the corners close to the boundaries of the 8×8 cell. The 6-pixel margins overlap with adjacent cells (Figure B.2b), but the 8×8 cells do not overlap. The *CornerdetNet* outputs three floating-point numbers. The first one is a logit of a binary classifier predicting whether a corner is present or not, and the other two are normalized coordinates ($[0, 1] \times [0, 1]$) of the corner relative to the 8×8 cell. The training loss for *CornerdetNet* is:

$$\mathcal{L}_c(p^*, p; \mathbf{c}^*, \mathbf{c}) = \mathcal{L}_p(p^*, p) + \lambda_c p ||\mathbf{c} - \mathbf{c}^*||_2^2 \tag{B.1}$$

where λ_c balances the prediction loss and localization loss; we set $\lambda_c = 200$ when training *CornerdetNet*. p^* represents the logit of the binary classifier, c^* represents the prediction of corner location, p and c represents the ground truth respectively, and $\mathcal{L}_p(p^*, p)$ is cross entropy.

B.1.1.1 Corner Clustering and Refinement

When a corner lies exactly on the boundary of two 8×8 cells, it can be detected more than once (Figure B.1a). To fix such duplicate detections, we perform a clustering pass: if any two detected corners are too close (¡ 3 pixels), we discard the one with the lower logit value. Since this might introduce additional localization noise, we generate new crops randomly perturbed around the original corner positions, run localization on each of these crops and average the results in global pixel coordinates; see Figure B.1b. This

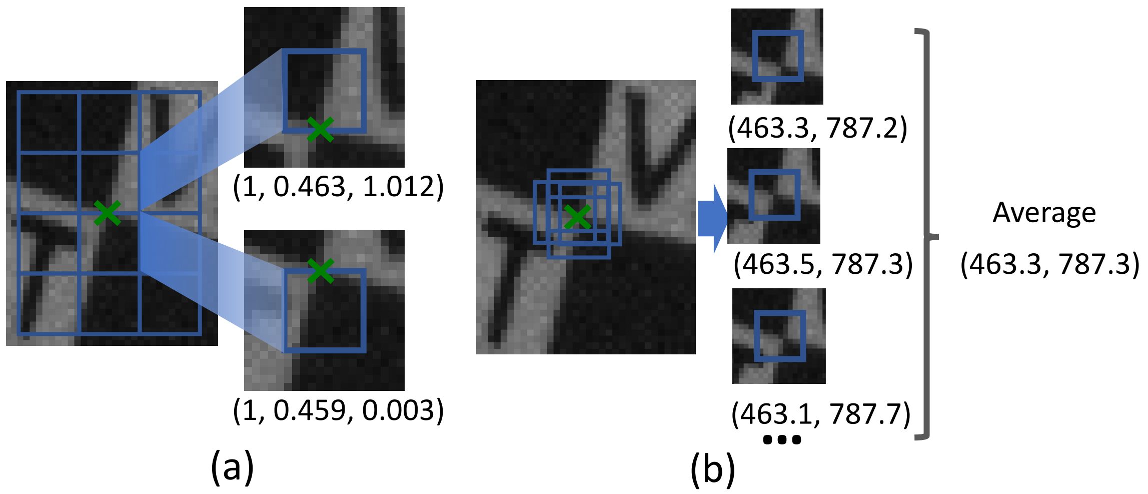


Figure B.1: Design of our corner detector. (a) A corner is detected twice because it is on the boundary between 2 cells. In the parentheses are the *CornerdetNet* outputs on each crop. (b) The re-crops generated for a detected corner. In the parentheses are the corner position in global pixel coordinates.

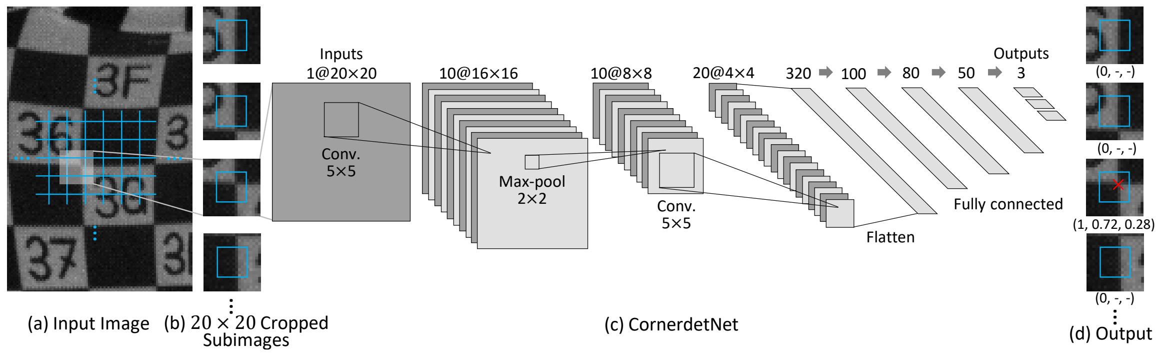


Figure B.2: Architecture of *CornerdetNet*. (a) The input image is divided into a regular grid of 8 × 8 cells. (b) Each cell is expanded by a margin and the corresponding expanded crop is passed as input to *CornerdetNet*. (c) The architecture of our *CornerdetNet*. (d) Example outputs; even though there is a corner in the second image, it is outside of the inner 8 × 8 cell, thus the detector correctly reports 0 (no corner).

helps especially when corners are crossing the boundaries of the 8 × 8 cells (Figure B.1a).

B.1.2 Quad Classifiers

B.1.2.1 Candidate Quad Generation

It would be wasteful to enumerate all 4-tuples of corners for further processing by neural networks. Therefore, we first apply simple criteria to filter out quads that cannot contain a valid code. We start by iterating over all the corners, and for each corner,

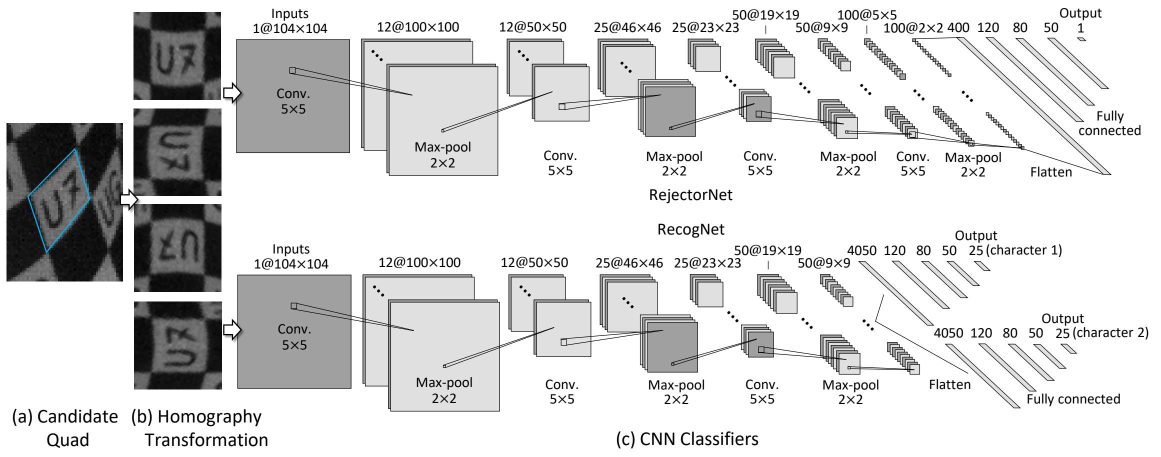


Figure B.3: Our quad processing pipeline: (a) Example candidate quad. (b) The undistorted candidate quads with margin; these images are input to our quad classifiers. (c) The architectures of *RejectorNet* and *RecogNet* CNNs.

we select three other corners within a bounding box. When connecting corners into a quad, we ensure that each quad is convex, clock-wise oriented and unique. Additional filtering criteria include geometric criteria and image based criteria: geometric criteria constrain the area, maximum/minimum edge-lengths and maximum/minimum angles of the generated candidate quad; image based criteria constrain the average intensity, and standard deviation of all the pixels in the generated candidate quad. To obtain the range for each criterion, we gather statistics for each of those quantities in the training dataset (Section 6.3) and create conservative intervals to ensure that we cannot mistakenly reject any valid quad. The candidate quads that pass all of these early rejection filters are transformed using homography and passed to further processing to quad classifier neural networks. Figure 6.6c shows an example of an invalid quad and Figure 6.6d demonstrates a valid one. Our quad processing pipeline is visualized in Figure B.3

B.1.2.2 Why Separate *RejectorNet* and *RecogNet*?

We considered combining the two networks into one, but we found that network training is easier if we treat each problem separately. Specifically, the *RejectorNet* should perform quality control of a 104×104 standardized image, including rejection of errors made by *CornerdetNet* (Figure B.6b). Because we prefer missing observations to errors, we train *RejectorNet* to be conservative and reject any inputs of dubious quality. The second network, *RecogNet*, has to recognize two characters in any image. We can make *RecogNet* more

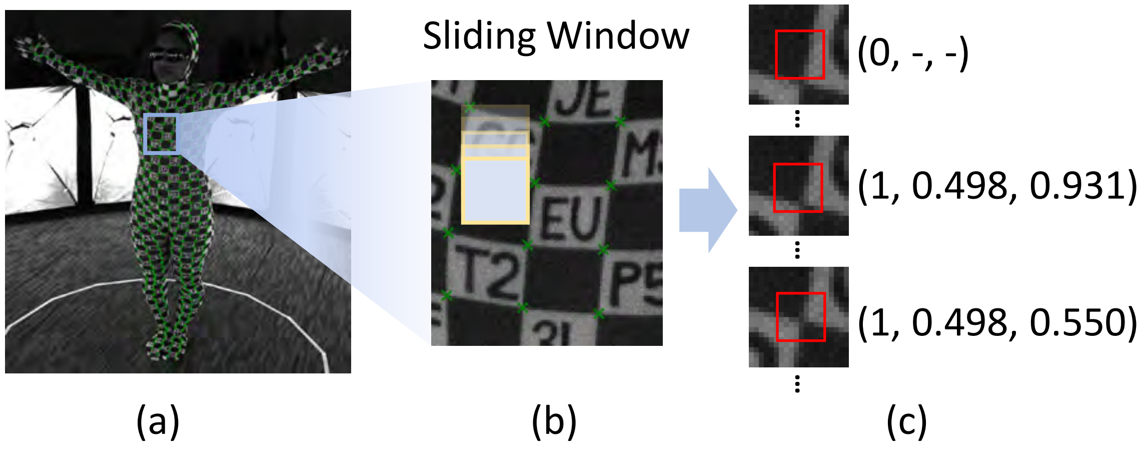


Figure B.4: Generating training data for *CornerdetNet*: (a) The annotated image. (b) Sliding a 20×20 window across the annotated image. For each position of the window we generate a crop – input for *CornerdetNet*. (c) The crop is a positive sample $(1, x, y)$ for *CornerdetNet* if it contains a valid checkerboard corner in its center 8×8 pixels (red square); otherwise it is a negative sample (0, -, -).

reliable by training it even on very difficult input images, enhancing the robustness of the entire pipeline. The details of our training process and data augmentation are discussed in Section B.2.2.

