ARTIST-GUIDED PHYSICS-BASED ANIMATION

by

Benjamin J. Jones

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

School of Computing

The University of Utah

August 2015

The University of Utah Graduate School

STATEMENT OF DISSERTATION APPROVAL

The dissertation of **Benjamin J. Jones**

has been approved by the following supervisory committee members:

Adam Wade Bargteil	, Chair	**3/20/15**
		Date Approved
Robert Michael Kirby II	, Member	**3/20/15**
		Date Approved
Cem Yuksel	, Member	**3/20/15**
		Date Approved
Ladislav Kavan	, Member	**4/14/15**
		Date Approved
Nils Thuerey	, Member	**4/7/15**
		Date Approved

and by **Ross T. Whitaker**, Chair/Dean of

the School of **Computing**

and by David B. Kieda, Dean of The Graduate School.

ABSTRACT

Physics-based animation has proven to be a powerful tool for creating compelling animations for film and games. Most techniques in graphics are based on methods developed for predictive simulation for engineering applications; however, the goals for graphics applications are dramatically different than the goals of engineering applications. As a result, most physics-based animation tools are difficult for artists to work with, providing little direct control over simulation results. In this thesis, we describe tools for physics-based animation designed with artist needs and expertise in mind.

Most materials can be modeled as elastoplastic: they recover from small deformations, but large deformations permanently alter their rest shape. Unfortunately, large plastic deformations, common in graphical applications, cause simulation instabilities if not addressed. Most elastoplastic simulation techniques in graphics rely on a finite-element approach where objects are discretized into a tetrahedral mesh. Using these approaches, maintaining simulation stability during large plastic flows requires remeshing, a complex and computationally expensive process. We introduce a new point-based approach that does not rely on an explicit mesh and avoids the expense of remeshing. Our approach produces comparable results with much lower implementation complexity. Points are a ubiquitous primitive for many effects, so our approach also integrates well with existing artist pipelines.

Next, we introduce a new technique for animating stylized images which we call *Dynamic Sprites*. Artists can use our tool to create digital assets that interact in a natural, but stylized, way in virtual environments. In order to support the types of nonphysical, exaggerated motions often desired by artists, our approach relies on a heavily modified deformable body simulator, equipped with a set of new intuitive controls and an example-based deformation model. Our approach allows artists to specify how the shape of the object should change as it moves and collides in interactive virtual environments.

Finally, we introduce a new technique for animating destructive scenes. Our approach is built on the insight that the most important visual aspects of destruction are plastic deformation and fracture. Like with *Dynamic Sprites*, we use an example-based model of deformation for intuitive artist control. Our simulator treats objects as rigid when

computing dynamics but allows them to deform plastically and fracture in between timesteps based on interactions with the other objects. We demonstrate that our approach can efficiently animate the types of destructive scenes common in film and games.

These animation techniques are designed to exploit artist expertise to ease creation of complex animations. By using artist-friendly primitives and allowing artists to provide characteristic deformations as input, our techniques enable artists to create more compelling animations, more easily.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

The work in this dissertation would not have been possible without the guidance of my advisor, Adam Bargteil. He helped me gain the technical expertise needed to perform graphics research, but more importantly, he taught me the soft skills I needed to communicate my ideas clearly and effectively with the research community.

I'm also very grateful to my collaborators: Stephen Ward, Ashok Jallepalli, Joseph Perenia, and Alex Stuart at Utah; Jovan Popovic, Jim McCann, and Wilmot Li from Adobe; Nils Thuerey from TU Munich; Tamar Shinar from UC Riverside; and Josh Levine from Clemson. The ideas, insights, guidance, and even code you provided made our projects successful.

Thank you to my committee, Adam Bargteil, Mike Kirby, Cem Yuksel, Ladislav Kavan, and Nils Thuerey, for keeping me on track and focused. You have all provided me with great advice and ideas that helped shape the work in my dissertation.

Finally, I'd like to thank my friends and family who kept my spirits up in the face of paper deadlines, negative reviews, and grad school-induced exhaustion.