B.2 Data Acquisition

B.2.1 Data Conversion

B.2.1.1 Corner Detector

We generate the training data for *CornerdetNet* by sliding a 20×20 window with stride 1, as shown in Figure B.4b. Each of the 20×20 crops is an input to *CornerdetNet*, labeled positive if and only if an annotated corner lies inside its center 8×8 pixels. For positive samples we also compute the sub-pixel corner coordinates relative to the 8×8 cell.

B.2.1.2 Quad Classifiers

We start by generating candidate quads from the annotated corners in the manually annotated images using the algorithm discussed in Section B.1.2. Note that the same quad generation algorithm will be used during deployment, i.e., when processing new motion sequences. The quad generator is conservative and creates many quads that do not correspond to valid white squares; see Figure B.5c. However, we know which quads are valid, because all of the valid ones were manually annotated; see Figure B.5a. This allows

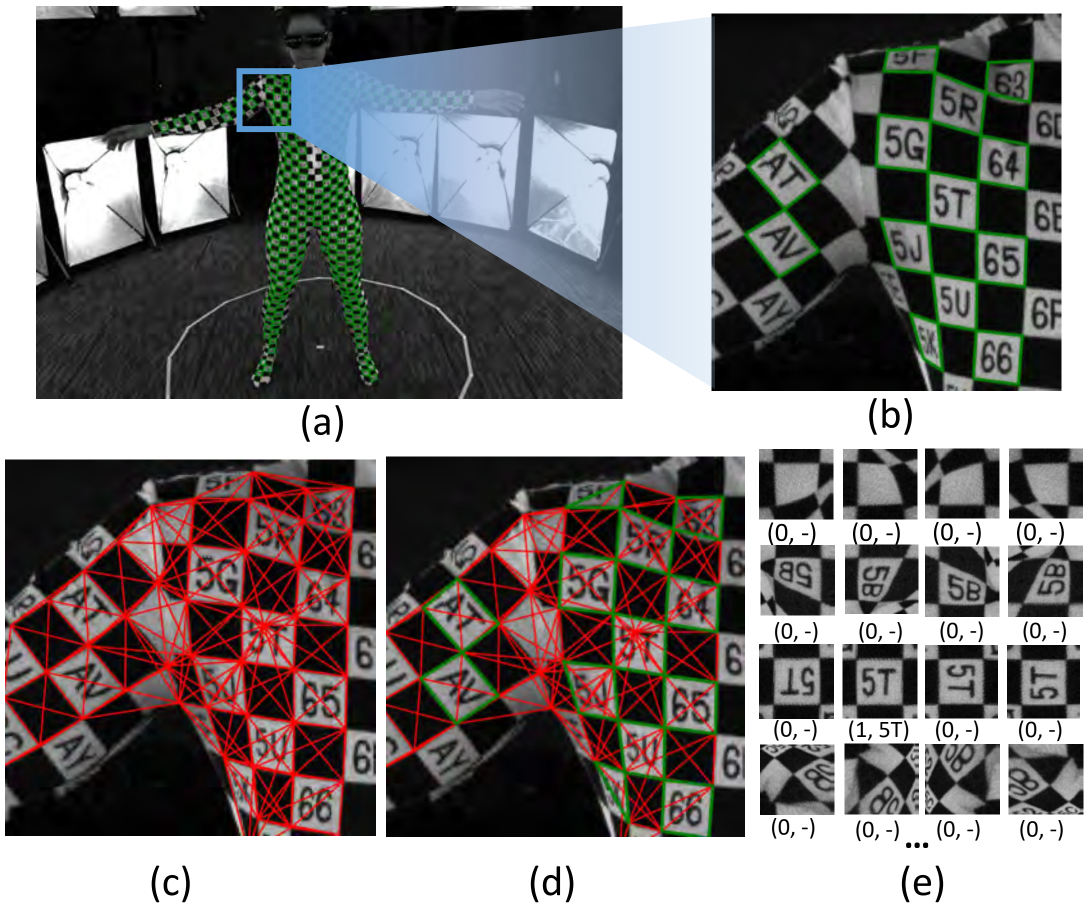


Figure B.5: Generation of quad annotations. (a, b): Manual quad annotations. (c) The candidate quads generated using the algorithm from Section 6.2.5. (d) Selection of valid quads. (e) Homography-transformed quads with four possible rotations, including ground truth labels for training *RejectorNet* and *RecogNet*.

us to automatically generate both positive and negative examples for a given candidate-quad generator; see Figure B.5d. These 104×104 images are used to train *RejectorNet*. It is important that the quad generator used during deployment is identical to the quad generator used when generating the training data for *RejectorNet*. The two-letter code annotations of the valid quads are then using to train *RecogNet*.

B.2.2 Data Augmentation and Synthetic Data

B.2.2.1 Data Augmentation

All of the crops generated from annotated images as described in the Section B.2.1 are augmented by applying intensity perturbations (contrast, brightness, gamma). In addition, we also apply geometric deformations on each input image. For the corner detector, we also augment the training data by generating random rotations of each image, because checkerboard-like corners are rotation invariant.

Different data augmentation approaches need to be applied to *RejectorNet* and *RecogNet*. For the *RejectorNet*, we blur the image using Gaussian filter and add elastic deformation using thin-plate splines [217] to simulate skin deformation. We constrain the elastic deformations to fix the checkerboard-like corners in place; see Figure B.6a, otherwise positive examples could be turned into negative ones. We also use this fact to our advantage: if we displace a checkerboard-like corner of a valid white square, we obtain a new (augmented) negative example, simulating the case when quad's corners have not been correctly localized; see Figure B.6b.

Since *RecogNet* is required to predict characters from any input image, we can afford to augment our data more aggressively. Specifically, we use much more significant geometric distortions, intensity variations, blurring and additional noise; see Figure B.6c. This aggressive data augmentation has an interesting effect: the performance on the training data becomes worse, since we made the recognition task more difficult. However, we obtain better performance on the *test* set, which is what matters. This agrees with human intuition: if students are given harder homework (training), they will likely perform better in their first job.

B.2.2.2 Synthetic Data

To further enhance the diversity of our training data, we also generated synthetic data sets by rendering an animated SMPL [134] model. We use synthetic data only for training the *RecogNet*, because this was the bottleneck in the overall pipeline. We textured the body mesh with the same checkerboard-like pattern as used in the real suit and applied animations from a public motion capture database [218]. We randomly generated new two-letter codes, including variations in font types and sizes to emulate the handwriting of the codes. For each animation frame, we rendered images with virtual cameras, simulating our real capture setup by copying the intrinsic and extrinsic parameters from our real cameras. The visibility of corners in the rendered images is determined using ray tracing. To control the quality of quads that will be added to the training set, we check for corner visibility and use a classifier considering the quad's 3D normal direction and quad geometry in the rendered image.

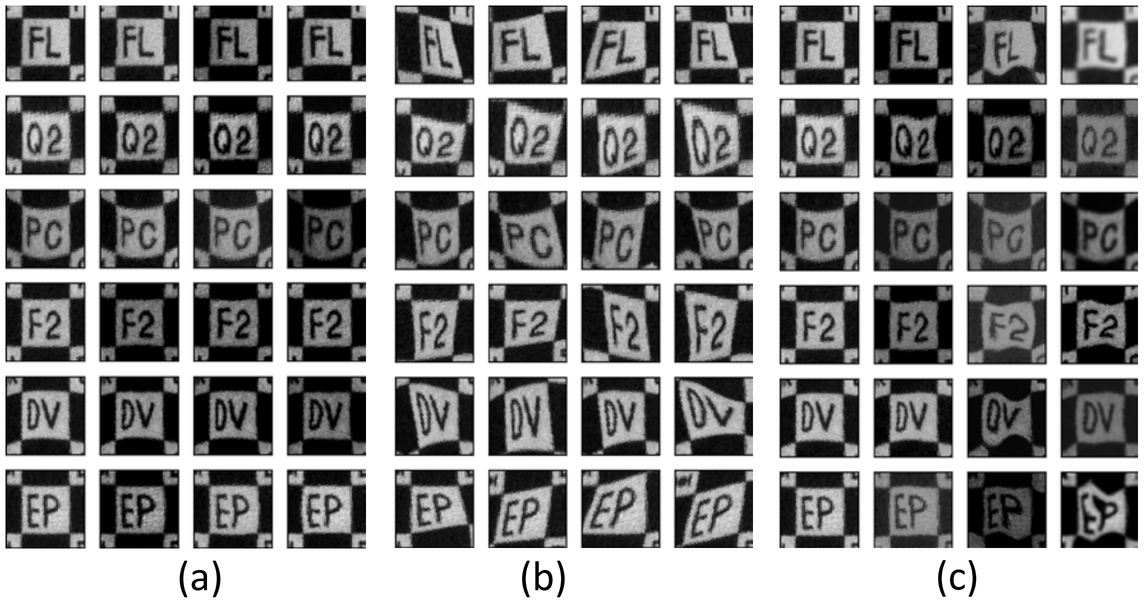


Figure B.6: Visualization of data fed to *RejectorNet*. (a) The augmented positive training data for *RejectorNet*. (b) We generate negative training samples for *RejectorNet* by warping one or more corners of a positive sample away from its original location, which simulates the case that the quad's corners were not correctly localized. (c) The data augmentation for *RecogNet* is aggressive, including significant blurring and large elastic deformations.

REFERENCES

[1] A. H. Chen, Z. Liu, Y. Yang, and C. Yuksel, "Vertex block descent," *ACM Trans. Graph.*, vol. 43, no. 4, p. 116, July 2024. [Online]. Available: https://doi.org/10.1145/3658179

[2] M. Li et al., "Incremental potential contact: intersection-and inversion-free, large-deformation dynamics," *ACM Trans. Graph.*, vol. 39, no. 4, p. 49, August 2020. [Online]. Available: https://doi.org/10.1145/3386569.3392425

[3] M. Macklin, K. Erleben, M. Müller, N. Chentanez, S. Jeschke, and Z. Corse, "Local optimization for robust signed distance field collision," *Proc. ACM Comput. Graph. Interact. Tech.*, vol. 3, no. 1, p. 8, May 2020. [Online]. Available: https://doi.org/10.1145/3384538

[4] A. McAdams et al., "Efficient elasticity for character skinning with contact and collisions," *ACM Trans. Graph.*, vol. 30, no. 4, p. 37, July 2011. [Online]. Available: https://doi.org/10.1145/2010324.1964932

[5] D. Baraff and A. P. Witkin, "Large steps in cloth simulation," in *Proc. of the 25th Annu. Conf. on Comput. Graph. and Interactive Techn., SIGGRAPH 1998, Orlando, FL, USA, July 19-24, 1998*, ser. SIGGRAPH '98, 1998, pp. 43–54. [Online]. Available: https://doi.org/10.1145/280814.280821

[6] G. Hirota, S. Fisher, A. State, C. Lee, and H. Fuchs, "An implicit finite element method for elastic solids in contact," in *The 14th Conf. on Comput. Animation, CA 2001, Seoul, South Korea, November 7-8, 2001*, 2001, pp. 136–254. [Online]. Available: https://doi.org/10.1109/CA.2001.982387

[7] P. Volino and N. Magnenat-Thalmann, "Comparing efficiency of integration methods for cloth simulation," in *Comput. Graph. Int. 2001 (CGI'01), July 3-6, 2001, Hong Kong, China, Proc.*, 2001, pp. 265–272. [Online]. Available: https://doi.org/10.1109/CGI.2001.934683

[8] S. Martin, B. Thomaszewski, E. Grinspun, and M. H. Gross, "Example-based elastic materials," in *ACM SIGGRAPH 2011 Papers*, ser. SIGGRAPH '11, vol. 30, no. 4, August 2011, p. 72. [Online]. Available: https://doi.org/10.1145/2010324.1964967

[9] C. Kane, J. E. Marsden, M. Ortiz, and M. West, "Variational integrators and the newmark algorithm for conservative and dissipative mechanical systems," *Int. J. for Numer. Methods in Eng.*, vol. 49, no. 10, pp. 1295–1325, 2000. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/1097-0207%2820001210%2949%3A10%3C1295%3A%3AAID-NME993%3E3.0.CO%3B2-W

[10] R. Bridson, R. Fedkiw, and J. Anderson, "Robust treatment of collisions, contact and friction for cloth animat." *ACM Trans. Graph.*, vol. 21, no. 3, p. 594–603, July 2002. [Online]. Available: https://doi.org/10.1145/566654.566623

[11] R. Bridson, S. Marino, and R. Fedkiw, "Simulation of clothing with folds and wrinkles," in *Int. Conf. on Comput. Graph. and Interactive Techn., SIGGRAPH 2005, Los Angeles, California, USA, July 31 - August 4, 2005, Courses*, July 2005, p. 3. [Online]. Available: https://doi.org/10.1145/1198555.1198573

[12] K. Choi and H. Ko, "Stable but responsive cloth," in *Int. Conf. on Comput. Graph. and Interactive Techn., SIGGRAPH 2005, Los Angeles, California, USA, July 31 - August 4, 2005, Courses*, 2005, p. 1. [Online]. Available: https://doi.org/10.1145/1198555.1198571

[13] M. Hauth and O. Etzmuss, "A High Performance Solver for the Animat. of Deformable Objects using Advanced Numerical Methods," *Comput. Graph. Forum*, vol. 20, no. 3, pp. 319–328, September 2001. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1111/1467-8659.00524

[14] B. Eberhardt, O. Etzmuß, and M. Hauth, "Implicit-explicit schemes for fast animation with particle systems," in *Proc. of the Eurographics Workshop on Comput. Animation and Simul. 2000, Interlaken, Switzerland, August 21-22, 2000*, ser. Eurographics, 2000, pp. 137–151, series Title: EuroGraph. [Online]. Available: https://doi.org/10.1007/978-3-7091-6344-3_11

[15] A. Stern and E. Grinspun, "Implicit-explicit variational integration of highly oscillatory problems," *Multiscale Model. & Simul.*, vol. 7, no. 4, pp. 1779–1794, 2009. [Online]. Available: https://doi.org/10.1137/080732936

[16] U. M. Ascher and E. Boxerman, "On the modified conjugate gradient method in cloth simulation," *The Vis. Comput.*, vol. 19, pp. 526–531, 2003.

[17] J. Teran, E. Sifakis, G. Irving, and R. Fedkiw, "Robust quasistatic finite elements and flesh simulation," in *Proc. of the 2005 ACM SIGGRAPH/Eurographics Symp. on Comput. Animation, SCA 2005, Los Angeles, CA, USA, July 29-31, 2005*, ser. SCA '05, 2005, pp. 181–190. [Online]. Available: https://doi.org/10.2312/SCA/SCA05/181-190

[18] F. Hecht, Y. J. Lee, J. R. Shewchuk, and J. F. O'Brien, "Updated sparse cholesky factors for corotational elastodynamics," *ACM Trans. Graph.*, vol. 31, no. 5, pp. 1–13, August 2012. [Online]. Available: https://dl.acm.org/doi/10.1145/2231816.2231821

[19] E. Grinspun, P. Krysl, and P. Schröder, "Charms: a simple framework for adaptive simulation," *ACM Trans. Graph.*, vol. 21, no. 3, p. 281–290, July 2002. [Online]. Available: https://doi.org/10.1145/566654.566578

[20] S. Capell, S. Green, B. Curless, T. Duchamp, and Z. Popovic, "A multiresolution framework for dynamic deformations," in *Proc. of the 2002 ACM SIGGRAPH/Eurographics Symp. on Comput. Animation, San Antonio, TX, USA, July 21-22, 2002*, July 2002, pp. 41–47. [Online]. Available: https://doi.org/10.1145/545261.545268

[21] J. Bolz, I. Farmer, E. Grinspun, and P. Schröder, "Sparse matrix solvers on the gpu: Conjugate gradients and multigrid," *ACM Trans. Graph.*, vol. 22, no. 3, p. 917–924, July 2003. [Online]. Available: https://doi.org/10.1145/882262.882364

[22] R. Tamstorf, T. Jones, and S. F. McCormick, "Smoothed aggregation multigrid for cloth simulation," *ACM Trans. Graph.*, vol. 34, no. 6, p. 245, November 2015. [Online]. Available: https://doi.org/10.1145/2816795.2818081

[23] Z. Xian, X. Tong, and T. Liu, "A scalable galerkin multigrid method for real-time simulation of deformable objects," *ACM Trans. Graph.*, vol. 38, no. 6, p. 162, November 2019. [Online]. Available: https://doi.org/10.1145/3355089.3356486

[24] X. Wang et al., "Hierarchical optimization time integration for cfl-rate mpm stepping," *ACM Trans. Graph.*, vol. 39, no. 3, p. 21, April 2020. [Online]. Available: https://doi.org/10.1145/3386760

[25] Y. Zhu, E. Sifakis, J. Teran, and A. Brandt, "An efficient multigrid method for the simulation of high-resolution elastic solids," *ACM Trans. Graph.*, vol. 29, no. 2, p. 16, April 2010. [Online]. Available: https://doi.org/10.1145/1731047.1731054

[26] M. Müller, J. Dorsey, L. McMillan, R. Jagnow, and B. Cutler, "Stable real-time deformations," in *Proc. of the 2002 ACM SIGGRAPH/Eurographics Symp. on Comput. Animation, San Antonio, TX, USA, July 21-22, 2002*, ser. SCA '02, 2002, pp. 49–54. [Online]. Available: https://doi.org/10.1145/545261.545269

[27] T. Liu, S. Bouaziz, and L. Kavan, "Quasi-newton methods for real-time simulation of hyperelastic materials," *ACM Trans. Graph.*, vol. 36, no. 4, p. 116a, July 2017. [Online]. Available: https://doi.org/10.1145/3072959.2990496

[28] I. Chao, U. Pinkall, P. Sanan, and P. Schröder, "A simple geometric model for elastic deformations," *ACM Trans. Graph.*, vol. 29, no. 4, p. 38, July 2010. [Online]. Available: https://doi.org/10.1145/1778765.1778775

[29] M. Müller, B. Heidelberger, M. Teschner, and M. Gross, "Meshless deformations based on shape matching," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 471–478, 2005.

[30] S. Bouaziz, S. Martin, T. Liu, L. Kavan, and M. Pauly, "Projective dynamics: fusing constraint projections for fast simulation," *ACM Trans. Graph.*, vol. 33, no. 4, p. 154, July 2014. [Online]. Available: https://doi.org/10.1145/2601097.2601116

[31] C. Kane, J. E. Marsden, and M. Ortiz, "Symplectic-energy-momentum preserving variational integrators," *J. of Math. Phys.*, vol. 40, no. 7, pp. 3353–3371, July 1999. [Online]. Available: https://pubs.aip.org/jmp/article/40/7/3353/231286/Symplectic-energy-momentum-preserving-variational

[32] J. Simo, N. Tarnow, and K. Wong, "Exact energy-momentum conserving algorithms and symplectic schemes for nonlinear dynamics," *Comput. Methods in Appl. Mechanics and Eng.*, vol. 100, no. 1, pp. 63–116, October 1992. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/004578259290115Z

[33] A. Lew, J. E. Marsden, M. Ortiz, and M. West, "Variational time integrators," *Int. J. for Numer. Methods in Eng.*, vol. 60, no. 1, pp. 153–212, May 2004. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1002/nme.958

[34] L. Kharevych, W. Yang, Y. Tong, E. Kanso, J. E. Marsden, P. Schröder, and M. Desbrun, "Geometric, variational integrators for computer animation," in *Proc.*

of the 2006 ACM SIGGRAPH/Eurographics Symp. on Comput. Animation, SCA 2006, Vienna, Austria, September 2-4, 2006, ser. SCA '06, 2006, pp. 43–51. [Online]. Available: https://doi.org/10.2312/SCA/SCA06/043-051