CHAPTER 1

INTRODUCTION

Physics-based animation has become a ubiquitous tool for creating realistic depictions of natural (and sometimes unnatural) phenomena in film and games. These approaches solve the equations of motion, discovered by physicists and engineers for a variety of materials, to compute lifelike motion. Using computers to simulate physics is not a new idea: some of the earliest uses of computers were to compute bullet trajectories to create firing tables. Since then, engineers have used simulation to aid design and predict failures before building expensive physical prototypes. In the 1980s, graphics researchers began to use simulation as a tool for animation, adapting the tools and techniques developed by engineers and scientific computing researchers for a new application. This approach was remarkably successful as evidenced by the pervasive, high-quality effects in movies and video games today.

Though popular, physics-based animation is very challenging for artists to use effectively since it removes the direct control artists rely on for most other aspects of animation – modeling, key-framing, or manipulating animation curves. A typical workflow for artists using simulation techniques is a costly and time-consuming simulate, tweak, repeat cycle. Artists adjust initial conditions and material properties, such as density or stiffness, then rerun the simulation, hopefully achieving an animation closer to the desired result. Large-scale simulations are slow to run, and it is difficult to predict the result of changing simulation parameters since they are interconnected in complex, nonlinear ways.

Most research about physics-based animation is focused on simulating more phenomena, more accurately, and more efficiently. Many of these works focus on issues important to engineers and numericists, such as rate of convergence or discretization error. For graphics applications, however, our goal is to create compelling animations: simulation can be a useful tool, but the end goal is to aid animators, not to predict physical phenomena. In particular, artistic control is a more important design criterion for graphics applications than physical accuracy. Inspired by this philosophy, in this dissertation, we describe techniques for artist-guided physics-based simulation. Specifically we created new techniques for animating deformable solids that allow artists more direct control over simulation results

than existing approaches. Our methods can be used to animate elastoplastic materials, create stylized animated sprites, and animate deforming, near-rigid objects.

Our first contribution is a new approach for animating elastoplastic materials. Most materials can be modeled as elastoplastic, deforming elastically up to a limit, then undergoing permanent plastic deformation if internal stresses are too great. In order to compute elastic forces, we require a surjective map from the object's rest shape to its current configuration, i.e. no two distinct points in the current configuration can map to the same point in the rest configuration. Since plastic deformation changes the rest shape of the object, this mapping may become degenerate, causing simulation instability. In order to ensure stability, plastic deformation must either be limited, or the rest shape discretization must be modified after large plastic flows. Most existing methods use a finite-element-based approach using a tetrahedral mesh discretization and require complex, computationally expensive volumetric remeshing operations to maintain stability. We introduce a point-based approach to animating elastoplastic materials that avoids the need for remeshing, only requiring nearest-neighbors queries. As an added benefit, artists frequently work with point primitives, for example in particle systems, so they can leverage their tools and experience to postprocess simulation results.

Our second contribution is the method for creating stylized, animated artistic assets that we call *Dynamic Sprites*. While powerful tools exist for creating stylized static images, animating such images is a challenging problem. Stylized object behaviors are often non-physical, so traditional simulation techniques are unable to produce satisfactory results. Our approach allows artists to intuitively describe object behaviors using example deformations. These objects exhibit stylized motions when placed in an interactive environment. We introduce a set of controls that allow artists flexibility with respect to the laws of physics, while using physics to naturally handle collision response and timing. The resulting animations have exaggerated motion that matches the stylized static images provided as input.

Finally, we introduce a new artist-guided technique for animating destructive scenes containing near-rigid objects. For these scenes, the most important visual features are plastic deformation and fracture. State-of-the-art methods either animate objects using finite-element-based elastoplastic simulation techniques, with all the challenges described above, or model them as purely rigid and do not support plastic deformation. Elastic vibrations are mostly imperceptible, so spending computational resources to simulate them is a waste of effort, but we still want to capture plastic deformation. We introduce an example-based model for plasticity while computing dynamics and fracture using existing

rigid-body techniques. Artists provide example deformations and fracture patterns while authoring a scene. At runtime, we simulate objects as rigid bodies and deform and fracture objects based on the impulses computed by the rigid body simulator. The resulting animations contain physically plausible deformations that match the style of the provided examples.

1.1 Thesis Statement

Physics-based animation is a tool for artists, so simulation methods must be designed to address the needs and challenges of artists. This approach requires rethinking design decisions and tradeoffs made for engineering applications such as choice of discretization primitive, and developing new material models that provide flexible artistic control.

CHAPTER 2

RELATED WORK

Since the 1980s, researchers have developed techniques to animate a huge variety of materials, including rigid bodies, deformable bodies, gasses, and liquids. Since the methods described in this thesis are focused on deformable solids, our survey of related work will also be concentrated on those phenomena.

2.1 Elasticity

The use of simulation to animate elastic materials dates back to Terzopoulos and colleagues in the late 1980s [Terzopoulos et al. 1987]. Their work was the first to solve the partial differential equations resulting from continuum mechanics to compute motion of deformable objects. Most modern approaches are based on the finite-element discretization described by O'Brien and Hodgins [1999]. O'Brien and Hodgins used a quadratic strain measure in their approach that is invariant to rotations, but is less stable and more expensive than a linear strain measure. Müller and colleagues introduced *stiffness warping*, which permits the use of a linear strain measure while avoiding linearization artifacts, even in the presence of rotational deformation [Müller et al. 2002; Müller and Gross 2004]. Irving and colleagues introduced *invertible finite elements*, a method stable even in the presence of extreme deformations [Irving et al. 2004]. Successful production systems have been developed using these techniques for both real-time [Parker and O'Brien 2009] and offline applications [Cole 2011].

A variety of meshless methods have been developed for animating elasticity. The conceptually most simple approach is to connect point masses with springs (e.g. [Baraff and Witkin 1998; Liu et al. 2013]). However, these approaches are not based on continuum mechanics and typically do not converge under refinement; forces depend on mesh topology and spring stiffnesses rather than elastic parameters such as Young's modulus or Poisson's ratio. Müller and colleagues [2004] developed a point-based approach using a moving least squares approximation of the deformation gradient and is based on continuum mechanics. Deformation is estimated by animating the relative motion of each particle and its neigh-

bors. Many researchers, including ourselves, use a similar moving least squares approach [Gerszewski et al. 2009; Zhou et al. 2013]. Martin and colleagues [2010] introduced *elastons*, which provide a unified point-based approach for elastic volumes, shells, and rods. Their method relies on a generalized moving least squares estimate of deformation that considers derivative information, avoiding the need for volumetric sampling around shells and rods.

For many real-time applications, continuum-mechanics-based approaches are too computationally expensive, and in the case of finite-element approaches, complicate modeling since they require volumetric (typically tetrahedral or hexahedral) meshes. This has led researchers to develop geometric methods that produce plausible deformations but have low implementation complexity and runtime cost, and do not require volumetric meshes. The first popular technique is known as position-based dynamics, first developed by Jakobsen for the video game *Hitman* [2001] and later formalized by Müller and colleagues [2007]. In this framework, objects are modeled as a set of particles and elasticity is emulated through the use of constraint functions. Particles are integrated through time independently using a first-order forward Euler scheme, and then the particle positions are iteratively adjusted to satisfy the constraints, which are functions of positions only (i.e. do not depend on velocity). This approach is incredibly flexible: the solver can operate on any constraint functions of position. Researchers have developed constraints to model a variety of materials, including gasses and liquids at interactive rates [Macklin and Müller 2013; Macklin et al. 2014]. Because position-based dynamics modifies positions of particles directly, rather than by applying forces, animations often suffer from artificial damping.

Another geometric approach for animating deformable bodies is *shape matching*, developed by Müller and colleagues [2005]. In this method, objects are divided into *shapes* –collections of particles. Elastic forces are computed by finding a best-fitting rigid transformation from the rest position of a shape to its current world configuration and then applying spring forces to pull particles toward the rigidly transformed shape. The main advantage of this approach is that shapes are a very flexible primitive: they can consist of a few particles, or the entire object. For example, Rivers and James [2007] developed an optimized approach using shapes defined as overlapping subsets of a voxel lattice. One limitation of the original shape-matching approach is that shapes must occupy a volume, making it poorly suited to modeling strands or hair, for which points are nearly collinear. Müller and Chentanez [2011] proposed a solution to this problem by associating orientations with particles, which reduced volumetric sampling requirements. While this approach relies on summing rotation matrices, which will not compose or interpolate rotations, the authors

produced compelling results.

Though Müller and colleagues provided a parameter range where shape matching is provably stable for a single shape, this analysis breaks down in the presence of nonlinearities introduced by many overlapping clusters. Bargteil and Jones [2014] proposed a strain limiting approach similar to position-based dynamics to improve stability when using multiple clusters while avoiding the artificial damping of pure position-based dynamics.