[35] T. F. Gast, C. Schroeder, A. Stomakhin, C. Jiang, and J. M. Teran, "Optimization Integrator for Large Time Steps," *IEEE Trans.on Vis.ization and Comput. Graph.*, vol. 21, no. 10, pp. 1103–1115, October 2015. [Online]. Available: http://ieeexplore.ieee.org/document/7164346/

[36] M. Li, M. Gao, T. Langlois, C. Jiang, and D. M. Kaufman, "Decomposed optimization time integrator for large-step elastodynamics," *ACM Trans. Graph.*, vol. 38, no. 4, p. 70, July 2019. [Online]. Available: https://doi.org/10.1145/3306346.3322951

[37] M. Müller, B. Heidelberger, M. Hennix, and J. Ratcliff, "Position based dynamics," *J. Vis. Comun. Image Represent.*, vol. 18, no. 2, p. 109–118, April 2007. [Online]. Available: https://doi.org/10.1016/j.jvcir.2007.01.005

[38] M. Macklin, M. Müller, and N. Chentanez, "XPBD: position-based simulation of compliant constrained dynamics," in *Proc. of the 9th Int. Conf. on Motion in Games, MIG 2016, Burlingame, California, USA, October 10-12, 2016*, October 2016, pp. 49–54. [Online]. Available: https://doi.org/10.1145/2994258.2994272

[39] M. Fratarcangeli and F. Pellacini, "Scalable Partitioning for Parallel Position Based Dynamics," *Comput. Graph. Forum*, vol. 34, no. 2, pp. 405–413, May 2015. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1111/cgf.12570

[40] M. Fratarcangeli, V. Tibaldo, and F. Pellacini, "Vivace: A practical Gauss-Seidel method for stable soft body dynamics," *ACM Trans. Graph.*, vol. 35, no. 6, p. 214, December 2016. [Online]. Available: https://doi.org/10.1145/2980179.2982437

[41] Q.-M. Ton-That, P. G. Kry, and S. Andrews, "Parallel block Neo-Hookean XPBD using graph clustering," *Comput.s & Graph.*, vol. 110, pp. 1–10, February 2023. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S009784932200187X

[42] H. Wang, "A chebyshev semi-iterative approach for accelerating projective and position-based dynamics," *ACM Trans. Graph.*, vol. 34, no. 6, pp. 1–9, November 2015. [Online]. Available: https://dl.acm.org/doi/10.1145/2816795.2818063

[43] P. Huthwaite, "Accelerated finite element elastodynamic simulations using the gpu," *J. Comput. Phys.*, vol. 257, no. PA, p. 687–707, January 2014.

[44] H. Wang and Y. Yang, "Descent methods for elastic body simulation on the gpu," *ACM Trans. Graph.*, vol. 35, no. 6, p. 212, December 2016. [Online]. Available: https://doi.org/10.1145/2980179.2980236

[45] L. Lan, G. Ma, Y. Yang, C. Zheng, M. Li, and C. Jiang, "Penetration-free projective dynamics on the gpu," *ACM Trans. Graph.*, vol. 41, no. 4, p. 69, July 2022. [Online]. Available: https://doi.org/10.1145/3528223.3530069

[46] M. Macklin, K. Erleben, M. Müller, N. Chentanez, S. Jeschke, and T. Kim, "Primal/Dual Descent Methods for Dynamics," *Comput. Graph. Forum*, vol. 39, no. 8, pp. 89–100, December 2020. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1111/cgf.14104

[47] C. Li, M. Tang, R. Tong, M. Cai, J. Zhao, and D. Manocha, "P-cloth: interactive complex cloth simulation on multi-gpu systems using dynamic matrix assembly and pipelined implicit integrators," *ACM Trans. Graph.*, vol. 39, no. 6, p. 180, November 2020. [Online]. Available: https://doi.org/10.1145/3414685.3417763

[48] X. Li, Y. Fang, L. Lan, H. Wang, Y. Yang, M. Li, and C. Jiang, "Subspace-preconditioned GPU projective dynamics with contact for cloth simulation," in *SIGGRAPH Asia 2023 Conf. Papers, SA 2023, Sydney, NSW, Aust., December 12-15, 2023*, ser. SA '23, 2023, pp. 1:1–1:12. [Online]. Available: https://doi.org/10.1145/3610548.3618157

[49] S. J. Wright, "Coordinate descent algorithms," *Math. programming*, vol. 151, no. 1, pp. 3–34, 2015.

[50] A. Naitsat, Y. Zhu, and Y. Y. Zeevi, "Adaptive block coordinate descent for distortion optimization," vol. 39, no. 6, 2020, pp. 360–376. [Online]. Available: https://doi.org/10.1111/cgf.14043

[51] L. Lan, M. Li, C. Jiang, H. Wang, and Y. Yang, "Second-order stencil descent for interior-point hyperelasticity," *ACM Trans. Graph.*, vol. 42, no. 4, p. 108, July 2023. [Online]. Available: https://doi.org/10.1145/3592104

[52] J. Allard, F. Faure, H. Courtecuisse, F. Falipou, C. Duriez, and P. G. Kry, "Volume contact constraints at arbitrary resolution," *ACM Trans. Graph.*, vol. 29, no. 4, p. 82, July 2010. [Online]. Available: https://doi.org/10.1145/1778765.1778819

[53] B. Wang, F. Faure, and D. K. Pai, "Adaptive image-based intersection volume," *ACM Trans. Graph.*, vol. 31, no. 4, p. 97, July 2012. [Online]. Available: https://doi.org/10.1145/2185520.2185593

[54] M. Verschoor and A. C. Jalba, "Efficient and accurate collision response for elastically deformable models," *ACM Trans. Graph.*, vol. 38, no. 2, p. 17, March 2019. [Online]. Available: https://doi.org/10.1145/3209887

[55] O. Ding and C. Schroeder, "Penalty force for coupling materials with coulomb friction," *IEEE Trans.on Vis.ization and Comput. Graph.*, vol. 26, no. 7, pp. 2443–2455, 2020.

[56] I. Huněk, "On a penalty formulation for contact-impact problems," *Comput.s & structures*, vol. 48, no. 2, pp. 193–203, 1993.

[57] T. Belytschko and M. O. Neal, "Contact-impact by the pinball algorithm with penalty and lagrangian methods," *Int. J. for Numer. Methods in Eng.*, vol. 31, no. 3, pp. 547–572, 1991.

[58] E. Drumwright, "A fast and stable penalty method for rigid body simulation," *IEEE Trans.on Vis.ization and Comput. Graph.*, vol. 14, no. 1, pp. 231–240, 2008.

[59] L. Kavan, "Rigid body collision response," *Vectors*, vol. 1000, no. 2, pp. 31–42, 2003.

[60] C. O'Sullivan and J. Dingliana, "Real-time collision detection and response using sphere-trees," 1999. [Online]. Available: https://api.semanticscholar.org/CorpusID:8354819

[61] C. Lee and L. Fu, “Impulse-based dynamic simulation of articulated rigid bodies with aerodynamics,” in *Proc. of the IEEE Int. Conf. on Syst., Man and Cybern., Taipei, Taiwan, October 8-11, 2006*, ser. I3D ’95, 2006, pp. 4420–4427. [Online]. Available: https://doi.org/10.1109/ICSMC.2006.384830

[62] Y. Li and J. Barbič, “Immersion of self-intersecting solids and surfaces,” *ACM Trans. Graph.*, vol. 37, no. 4, p. 45, July 2018. [Online]. Available: https://doi.org/10.1145/3197517.3201327

[63] S. Fisher and M. C. Lin, “Deformed distance fields for simulation of non-penetrating flexible bodies,” in *Proc. of the Eurographic Workshop on Comput. Animat. and Simul.*, 2001, p. 99–111.

[64] ——, “Fast penetration depth estimation for elastic bodies using deformed distance fields,” in *IEEE/RSJ Int. Conf. on Intell. Robots and Syst., IROS 2001: Expanding the Societal Role of Robot. in the the Next Millennium, Maui, HI, USA, October 29 - November 3, 2001*, vol. 1, 2001, pp. 330–336. [Online]. Available: https://doi.org/10.1109/IROS.2001.973379

[65] Y. Teng, M. A. Otaduy, and T. Kim, “Simulating articulated subspace self-contact,” *ACM Trans. Graph.*, vol. 33, no. 4, p. 106, July 2014. [Online]. Available: https://doi.org/10.1145/2601097.2601181

[66] H. Chen, E. Diaz, and C. Yuksel, “Shortest path to boundary for self-intersecting meshes,” *ACM Trans. Graph.*, vol. 42, no. 4, p. 146, July 2023. [Online]. Available: https://doi.org/10.1145/3592136

[67] M. Li, D. M. Kaufman, and C. Jiang, “Codimensional incremental potential contact,” *ACM Trans. Graph.*, vol. 40, no. 4, p. 170, July 2021. [Online]. Available: https://doi.org/10.1145/3450626.3459767

[68] Z. Ferguson, M. Li, T. Schneider, F. Gil-Ureta, T. Langlois, C. Jiang, D. Zorin, D. M. Kaufman, and D. Panozzo, “Intersection-free rigid body dynamics,” *ACM Trans. Graph.*, vol. 40, no. 4, Jul. 2021. [Online]. Available: https://doi.org/10.1145/3450626.3459802

[69] L. Lan, D. M. Kaufman, M. Li, C. Jiang, and Y. Yang, “Affine body dynamics: Fast, stable and intersection-free simulation of stiff materials,” *ACM Trans. Graph.*, vol. 41, no. 4, p. 67, Jul 2022. [Online]. Available: https://doi.org/10.1145/3528223.3530064

[70] L. Lan, M. Li, C. Jiang, H. Wang, and Y. Yang, “Second-order stencil descent for interior-point hyperelasticity,” *ACM Trans. Graph.*, vol. 42, no. 4, p. 108, Jul 2023. [Online]. Available: https://doi.org/10.1145/3592104

[71] K. Huang, F. M. Chitalu, H. Lin, and T. Komura, “Gipc: Fast and stable gauss-newton optimization of ipc barrier energy,” *ACM Trans. Graph.*, vol. 43, no. 2, p. 23, Mar 2024. [Online]. Available: https://doi.org/10.1145/3643028

[72] L. Lan, Y. Yang, D. Kaufman, J. Yao, M. Li, and C. Jiang, “Medial IPC: Accelerated incremental potential contact with medial elastics,” *ACM Trans. Graph.*, vol. 40, no. 4, p. 158, Jul 2021. [Online]. Available: https://doi.org/10.1145/3450626.3459753

[73] J. Nocedal and S. J. Wright, *Numerical optimization*, 1999.