2.2 Plasticity

Terzopolous and Fleischer recognized the importance of animating plastic phenomena, extending their pioneering work simulating elastic materials to include plasticity just one year later [1988]. O'Brien and colleagues similarly incorporated plasticity into their finite-element approach to simulate ductile fracture [2002]. Early work in graphics relied on an additive plasticity model, in which total strain is the sum of elastic and plastic components. Irving and colleagues [2004] recognized that such an additive model is appropriate only for infinitesimal deformations, and that for finite deformations a multiplicative model is more appropriate. Bargteil and colleagues [2007] proposed a method for performing such a factorization that incorporates important features of plasticity such as plastic yield, work hardening/softening, and creep.

With a suitable model for how and when plastic deformation should occur, the remaining challenges for simulating plastic materials are related to ensuring simulation stability in the presence of large plastic flows. Bargteil and colleagues performed wholesale, conforming volumetric remeshing when basis functions became sufficiently ill-conditioned. Wojtan and Turk [2008] used a nonconforming tetrahedral mesh, greatly reducing time spent performing remeshing. They also avoided visual artifacts by maintaining separate surface and simulation meshes. Both of these approaches remesh the entire domain, interpolating simulation variables to the new tetrahedra. This approach causes artificial smoothing and diffusion. To combat this, Wicke and colleagues [2010] proposed a method based on local-remeshing, where only elements with poorly conditioned basis functions are remeshed. There are two issues that complicate this approach. First, volumetric meshing generally, and local remeshing specifically, are challenging, poorly understood problems; solutions often rely on heuristics that seem to work "well-enough" in practice with very few quality guarantees. Second, plastic deformation modifies the rest shape of each element independently, so when remeshing is required, tetrahedra no longer fit together. In order to apply topological operations such as edge flips, Wicke and colleagues construct a consistent *material space*

which minimizes internal strain energy, storing a per-element map from rest space to this material space. By extending the set of mesh improvement operations and introducing an approach to enforce incompressibility without locking, Clausen and colleagues [2013] extended this technique to animate materials ranging from purely elastic to purely liquid. These finite-element-based techniques produce extremely high-quality results, but have very high implementation complexity.

Meshless methods have also been developed for animating plasticity. Müller and colleagues [2004] divided plasticity into two regimes, storing either plastic offsets for mostly elastic materials, or elastic offsets for mostly plastic materials. This allowed them to model mostly elastic objects with limited permanent deformation, or liquid-like materials with limited elasticity. Gerszewski and colleagues [2009] compute elastic forces by tracking the deformation gradient of the object rather than its rest state. This avoids instabilities when the rest state may become degenerate, for example deformed into a coplanar configuration, but is susceptible to drift and can therefore handle limited elastic deformation. Zhou and colleagues [2013] adapted the technique of Gerszewski and colleagues to use implicit integration to improve stability. Our approach, *deformation embedding*, explicitly maintains the rest state of the object, allowing for arbitrary elastic deformations.

Other researchers have modeled plastic materials as mostly-liquid, modifying fluid simulators to incorporate elastic forces. Goktekin and colleagues [2004] used an eulerian fluid simulator, adding elastic forces computed by tracking the elastic strain throughout the material. Clavet and colleagues [2005] developed a meshless approach using an SPH-like fluid simulator and incorporating elasticity by adding spring forces between particles. Plasticity is incorporated by changing spring rest lengths. Since these approaches model materials as modified fluids, animated materials behave more like fluids than solids.

2.3 Fracture

As with plasticity, fracture was quickly recognized as an important visual feature when animating deformable solids. Terzopolous and Fleicher [1988] incorporated fracture along with plasticity one year after their seminal paper on simulating elastic bodies. Approaches based on finite-elements are common in practice, and typically rely on the method of O'Brien and Hodgins [1999] and its refinements [O'Brien et al. 2002; Parker and O'Brien 2009; Pfaff et al. 2014]. To avoid the complexity of meshing, Pauly and colleagues [2005] developed a meshless approach. Hegemann and colleagues [2013] proposed a level-set-based technique capable of handling topological changes implicitly, capturing small-scale surface

details without high-resolution volumetric meshing.