[74] Y. Yuan, "Recent advances in trust region algorithms," *Math. Program.*, vol. 151, no. 1, pp. 249–281, 2015. [Online]. Available: https://doi.org/10.1007/s10107-015-0893-2

[75] J. V. Burke, "A robust trust region method for constrained nonlinear programming problems," *SIAM J. on Optim.*, vol. 2, no. 2, pp. 325–347, 1992.

[76] J. J. Moré, "Recent developments in algorithms and software for trust region methods," *Math. Program. The State of the Art: Bonn 1982*, pp. 258–287, 1983.

[77] A. R. Conn, N. I. Gould, and P. L. Toint, "Global convergence of a class of trust region algorithms for optimization with simple bounds," *SIAM journal on numerical analysis*, vol. 25, no. 2, pp. 433–460, 1988.

[78] J. V. Burke, J. J. Moré, and G. Toraldo, "Convergence properties of trust region methods for linear and convex constraints," *Math. Program.*, vol. 47, no. 1, pp. 305–336, 1990.

[79] X. Zhang, M. Lee, and Y. J. Kim, "Interactive continuous collision detection for non-convex polyhedra," *The Vis. Comput.*, vol. 22, pp. 749–760, 2006.

[80] M. Tang, Y. J. Kim, and D. Manocha, "C^2a: Controlled conservative advancement for continuous collision detection of polygonal models," in *2009 IEEE Int. Conf. on Robot. and Automat., ICRA 2009, Kobe, Jpn., May 12-17, 2009*. IEEE, 2009, pp. 849–854. [Online]. Available: https://doi.org/10.1109/ROBOT.2009.5152234

[81] L. Wu, B. Wu, Y. Yang, and H. Wang, "A safe and fast repulsion method for gpu-based cloth self collisions," *ACM Trans. Graph. (SIGGRAPH)*, vol. 40, no. 1, p. 5, Dec 2020. [Online]. Available: https://doi.org/10.1145/3430025

[82] T. Wang, J. Chen, D. Li, X. Liu, H. Wang, and K. Zhou, "Fast gpu-based two-way continuous collision handling," *ACM Trans. Graph. (SIGGRAPH)*, vol. 42, no. 5, p. 167, Jul 2023. [Online]. Available: https://doi.org/10.1145/3604551

[83] X. Chen and S. McMains, "Polygon offsetting by computing winding numbers," in *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, vol. 4739, 2005, pp. 565–575.

[84] Q. Bo, "Recursive polygon offset computing for rapid prototyping applications based on voronoi diagrams," *The International Journal of Advanced Manufacturing Technology*, vol. 49, pp. 1019–1028, 2010.

[85] O. Aichholzer, F. Aurenhammer, D. Alberts, and B. Gärtner, *A novel type of skeleton for polygons*. Springer, 1996.

[86] S. Huber, "The topology of skeletons and offsets," in *Proc. 34th Europ. Workshop on Comp. Geom.(EuroCG'18)*, 2018.

[87] G. Barequet and A. Goryachev, "Offset polygon and annulus placement problems," *Computational Geometry*, vol. 47, no. 3, Part A, pp. 407–434, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925772113001168

[88] T. Banchoff, "Critical points and curvature for embedded polyhedra," *Journal of Differential Geometry*, vol. 1, no. 3-4, pp. 245–256, 1967.

[89] T. F. Banchoff, "Critical points and curvature for embedded polyhedral surfaces," *The American Mathematical Monthly*, vol. 77, no. 5, pp. 475–485, 1970.

[90] U. Brehm and W. Kühnel, "Smooth approximation of polyhedral surfaces regarding curvatures," *Geometriae Dedicata*, vol. 12, no. 4, pp. 435–461, 1982.

[91] B. K. P. Horn, "Extended gaussian images," *Proceedings of the IEEE*, vol. 72, no. 12, pp. 1671–1686, 1984.

[92] J. J. Little, "Extended gaussian images, mixed volumes, shape reconstruction," in *Proceedings of the first annual symposium on Computational geometry*, 1985, pp. 15–23.

[93] J. Cohen, M. Olano, and D. Manocha, "Appearance-preserving simplification," in *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, 1998, pp. 115–122.

[94] G. Echeverria, "The polyhedral gauss map and discrete curvature measures in geometric modelling," Ph.D. dissertation, Sheffield Hallam University, UK, 2007. [Online]. Available: https://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.440302

[95] A. Menache, *Understanding motion capture for Comput. Animat. and video games*, 2000.

[96] S. I. Park and J. K. Hodgins, "Capturing and animating skin deformation in human motion," *ACM Trans. Graph.*, vol. 25, no. 3, p. 881–889, July 2006. [Online]. Available: https://doi.org/10.1145/1141911.1141970

[97] S. I. Park and J. Hodgins, "Data-driven modeling of skin and muscle deformation," *ACM Trans. Graph.*, vol. 27, no. 3, p. 1–6, August 2008. [Online]. Available: https://doi.org/10.1145/1360612.1360695

[98] M.-H. Song and R. I. Godøy, "How fast is your body motion? Determining a sufficient frame rate for an optical motion tracking system using passive markers," *PloS one*, vol. 11, no. 3, pp. 12–26, 2016.

[99] A. Aristidou, D. Cohen-Or, J. K. Hodgins, and A. Shamir, "Self-similarity analysis for motion capture cleaning," in *Comput. Graph. Forum*, vol. 37, no. 2. Wiley Online Library, 2018, pp. 297–309. [Online]. Available: https://doi.org/10.1111/cgf.13362

[100] D. Holden, "Robust solving of optical motion capture data by denoising," *ACM Trans. Graph.*, vol. 37, no. 4, p. 165, July 2018. [Online]. Available: https://doi.org/10.1145/3197517.3201302

[101] S. Han, B. Liu, R. Wang, Y. Ye, C. D. Twigg, and K. Kin, "Online optical marker-based hand tracking with deep labels," *ACM Trans. Graph.*, vol. 37, no. 4, p. 166, July 2018. [Online]. Available: https://doi.org/10.1145/3197517.3201399

[102] R. White, K. Crane, and D. A. Forsyth, "Capturing and animating occluded cloth," *ACM Trans. Graph.*, vol. 26, no. 3, pp. 34–es, 2007.

[103] V. Scholz, T. Stich, M. Keckeisen, M. Wacker, and M. A. Magnor, "Garment motion capture using color-coded patterns," in *ACM SIGGRAPH 2005 Sketches*, ser. SIGGRAPH '05, vol. 24, no. 3, 2005, pp. 439–447. [Online]. Available: https://doi.org/10.1111/j.1467-8659.2005.00869.x

[104] R. Y. Wang and J. Popović, "Real-time hand-tracking with a color glove," *ACM Trans. Graph.*, vol. 28, no. 3, p. 63, July 2009. [Online]. Available: https://doi.org/10.1145/1531326.1531369

[105] D. Gavrila and L. S. Davis, "3-d model-based tracking of humans in action: a multi-view approach," in *1996 Conf. on Comput. Vis. and Pattern Recognit. (CVPR '96), June 18-20, 1996 San Francisco, CA, USA*. (Palm Springs), 1996, pp. 73–80. [Online]. Available: https://doi.org/10.1109/CVPR.1996.517056

[106] C. Bregler, J. Malik, and K. Pullen, "Twist based acquisition and tracking of animal and human kinematics," *Int. J. of Comput. Vis.*, vol. 56, no. 3, pp. 179–194, 2004.

[107] R. Kehl and L. Van Gool, "Markerless tracking of complex human motions from multiple views," *Comput. Vis. and Image Understanding*, vol. 104, no. 2-3, pp. 190–209, 2006.

[108] T. Brox, B. Rosenhahn, J. Gall, and D. Cremers, "Combined region and motion-based 3d tracking of rigid and articulated objects," *IEEE Trans.on pattern analysis and machine intelligence*, vol. 32, no. 3, pp. 402–415, 2009.

[109] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proc. of the Int. Conf. on Comput. Vis., Kerkyra, Corfu, Greece, September 20-25, 1999*, vol. 2, 1999, pp. 1150–1157. [Online]. Available: https://doi.org/10.1109/ICCV.1999.790410

[110] M. Chen, C. Wang, and L. Liu, "Deep video-based performance synthesis from sparse multi-view capture," vol. 38, no. 7, August 2019, pp. 543–554. [Online]. Available: https://doi.org/10.1111/cgf.13859

[111] D. Vlasic, I. Baran, W. Matusik, and J. Popovic, "Articulated mesh animation from multi-view silhouettes," in *ACM Trans. Graph.*, vol. 27, no. 3. ACM, 2008, p. 97. [Online]. Available: https://doi.org/10.1145/1360612.1360696

[112] Y. Liu, J. Gall, C. Stoll, Q. Dai, H.-P. Seidel, and C. Theobalt, "Markerless motion capture of multiple characters using multiview image segmentation," *IEEE Trans.on Pattern Anal. and Mach. Intell.*, vol. 35, no. 11, pp. 2720–2735, 2013.

[113] P. Sand, L. McMillan, and J. Popović, "Continuous capture of skin deformation," *ACM Trans. Graph.*, vol. 22, no. 3, p. 578–586, July 2003. [Online]. Available: https://doi.org/10.1145/882262.882310

[114] J. Starck and A. Hilton, "Surface capture for performance-based animat." *IEEE Comput. Graph. and Appl.*, vol. 27, no. 3, pp. 21–31, 2007.

[115] S. Corazza, L. Mündermann, E. Gambaretto, G. Ferrigno, and T. P. Andriacchi, "Markerless motion capture through visual hull, articulated icp and subject specific model generation," *Int. J. of Comput. Vis.*, vol. 87, no. 1-2, pp. 156–169, 2010.

[116] J. Gall, B. Rosenhahn, T. Brox, and H.-P. Seidel, “Optimization and filtering for human motion capture,” *Int. J. of Comput. Vis.*, vol. 87, no. 1-2, pp. 75–76, 2010.

[117] C. Stoll, N. Hasler, J. Gall, H. Seidel, and C. Theobalt, “Fast articulated motion tracking using a sums of gaussians body model,” in *IEEE Int. Conf. on Comput. Vis., ICCV 2011, Barcelona, Spain, November 6-13, 2011*. IEEE, 2011, pp. 951–958. [Online]. Available: https://doi.org/10.1109/ICCV.2011.6126338

[118] G. Pavlakos, X. Zhou, K. G. Derpanis, and K. Daniilidis, “Harvesting multiple views for marker-less 3d human pose annotations,” in *2017 IEEE Conf. on Comput. Vis. and Pattern Recognit., CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, 2017, pp. 1253–1262. [Online]. Available: https://doi.org/10.1109/CVPR.2017.138

[119] T. Xu and W. Takano, “Graph stacked hourglass networks for 3d human pose estimation,” in *IEEE Conf. on Comput. Vis. and Pattern Recognit., CVPR 2021, virtual, June 19-25, 2021*, 2021, pp. 16 105–16 114. [Online]. Available: https://openaccess.thecvf.com/content/CVPR2021/html/Xu_Graph_Stacked_Hourglass_Networks_for_3D_Human_Pose_Estimation_CVPR_2021_paper.html

[120] L. Pishchulin, E. Insafutdinov, S. Tang, B. Andres, M. Andriluka, P. V. Gehler, and B. Schiele, “Deepcut: Joint subset partition and labeling for multi person pose estimation,” in *2016 IEEE Conf. on Comput. Vis. and Pattern Recognit., CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, 2016, pp. 4929–4937. [Online]. Available: https://doi.org/10.1109/CVPR.2016.533

[121] Y. Raaj, H. Idrees, G. Hidalgo, and Y. Sheikh, “Efficient online multi-person 2d pose tracking with recurrent spatio-temporal affinity fields,” in *IEEE Conf. on Comput. Vis. and Pattern Recognit., CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, 2019, pp. 4620–4628. [Online]. Available: http://openaccess.thecvf.com/content_CVPR_2019/html/Raaj_Efficient_Online_Multi-Person_2D_Pose_Tracking_With_Recurrent_Spatio-Temporal_Affinity_CVPR_2019_paper.html

[122] B. Qiang, S. Zhang, Y. Zhan, W. Xie, and T. Zhao, “Improved convolutional pose machines for human pose estimation using image sensor data,” in *2016 IEEE Conf. on Comput. Vis. and Pattern Recognit. (CVPR)*, vol. 19, no. 3, 2019, p. 718. [Online]. Available: https://doi.org/10.3390/s19030718

[123] Z. Cao, G. Hidalgo, T. Simon, S. Wei, and Y. Sheikh, “Openpose: Realtime multi-person 2d pose estimation using part affinity fields,” in *Proc. of the IEEE Conf. on Comput. Vis. and Pattern Recognit.*, vol. 43, no. 1, 2021, pp. 172–186. [Online]. Available: https://doi.org/10.1109/TPAMI.2019.2929257

[124] ——, “Openpose: Realtime multi-person 2d pose estimation using part affinity fields,” in *2017 IEEE Conf. on Comput. Vis. and Pattern Recognit. (CVPR)*, vol. 43, no. 1, July 2021, pp. 172–186. [Online]. Available: https://doi.org/10.1109/TPAMI.2019.2929257

[125] G. H. Martinez, Y. Raaj, H. Idrees, D. Xiang, H. Joo, T. Simon, and Y. Sheikh, “Single-network whole-body pose estimation,” in *2019 IEEE/CVF Int. Conf. on Comput. Vis., ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, 2019, pp. 6981–6990. [Online]. Available: https://doi.org/10.1109/ICCV.2019.00708

[126] D. Mehta et al., “Vnect: Real-time 3d human pose estimation with a single rgb camera,” *ACM Trans. Graph.*, vol. 36, no. 4, p. 44, 2017.

[127] G. Pavlakos, V. Choutas, N. Ghorbani, T. Bolkart, A. A. A. Osman, D. Tzionas, and M. J. Black, “Expressive body capture: 3d hands, face, and body from a single image,” in *IEEE Conf. on Comput. Vis. and Pattern Recognit., CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, June 2019, pp. 10 975–10 985. [Online]. Available: http://openaccess.thecvf.com/content_CVPR_2019/html/Pavlakos_Expressive_Body_Capture_3D_Hands_Face_and_Body_From_a_CVPR_2019_paper.html

[128] D. Xiang, H. Joo, and Y. Sheikh, “Monocular total capture: Posing face, body, and hands in the wild,” in *IEEE Conf. on Comput. Vis. and Pattern Recognit., CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, June 2019, pp. 10 965–10 974. [Online]. Available: http://openaccess.thecvf.com/content_CVPR_2019/html/Xiang_Monocular_Total_Capture_Posing_Face_Body_and_Hands_in_the_CVPR_2019_paper.html

[129] R. A. Güler, N. Neverova, and I. Kokkinos, “Densepose: Dense human pose estimation in the wild,” in *2018 IEEE Conf. on Comput. Vis. and Pattern Recognit., CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, June 2018, pp. 7297–7306. [Online]. Available: http://openaccess.thecvf.com/content_cvpr_2018/html/Guler_DensePose_Dense_Human_CVPR_2018_paper.html

[130] Y. Xu, S. Zhu, and T. Tung, “Denserac: Joint 3d pose and shape estimation by dense render-and-compare,” in *2019 IEEE/CVF Int. Conf. on Comput. Vis., ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, 2019, pp. 7759–7769. [Online]. Available: https://doi.org/10.1109/ICCV.2019.00785

[131] G. Pons-Moll, J. Romero, N. Mahmood, and M. J. Black, “Dyna: A model of dynamic human shape in motion,” *ACM Trans. Graph.*, vol. 34, no. 4, p. 120, 2015.

[132] K. M. Robinette, S. Blackwell, H. Daanen, M. Boehmer, and S. Fleming, “Civilian american and european surface anthropometry resource (caesar), final report. volume 1. summary,” SYTRONICS INC DAYTON OH, Tech. Rep., 2002.

[133] D. Anguelov, P. Srinivasan, D. Koller, S. Thrun, J. Rodgers, and J. Davis, “SCAPE: shape completion and animation of people,” in *ACM Trans. Graph.*, vol. 24, no. 3. ACM, 2005, pp. 408–416. [Online]. Available: https://doi.org/10.1145/1073204.1073207

[134] M. Loper, N. Mahmood, J. Romero, G. Pons-Moll, and M. J. Black, “Smpl: a skinned multi-person linear model,” *ACM Trans. Graph.*, vol. 34, no. 6, p. 248, October 2015. [Online]. Available: https://doi.org/10.1145/2816795.2818013

[135] A. A. A. Osman, T. Bolkart, and M. J. Black, “STAR: sparse trained articulated human body regressor,” in *Comput. Vis. - ECCV 2020 - 16th Eur. Conf., Glasgow, UK, August 23-28, 2020, Proc., Part VI*, ser. Lecture Notes in Computer Science, vol. 12351, 2020, pp. 598–613. [Online]. Available: https://doi.org/10.1007/978-3-030-58539-6_36

[136] H. Joo, T. Simon, and Y. Sheikh, “Total capture: A 3d deformation model for tracking faces, hands, and bodies,” in *2018 IEEE Conf. on Comput. Vis. and Pattern Recognit., CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, June 2018, pp.

8320–8329. [Online]. Available: http://openaccess.thecvf.com/content_cvpr_2018/html/Joo_Total_Capture_A_CVPR_2018_paper.html

[137] G. Pavlakos, V. Choutas, N. Ghorbani, T. Bolkart, A. A. A. Osman, D. Tzionas, and M. J. Black, "Expressive body capture: 3d hands, face, and body from a single image," in *IEEE Conf. on Comput. Vis. and Pattern Recognit., CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, 2019, pp. 10 975–10 985. [Online]. Available: http://openaccess.thecvf.com/content_CVPR_2019/html/Pavlakos_Expressive_Body_Capture_3D_Hands_Face_and_Body_From_a_CVPR_2019_paper.html

[138] K. Guo et al., "The relightables: Volumetric performance capture of humans with realistic relighting," *ACM Trans. Graph.*, vol. 38, no. 6, pp. 1–19, 2019.

[139] A. Meka, R. Pandey, C. Häne, S. Orts-Escolano, P. Barnum, P. Davidson, D. Erickson, Y. Zhang, J. Taylor, S. Bouaziz, C. LeGendre, W. Ma, R. S. Overbeck, T. Beeler, P. E. Debevec, S. Izadi, C. Theobalt, C. Rhemann, and S. R. Fanello, "Deep relightable textures: volumetric performance capture with neural rendering," vol. 39, no. 6, Dec 2020, pp. 259:1–259:21. [Online]. Available: https://doi.org/10.1145/3414685.3417814

[140] M. Fiala, "Artag, a fiducial marker system using digital techniques," in *2005 IEEE Comput. Soc. Conf. on Comput. Vis. and Pattern Recognit. (CVPR 2005), 20-26 June 2005, San Diego, CA, USA*, vol. 2. IEEE, 2005, pp. 590–596. [Online]. Available: https://doi.org/10.1109/CVPR.2005.74

[141] E. Olson, "Apriltag: A robust and flexible visual fiducial system," in *IEEE Int. Conf. on Robot. and Automat., ICRA 2011, Shanghai, China, 9-13 May 2011*. IEEE, 2011, pp. 3400–3407. [Online]. Available: https://doi.org/10.1109/ICRA.2011.5979561

[142] J. Wang and E. Olson, "Apriltag 2: Efficient and robust fiducial detection," in *2016 IEEE/RSJ Int. Conf. on Intell. Robots and Syst., IROS 2016, Daejeon, South Korea, October 9-14, 2016*. IEEE, 2016, pp. 4193–4198. [Online]. Available: https://doi.org/10.1109/IROS.2016.7759617

[143] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and M. J. Marín-Jiménez, "Automatic generation and detection of highly reliable fiducial markers under occlusion," *Pattern Recognit.*, vol. 47, no. 6, pp. 2280–2292, 2014.

[144] D. DeTone, T. Malisiewicz, and A. Rabinovich, "Superpoint: Self-supervised interest point detection and description," in *2018 IEEE Conf. on Comput. Vis. and Pattern Recognit. Workshops, CVPR Workshops 2018, Salt Lake City, UT, USA, June 18-22, 2018*, June 2018, pp. 224–236. [Online]. Available: http://openaccess.thecvf.com/content_cvpr_2018_workshops/w9/html/DeTone_SuperPoint_Self-Supervised_Interest_CVPR_2018_paper.html

[145] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *Comput. Vis. - ECCV 2006, 9th Eur. Conf. on Comput. Vis., Graz, Austria, May 7-13, 2006, Proc., Part I*, ser. Lecture Notes in Computer Science, vol. 3951. Springer, 2006, pp. 430–443. [Online]. Available: https://doi.org/10.1007/11744023_34

[146] S. Bennett and J. Lasenby, "Chess–quick and robust detection of chess-board features," *Comput. Vis. and Image Understanding*, vol. 118, pp. 197–210, 2014.