For brittle materials, it is often undesirable to compute elastic dynamics, when only rigid motion and fracture are desired. Müller and colleagues [2001] were able to animate fracture in real time by treating objects as rigid for the purpose of dynamics, and performing a quasistatic solve to incorporate plastic deformation and fracture in between timesteps. Bao and colleagues [2007] improved on this approach by projecting out the null space (i.e. rigid motion) during quasistatic analysis and supporting a simple plasticity model.

To support artistic control, and to reduce computational costs during simulation, a variety approaches based on prescoring have been developed. Su and colleagues [2009] aligned a fracture pattern to the location of impacts to produce plausible and controllable fracture. By performing an approximate convex decomposition, Müller and colleagues [2013] were able to achieve high-quality, intricate fractures in real time.

2.4 Example-Based Deformation

One of the biggest challenges when using physics-based animation is providing artists with appropriate controls over the outcome of simulations. For modeling tasks, artists have precise control over their meshes. Likewise, for keyframe animations, artists have control of both the keyframes and animation curves between them. Once an artists has chosen to use physics-based animation, however, their control is limited to material properties and initial conditions. A popular solution is to allow artists to iterate using lower resolution simulations and then add detail to create a high-resolution animation [Bergou et al. 2007; Kavan et al. 2011].

Martin and colleagues [2011] proposed an alternative approach: example-based deformations. Artists provide example deformations of their model and an extra potential is included to create internal forces that act to return a deformed object to the *example manifold* – interpolations of the provided examples. The main benefit of this approach is that artists have intuitive controls over high-level object behavior (i.e. shapes), but features such as global motion and collision response are handled automatically. Since its introduction, researchers have proposed more efficient formulations, suitable for real-time scenarios [Schumacher et al. 2012; Koyama et al. 2012]. Our work is inspired by the example-based simulation paradigm and extends the method, addressing shortcomings of the original approach and adapting it to new applications.

These example-based approaches require a method of interpolating between deformed shapes. Martin and colleagues interpolated the per-element strains of mesh tetrahedra to compute intermediate shapes. As described above in the case of plastic deformation, these

deformed tetrahedra do not "fit together," so they must perform an optimization to find a similar, realizable configuration. Schumacher and colleagues [2012] avoided explicitly performing this reconstruction to improve efficiency. In their shape-matching framework, Koyama and colleagues [2012] linearly interpolated the scale-shear component of their shape basis matrices. In contrast, our approach is based on linear blend skinning; each example is defined as the set of transformations for each bone and shape interpolation is performed by interpolating these bone transformations.

CHAPTER 3

DEFORMATION EMBEDDING FOR POINT-BASED ELASTOPLASTIC SIMULATION

3.1 Introduction

Computer animation of elastoplastic materials, such as modeling clay, chewing gum, and bread dough, has long been a goal of computer graphics, probably because these materials demonstrate such intriguing behavior. As a field we have made progress toward this goal and elastoplastic materials have recently been showcased in special effects, for example the honey in *Bee Movie* [Ruilova 2007] and the food in *Ratatouille* [Goktekin et al. 2007]. However, significant limitations in current techniques remain: some handle only limited plastic deformation, some handle only limited elastic deformation, and others require complex remeshing methodologies.

We present a straightforward, easy-to-implement, point-based approach for animating elastoplastic materials. Our approach can handle extreme elastic and plastic deformations. Because the approach is point-based, there is no need for complex remeshing—the corresponding operation is a simple neighborhood query.

A material that has undergone plastic deformations does not, in general, have a zero-stress state that can be realized in three-dimensional space. Put another way, the *rest space* is not embeddable in three-dimensional space. This fact poses a challenge because elastic forces depend on a mapping from rest space to deformed or *world space*. Some authors have addressed this challenge by keeping plastic offsets from an initial rest state, but this places limitations on the amount of plastic deformation that is possible. Others have abandoned the rest state and keep elastic offsets, resulting in limited elastic deformations and/or artificial plasticity. Borrowing an idea from Wicke and colleagues [2010], we take a compromise approach and store an *embedded space*—the weighted least-squares best embedding of rest space into three dimensions, while preserving the rest space exactly per particle. Nearest neighbor queries can be performed efficiently in embedded space, allowing us to update particle neighborhoods, while deformation can be computed accurately from

the rest space. Without updating particle neighborhoods, simulations become unstable under large plastic flows. By maintaining both spaces, our method can handle large plastic and elastic deformation simultaneously.