[147] D. Hu, D. DeTone, and T. Malisiewicz, "Deep charuco: Dark charuco marker pose estimation," in *IEEE Conf. on Comput. Vis. and Pattern Recognit., CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, 2019, pp. 8436–8444. [Online]. Available: http://openaccess.thecvf.com/content_CVPR_2019/html/Hu_Deep_ChArUco_Dark_ChArUco_Marker_Pose_Estimation_CVPR_2019_paper.html

[148] B. Chen, C. Xiong, and Q. Zhang, "CCDN: checkerboard corner detection network for robust camera calibration," in *Int. Conf. on Intell. Robot. and Appl.*, vol. abs/2302.05097. Springer, 2023, pp. 324–334. [Online]. Available: https://doi.org/10.48550/arXiv.2302.05097

[149] S. Donné, J. De Vylder, B. Goossens, and W. Philips, "Mate: Machine learning for adaptive calibration template detection," *Sensors*, vol. 16, no. 11, p. 1858, 2016.

[150] S. Long, X. He, and C. Yao, "Scene text detection and recognition: The deep learning era," *Int. J. of Comput. Vis.*, pp. 1–24, 2020.

[151] J. Ma et al., "Arbitrary-oriented scene text detection via rotation proposals," *IEEE Trans.on Multimedia*, vol. 20, no. 11, pp. 3111–3122, 2018.

[152] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Deep features for text spotting," in *Comput. Vis. - ECCV 2014 - 13th Eur. Conf., Zurich, Switzerland, September 6-12, 2014, Proc., Part IV*, ser. Lecture Notes in Computer Science, vol. 8692. Springer, 2014, pp. 512–528. [Online]. Available: https://doi.org/10.1007/978-3-319-10593-2_34

[153] B. Allain, J. Franco, and E. Boyer, "An efficient volumetric framework for shape tracking," in *IEEE Conf. on Comput. Vis. and Pattern Recognit., CVPR 2015, Boston, MA, USA, June 7-12, 2015*, 2015, pp. 268–276. [Online]. Available: https://doi.org/10.1109/CVPR.2015.7298623

[154] H. Li, B. Adams, L. J. Guibas, and M. Pauly, "Robust single-view geometry and motion reconstruction," *ACM Trans. Graph.*, vol. 28, no. 5, pp. 1–10, 2009.

[155] F. Bogo, M. J. Black, M. Loper, and J. Romero, "Detailed full-body reconstructions of moving people from monocular RGB-D sequences," in *2015 IEEE Int. Conf. on Comput. Vis., ICCV 2015, Santiago, Chile, December 7-13, 2015*, 2015, pp. 2300–2308. [Online]. Available: https://doi.org/10.1109/ICCV.2015.265

[156] M. Dou, J. Taylor, H. Fuchs, A. W. Fitzgibbon, and S. Izadi, "3d scanning deformable objects with a single RGBD sensor," in *IEEE Conf. on Comput. Vis. and Pattern Recognit., CVPR 2015, Boston, MA, USA, June 7-12, 2015*, 2015, pp. 493–501. [Online]. Available: https://doi.org/10.1109/CVPR.2015.7298647

[157] R. A. Newcombe, D. Fox, and S. M. Seitz, "Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time," in *IEEE Conf. on Comput. Vis. and Pattern Recognit., CVPR 2015, Boston, MA, USA, June 7-12, 2015*, 2015, pp. 343–352. [Online]. Available: https://doi.org/10.1109/CVPR.2015.7298631

[158] A. Collet et al., "High-quality streamable free-viewpoint video," *ACM Trans. Graph.*, vol. 34, no. 4, pp. 1–13, 2015.

[159] D. Casas, M. Tejera, J. Guillemaut, and A. Hilton, "4d parametric motion graphs for interactive animation," in *Symp. on Interactive 3D Graph. and Games, I3D '12, Costa Mesa, CA, USA, March 09 - 11, 2012*, 2012, pp. 103–110. [Online]. Available: https://doi.org/10.1145/2159616.2159633

[160] P. Huang, C. Budd, and A. Hilton, "Global temporal registration of multiple non-rigid surface sequences," in *The 24th IEEE Conf. on Comput. Vis. and Pattern Recognit., CVPR 2011, Colorado Springs, CO, USA, 20-25 June 2011*. IEEE, 2011, pp. 3473–3480. [Online]. Available: https://doi.org/10.1109/CVPR.2011.5995438

[161] T. Tung and T. Matsuyama, "Dynamic surface matching by geodesic mapping for 3d animation transfer," in *The Twenty-3rd IEEE Conf. on Comput. Vis. and Pattern Recognit., CVPR 2010, San Francisco, CA, USA, 13-18 June 2010*. IEEE, 2010, pp. 1402–1409. [Online]. Available: https://doi.org/10.1109/CVPR.2010.5539806

[162] A. Boukhayma, V. Tsiminaki, J. Franco, and E. Boyer, "Eigen appearance maps of dynamic shapes," in *Comput. Vis. - ECCV 2016 - 14th Eur. Conf., Amsterdam, The Netherlands, October 11-14, 2016, Proc., Part I*, ser. Lecture Notes in Computer Science, vol. 9905. Springer, 2016, pp. 230–245. [Online]. Available: https://doi.org/10.1007/978-3-319-46448-0_14

[163] F. Prada, M. Kazhdan, M. Chuang, A. Collet, and H. Hoppe, "Motion graphs for unstructured textured meshes," *ACM Trans. Graph.*, vol. 35, no. 4, pp. 1–14, 2016.

[164] B. Allen, B. Curless, and Z. Popovic, "The space of human body shapes: reconstruction and parameterization from range scans," in *ACM Trans. Graph.*, vol. 22, no. 3. ACM, 2003, pp. 587–594. [Online]. Available: https://doi.org/10.1145/882262.882311

[165] D. A. Hirshberg, M. Loper, E. Rachlin, and M. J. Black, "Coregistration: Simultaneous alignment and modeling of articulated 3d shape," in *Comput. Vis. - ECCV 2012 - 12th Eur. Conf. on Comput. Vis., Florence, Italy, October 7-13, 2012, Proc., Part VI*, ser. Lecture Notes in Computer Science, vol. 7577. Springer, 2012, pp. 242–255. [Online]. Available: https://doi.org/10.1007/978-3-642-33783-3_18

[166] F. Bogo, J. Romero, M. Loper, and M. J. Black, "FAUST: dataset and evaluation for 3d mesh registration," in *2014 IEEE Conf. on Comput. Vis. and Pattern Recognit., CVPR 2014, Columbus, OH, USA, June 23-28, 2014*, June 2014, pp. 3794–3801. [Online]. Available: https://doi.org/10.1109/CVPR.2014.491

[167] F. Bogo, J. Romero, G. Pons-Moll, and M. J. Black, "Dynamic FAUST: registering human bodies in motion," in *2017 IEEE Conf. on Comput. Vis. and Pattern Recognit., CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, July 2017, pp. 5573–5582. [Online]. Available: https://doi.org/10.1109/CVPR.2017.591

[168] E. Sifakis and J. Barbic, "Fem simulation of 3d deformable solids: A practitioner's guide to theory, discretization and model reduction," *ACM SIGGRAPH 2012 Courses, SIGGRAPH'12*, 08 2012.

[169] H. Chen, E. Diaz, and C. Yuksel, "Shortest path to boundary for self-intersecting meshes," vol. 42, no. 4, p. 146, 2023. [Online]. Available: https://doi.org/10.1145/3592136

[170] G. H. Golub and C. F. Van Loan, *Matrix computations*, 2013.

[171] J. A. Levine, A. W. Bargteil, C. Corsi, J. Tessendorf, and R. Geist, "A peridynamic perspective on spring-mass fracture," in *The Eurographics / ACM SIGGRAPH Symp. on Comput. Animation, SCA 2014, Copenhagen, Denmark, 2014*, ser. SCA '14, 2014, pp. 47–55. [Online]. Available: https://doi.org/10.2312/sca.20141122

[172] M. Müller, D. Charypar, and M. H. Gross, "Particle-based fluid simulation for interactive applications," in *Proc. of the 2003 ACM SIGGRAPH/Eurographics Symp. on Comput. Animation, San Diego, CA, USA, July 26-27, 2003*, ser. SCA '03, 2003, pp. 154–159. [Online]. Available: https://doi.org/10.2312/SCA03/154-159

[173] T. Takahashi, Y. Dobashi, I. Fujishiro, T. Nishita, and M. C. Lin, "Implicit formulation for sph-based viscous fluids," *Comput. Graph. Forum*, vol. 34, no. 2, p. 493–502, may 2015. [Online]. Available: https://doi.org/10.1111/cgf.12578

[174] A. Peer, M. Ihmsen, J. Cornelis, and M. Teschner, "An implicit viscosity formulation for SPH fluids," *ACM Trans. Graph.*, vol. 34, no. 4, p. 114, Jul 2015. [Online]. Available: https://doi.org/10.1145/2766925

[175] B. V. Mirtich, "Impulse-based dynamic simulation of rigid body systems," Ph.D. dissertation, University of California, Berkeley, 1996, aAI9723116.

[176] M. Müller, R. Keiser, A. Nealen, M. Pauly, M. H. Gross, and M. Alexa, "Point based animation of elastic, plastic and melting objects," in *Proc. of the 2004 ACM SIGGRAPH/Eurographics Symp. on Comput. Animation, Grenoble, France, August 27-29, 2004*, ser. SCA '04, 2004, pp. 141–151. [Online]. Available: https://doi.org/10.2312/SCA/SCA04/141-151

[177] B. Solenthaler, J. Schläfli, and R. Pajarola, "A unified particle model for fluid–solid interactions: Research articles," *Comput. Animat. Virtual Worlds*, vol. 18, no. 1, p. 69–82, Feb 2007.

[178] M. Becker, M. Ihmsen, and M. Teschner, "Corotated SPH for deformable solids," in *Proc. of the Eurographics Workshop on Natural Phenomena, NPH 2009, Munich, Germany, 2009*, ser. NPH'09, 2009, pp. 27–34. [Online]. Available: https://doi.org/10.2312/EG/DL/conf/EG2009/nph/027-034

[179] S. Martin, P. Kaufmann, M. Botsch, E. Grinspun, and M. Gross, "Unified simulation of elastic rods, shells, and solids," *ACM Trans. Graph.*, vol. 29, no. 4, p. 39, Jul 2010. [Online]. Available: https://doi.org/10.1145/1778765.1778776

[180] M. Macklin, M. Müller, N. Chentanez, and T.-Y. Kim, "Unified particle physics for real-time applications," *ACM Trans. Graph.*, vol. 33, no. 4, p. 153, Jul 2014. [Online]. Available: https://doi.org/10.1145/2601097.2601152

[181] B. Smith, F. D. Goes, and T. Kim, "Stable neo-hookean flesh simulation," *ACM Trans. Graph.*, vol. 37, no. 2, p. 12, March 2018. [Online]. Available: https://doi.org/10.1145/3180491

[182] P. Volino, N. Magnenat-Thalmann, and F. Faure, "A simple approach to nonlinear tensile stiffness for accurate cloth simulation," *ACM Trans. Graph.*, vol. 28, no. 4, p. 105, Sep 2009. [Online]. Available: https://doi.org/10.1145/1559755.1559762

[183] I. Wald, S. Woop, C. Benthin, G. S. Johnson, and M. Ernst, "Embree: a kernel framework for efficient cpu ray tracing," *ACM Trans. Graph.*, vol. 33, no. 4, p. 143, July 2014. [Online]. Available: https://doi.org/10.1145/2601097.2601199

[184] E. Wang et al., "Intel math kernel library," *High-Perform. Comput. on the Intel® Xeon Phi™: How to Fully Exploit MIC Archit.s*, pp. 167–188, 2014.

[185] L. A. Hageman and T. A. Porsching, "Aspects of nonlinear block successive overrelaxation," *SIAM J. Numer. Anal.*, vol. 12, no. 3, p. 316–335, June 1975. [Online]. Available: https://doi.org/10.1137/0712026

[186] L. Grippo and M. Sciandrone, "On the convergence of the block nonlinear gauss-seidel method under convex constraints," *Oper. Res. Lett.*, vol. 26, no. 3, p. 127–136, April 2000. [Online]. Available: https://doi.org/10.1016/S0167-6377(99)00074-7

[187] A. Aman, S. Demirci, and U. Güdükbay, "Compact tetrahedralization-based acceleration structures for ray tracing," *J. of Vis.ization*, vol. 25, no. 5, pp. 1103–1115, 2022.

[188] Q. Zhou and A. Jacobson, "Thingi10k: A dataset of 10,000 3d-printing models," *arXiv preprint arXiv:1605.04797*, 2016.

[189] J. Bender, M. Müller, M. A. Otaduy, M. Teschner, and M. Macklin, "A survey on position-based simulation methods in computer graphics," in *EG 2015 - Tut.*, vol. 33, no. 6, 2014, pp. 228–251. [Online]. Available: https://doi.org/10.1111/cgf.12346

[190] M. Macklin and M. Müller, "A constraint-based formulation of stable neo-hookean materials," in *MIG '21: Motion, Interaction and Games, Virtual Event, Switzerland, November 10-12, 2021*, ser. MIG '21, 2021, pp. 12:1–12:7. [Online]. Available: https://doi.org/10.1145/3487983.3488289

[191] M. Macklin, "Warp: A high-performance python framework for gpu simulation and graphics," https://github.com/nvidia/warp, March 2022, nVIDIA GPU Technology Conference (GTC).

[192] K. Huang, F. Chitalu, H. Lin, and T. Komura, "GIPC: Fast and stable Gauss-Newton optimization of IPC barrier energy," January 2024, arXiv:2308.09400 [cs] version: 4. [Online]. Available: http://arxiv.org/abs/2308.09400

[193] H. Zhou and H. Hu, "Human motion tracking for rehabilitation—a survey," *Biomed. Signal Process. and Control*, vol. 3, no. 1, pp. 1–18, 2008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1746809407000778

[194] S. Giovanni, Y. C. Choi, J. Huang, K. E. Tat, and K. Yin, "Virtual try-on using kinect and HD camera," in *Motion in Games - 5th Int. Conf., MIG 2012, Rennes, France, November 15-17, 2012. Proc.*, ser. Lecture Notes in Computer Science, vol. 7660. Springer, 2012, pp. 55–65. [Online]. Available: https://doi.org/10.1007/978-3-642-34710-8_6

[195] Q. Ma, J. Yang, A. Ranjan, S. Pujades, G. Pons-Moll, S. Tang, and M. J. Black, "Learning to dress 3d people in generative clothing," in *2020 IEEE/CVF Conf. on Comput. Vis. and Pattern Recognit., CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, June 2020, pp. 6468–6477.

[Online]. Available: https://openaccess.thecvf.com/content_CVPR_2020/html/Ma_Learning_to_Dress_3D_People_in_Generative_Clothing_CVPR_2020_paper.html

[196] A. Barmpoutis, "Tensor body: Real-time reconstruction of the human body and avatar synthesis from rgb-d," *IEEE Trans.on Cybern.*, vol. 43, no. 5, pp. 1347–1356, 2013.

[197] S. Lombardi, J. Saragih, T. Simon, and Y. Sheikh, "Deep appearance models for face rendering," *ACM Trans. Graph.*, vol. 37, no. 4, p. 68, July 2018. [Online]. Available: https://doi.org/10.1145/3197517.3201401

[198] Z. Zhang, " A Flexible New Technique for Camera Calibration ," *IEEE Trans.on Pattern Anal. & Mach. Intell.*, vol. 22, no. 11, pp. 1330–1334, November 2000. [Online]. Available: https://doi.ieeeComput.society.org/10.1109/34.888718

[199] "The OpenCV Library," *Dr. Dobb's J. of Softw. Tools*, 2000.

[200] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, "Bundle adjustment - A modern synthesis," in *Vis. Algorithms: Theory and Pract., Int. Workshop on Vis. Algorithms, held during ICCV '99, Corfu, Greece, September 21-22, 1999, Proc.*, ser. Lecture Notes in Computer Science, vol. 1883. Springer, 1999, pp. 298–372. [Online]. Available: https://doi.org/10.1007/3-540-44480-7_21

[201] S. Agarwal and K. Mierle, "Ceres solver: Tutorial &amp; reference," *Google Inc*, vol. 2, p. 72, 2012.

[202] A. Droschinsky, N. M. Kriege, and P. Mutzel, "Faster algorithms for the maximum common subtree isomorphism problem," in *41st Int. Symp. on Math. Found.s of Comput. Sci., MFCS 2016, August 22-26, 2016 - Kraków, Poland*, ser. LIPIcs, vol. 58, June 2016, pp. 33:1–33:14. [Online]. Available: https://doi.org/10.4230/LIPIcs.MFCS.2016.33

[203] C. Yan, H. Zhang, X. Li, and D. Yuan, "R-SSD: refined single shot multibox detector for pedestrian detection," in *Comput. Vis. – ECCV 2016*, vol. 52, no. 9, 2022, pp. 10 430–10 447. [Online]. Available: https://doi.org/10.1007/s10489-021-02798-1

[204] R. I. Hartley and P. Sturm, "Triangulation," *Comput. Vis. and Image Understanding*, vol. 68, no. 2, pp. 146–157, 1997.

[205] G. Upton and I. Cook, *Understanding statistics*, 1996.

[206] N. Magnenat-Thalmann, R. Laperrière, and D. Thalmann, "Joint-dependent local deformations for hand animation and object grasping," in *Proc. on Graph. Interface '88*, 1989, p. 26–33.

[207] M. Bergou, M. Wardetzky, D. Harmon, D. Zorin, and E. Grinspun, "A quadratic bending model for inextensible surfaces," in *Proc. of the 4th Eurographics Symp. on Geometry Process., Cagliari, Sardinia, Italy, June 26-28, 2006*, ser. ACM International Conference Proceeding Series, vol. 256, 2006, pp. 227–230. [Online]. Available: https://doi.org/10.2312/SGP/SGP06/227-230

[208] M. Abadi et al., "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, Retrived from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[209] J. Shi and C. Tomasi, "Good features to track," in *Conf. on Comput. Vis. and Pattern Recognit., CVPR 1994, 21-23 June, 1994, Seattle, WA, USA*, 1994, pp. 593–600. [Online]. Available: https://doi.org/10.1109/CVPR.1994.323794

[210] C. G. Harris and M. Stephens, "A combined corner and edge detector," in *Proc. of the Alvey Vis. Conf., AVC 1988, Manchester, UK, September, 1988*, vol. 15, no. 50. Citeseer, 1988, pp. 1–6. [Online]. Available: https://doi.org/10.5244/C.2.23

[211] G. Bradski and A. Kaehler, *Learning OpenCV: Comput. vision with the OpenCV library*, 2008.

[212] W. Förstner and E. Gülch, "A fast operator for detection and precise location of distinct points, corners and centres of circular features," in *Proc. ISPRS intercommission conference on fast processing of photogrammetric data*. Interlaken, 1987, pp. 281–305.

[213] R. Smith, "An overview of the tesseract OCR engine," in *9th Int. Conf. on Document Anal. and Recognit. (ICDAR 2007), 23-26 September, Curitiba, Paraná, Brazil*, vol. 2, September 2007, pp. 629–633. [Online]. Available: https://doi.org/10.1109/ICDAR.2007.4376991

[214] R. Nikhila et al., "Accelerating 3d deep learning with pytorch3d," *arXiv:2007.08501*, 2020.

[215] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, "Flownet 2.0: Evolution of optical flow estimation with deep networks," in *2017 IEEE Conf. on Comput. Vis. and Pattern Recognit., CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, July 2017, pp. 1647–1655. [Online]. Available: https://doi.org/10.1109/CVPR.2017.179

[216] É. Borel, "Sur quelques points de la théorie des fonctions," in *Annales scientifiques de l'École normale supérieure*, vol. 12, 1895, pp. 9–55.

[217] S. N. Wood, "Thin plate regression splines," *J. of the Roy. Statistical Soc.: Ser. B (Statistical Methodology)*, vol. 65, no. 1, pp. 95–114, 2003.

[218] N. Mahmood, N. Ghorbani, N. F. Troje, G. Pons-Moll, and M. J. Black, "AMASS: Archive of motion capture as surface shapes," in *Int. Conf. on Comput. Vis.*, October 2019, pp. 5442–5451.