While our primary contribution is the development of our embedding into three dimensions, we also suggest an approach for estimating volume that allows for nonuniform samplings. We demonstrate our approach on a variety of examples that display a large range of material behaviors, including simultaneous elastic and plastic deformations. Our method is the first point-based approach capable of simulating these examples, and is much simpler to implement than competing finite element approaches.

It is worth noting that most previous point-based approaches for animating large plastic flows incorporated SPH-like *pressure forces*. Such forces provide additional stability [Gerszewski et al. 2009] by favoring uniform sampling of simulated materials. However, they also alter the behavior of the underlying material model, preventing, for example, the simulation of hyperelastic materials. Moreover, such forces do not allow for adaptive sampling without employing complex adaptive simulation frameworks [Adams et al. 2007]. By eschewing these nonphysical pressure forces and relying exclusively on elastic forces and a volume preserving plasticity model, our framework allows for the simulation of arbitrary material models. While a subtle point, this represents a significant scientific advance and our examples demonstrate far larger plastic deformations than previously possible with point-based methods for purely elastoplastic materials.

3.2 Method

The core of our approach is the maintenance of three domains: world space, rest space, and embedded space. *World space* refers to the current, deformed object configuration. *Rest space* is local to a particle and refers to the particle's preferred relative positions of its neighbors, stored as 3D vectors per neighbor. In general, the rest space cannot be represented as a set of 3D points, as per-neighbor vectors cannot be reconciled exactly between all particles. We introduce the *embedded space*, which is a global, least-squares best fit of the rest space into three dimensions. It can be thought of as an approximate reference configuration for the material. The rest space and world space are used to compute the deformation gradient, from which we can compute elastic forces and plastic deformations. The embedded space allows efficient updates to neighborhoods in the presence of plastic deformation. Our method is summarized in Algorithm 1 and visualized in Figure 3.1.

Algorithm 1 Deformation Embedding for Elastoplasticity

```
initialize kernel support radii and particle neighborhoods
while Animating do
    approximate deformation gradient
    factor deformation into elastic and plastic components
    compute elastic forces
    integrate elastic, body, and damping forces
    compute global embedding
    resample particles
    update neighborhoods
end while
```

3.2.1 Notation

We use $\mathbf{x}_{ij}$, $\mathbf{e}_{ij}$, $\mathbf{u}_{ij}$ to denote to *neighbor vectors*—vectors between particles i and j—in world, embedded, and rest space, respectively. Capital bold face letters refer to matrices. The index j refers to the neighbors of particle i, which may change during the course of the simulation.

3.2.2 Approximating Deformation

Following the work of Müller and colleagues [2004], we use moving least squares (MLS) to approximate the deformation gradient, $\mathbf{F}$. For each particle i, we seek the matrix $\mathbf{F}_i$ that minimizes

$$\sum_j ||w_{ij}(\mathbf{F}_i\mathbf{u}_{ij} - \mathbf{x}_{ij})||^2 \tag{3.1}$$

where the sum is taken over the neighbors, j, and w_{ij} is a weighting kernel evaluated in rest space. We compute the minimizer by solving the following linear system,

$$\mathbf{F}_i(\sum_j w_{ij}\mathbf{u}_{ij}\mathbf{u}_{ij}^T) = \sum_j w_{ij}\mathbf{x}_{ij}\mathbf{u}_{ij}^T. \tag{3.2}$$

We refer to the quantity $\sum_j w_{ij}\mathbf{u}_{ij}\mathbf{u}_{ij}^T$ as the particle i's basis matrix, $\mathbf{A}_i$, which we invert to solve the system. Note that if the rest space vectors $\mathbf{u}$ become coplanar, then $\mathbf{A}$ becomes singular.

3.2.2.1 Particle Volume

It is common in point-based animation to assume that particles have a fixed, often uniform, mass or volume. This assumption often leads to density fluctuations that can cause spurious oscillations. In the context of plastic deformation, it is desirable to allow the sampling to vary over time and thus we require a way of estimating the volume occupied by a particle. Noting that $\mathbf{A}$ serves the same purpose as the basis matrix in tetrahedral finite

element methods, that the volume of a tetrahedron is a multiple of the determinant of the basis matrix, and that the units of $\det(\mathbf{A})$ are m^6, we approximate the volume of a particle as,

$$V_i = \sqrt{\frac{\det(\mathbf{A}_i)}{(\sum_j w_{ij})^3}}. \tag{3.3}$$

The denominator normalizes with respect to the weights used to compute $\mathbf{A}$, which are cubed by the determinant operation. We assign a constant density to the material and allow each particle's mass to change over time. The total mass of the material fluctuates slightly due to this approximation (see Section 3.3), but we did not observe any resulting artifacts in our examples.

3.2.3 Elastoplastic Model

We use a multiplicative plasticity model, where the total deformation gradient is the product of elastic and plastic deformations, i.e. $\mathbf{F} = \mathbf{F}_e\mathbf{F}_p$.

Following Irving and colleagues [2004], we diagonalize $\mathbf{F}_e = \mathbf{U}\hat{\mathbf{F}}_e\mathbf{V}^T$ and compute the diagonalized first Piola-Kirchhoff stress:

$$\hat{\mathbf{P}} = \lambda \mathrm{tr}(\hat{\mathbf{F}}_e - \mathbf{I})\mathbf{I} + 2\mu(\hat{\mathbf{F}}_e - \mathbf{I}). \tag{3.4}$$

We then rotate the diagonalized stress $\mathbf{P} = \mathbf{U}\hat{\mathbf{P}}\mathbf{V}^T$ and compute symmetrized forces

$$\begin{aligned} \mathbf{f}_i &= V_i\mathbf{P}_i\mathbf{A}_i^{-1}w_{ij}\mathbf{u}_{ij} \\ \mathbf{f}_j &= -\mathbf{f}_i \end{aligned} \tag{3.5}$$

which are linear in position.

Using the plasticity model developed by Bargteil and colleagues [2007], we factor our deformation gradient, $\mathbf{F}$, into elastic and plastic components $\mathbf{F}_e$ and $\mathbf{F}_p$, respectively. This model preserves volume in rest space, and accounts for a range of material properties including yield point, creep, and work hardening/softening. Specifically, we compute the singular value decomposition of $\mathbf{F} = \mathbf{U}\hat{\mathbf{F}}\mathbf{V}^T$ and factor the diagonal matrix $\hat{\mathbf{F}}$ into $\hat{\mathbf{F}}_e\hat{\mathbf{F}}_p$. To compute $\hat{\mathbf{F}}_p$, we first compute

$$\tilde{\mathbf{F}} = \frac{\hat{\mathbf{F}}}{\det(\hat{\mathbf{F}}^{1/3})}. \tag{3.6}$$

This computation is essentially normalizing $\hat{\mathbf{F}}$ to have determinant 1, ensuring volume preservation. Since $\hat{\mathbf{F}}$ is diagonal, we can perform the cube root operation on each entry independently. We then compute $\hat{\mathbf{F}}_p = \hat{\mathbf{F}}^\gamma$, where

$$\gamma(\mathbf{P}, P_Y, \nu, \alpha, K) = \frac{\nu\left(||\mathbf{P}|| - \max\left(P_Y + K\alpha\right)\right)}{||\mathbf{P}||} \tag{3.7}$$

which is clamped between 0 and 1. P_Y is the yield stress (at lower stresses there is no plastic flow); ν is the flow rate, which controls how quickly flow will occur; α is the total accumulated plastic stress as computed by the simulation; and K controls the rate of work hardening or softening. P_Y, ν, and K are user-defined, while α is a simulation variable integrated through time. The $K\alpha$ term can be thought of as adjusting the yield point. To prevent rotation of the plastic deformation, we rotate the diagonalized deformation into $\mathbf{F}_e = \mathbf{U}\hat{\mathbf{F}}_e\mathbf{V}^T$ and $\mathbf{F}_p = \mathbf{V}\hat{\mathbf{F}}_p\mathbf{V}^T$. We use $\hat{\mathbf{F}}_e$ to compute elastic stress in Equation 3.4. For more details, refer to Bargteil and colleagues [2007].

For damping forces, we use the SPH viscosity formulation of Müller and colleagues [2003]. These forces act to minimize velocity differences between neighboring particles, and are computed by

$$\mathbf{f}_{ij} = \eta \frac{m_j}{\rho_j} (\mathbf{v}_j - \mathbf{v}_i) \nabla^2 W(\mathbf{u}_{ij}, h). \tag{3.8}$$

Note that this simple model damps all relative motion and is non-zero for rigid rotation. Alternatively one could base damping forces on the time derivative of stress.

To update positions and velocities, we use either an explicit leapfrog scheme, or the following Newmark integration scheme adapted from Bridson and colleagues [2003], which integrates viscous forces implicitly and elastic forces explicitly:

- $v^{n+1/2} = v^n + \frac{\Delta t}{2} a_{viscous}(t^n, x^n, v^{n+1/2})$
- $x^{n+1} = x^n + \Delta t v^{n+1/2}$
- $v^{n+1} = v^{n+1/2} + \frac{\Delta t}{2} a_{elastic,body}(t^{n+1}, x^{n+1}, v^{n+1/2})$

Each timestep of the Newmark scheme is more expensive than the explicit scheme, but for simulations where high viscosity is desired, the implicit viscous solve permits larger timesteps, resulting in better performance overall. We experimented with linearly implicit Euler integration, but found that rotational motion was severely damped (see accompanying video) and did not pursue this further.

3.2.4 Embedding Deformation

As the material undergoes plastic deformation a particle's neighbors may move far away in rest space and no longer provide useful information. To update these neighborhoods when such changes occur, we maintain a globally optimal embedded space, encoded as a three-dimensional position, $\mathbf{e}_i$, for each particle. To update these positions, we first compute temporary rest space vectors that incorporate the plastic deformation that occurred over the previous timestep, $\tilde{\mathbf{u}}_{ij} = \mathbf{F}_p\mathbf{u}_{ij}$.

The optimal least-squares embedding of the particles into three-dimensional space will minimize the discrepancy of neighbor vectors between the embedded and rest spaces. We formulate this optimization over the embedded positions, $\mathbf{e}_i$, as a weighted linear least-squares problem

$$\underset{\mathbf{e}_i}{\operatorname{argmin}} \sum_{i,j} ||w_{ij}(\tilde{\mathbf{u}}_{ij} - \mathbf{e}_{ij})||^2, \tag{3.9}$$

which requires solving three decoupled, over-constrained linear systems: one for each dimension, x, y, and z. In matrix form, each system can be expressed as

$$\mathbf{Ce} = \mathbf{u}. \tag{3.10}$$

$\mathbf{C}$ is similar to the constraint matrices that appear when using Lagrange multipliers or projection methods for constrained dynamics. In our case, each row, r, of $\mathbf{C}$ encodes the (weighted) constraint that if particle j is a neighbor of particle i then $\mathbf{e}_{ij} = \tilde{\mathbf{u}}_{ij}$. More specifically,

$$\begin{aligned} \mathbf{C}_{ri} &= -w_{ij} \\ \mathbf{C}_{rj} &= w_{ij}. \end{aligned}$$

The vector $\mathbf{e}$ contains the embedded space positions for all particles and $\mathbf{u}_{ij} = w_{ij}\tilde{\mathbf{u}}_{ij}$. Note that while $\mathbf{e}_{ij} = -\mathbf{e}_{ji}$, the same is not true for $\tilde{\mathbf{u}}_{\mathbf{ij}}$ and $\tilde{\mathbf{u}}_{\mathbf{ji}}$ because rest space vectors, $\mathbf{u}_{ij}$ are transformed by $\mathbf{F}_{\mathbf{p_i}}$ each timestep and diverge over time.

Because each particle has roughly 32 neighbors (constraints), $\mathbf{C}$ is roughly $32n \times n$. We solve this non-square, highly over-constrained system by applying conjugate gradients to the normal equations,

$$\mathbf{C}^T\mathbf{Ce} = \mathbf{u}. \tag{3.11}$$

Denoting the set of neighbors of i as $n(i)$, the entries of $\mathbf{C}^T\mathbf{C}$ are

$$\left(\mathbf{C}^T\mathbf{C}\right)_{ij} = \begin{cases} 2\sum\limits_{k \in n(i)} w_{ik}^2 & \text{if } i = j \\ -2w_{ij}^2 & \text{if } i \in n(j) \wedge j \in n(i) \\ -w_{ij}^2 & \text{if } i \in n(j) \otimes j \in n(i) \\ 0 & \text{otherwise.} \end{cases} \tag{3.12}$$

This matrix is symmetric and diagonally dominant, but if all the neighborhoods are symmetric, will have a null-space containing the constant vector (corresponding to global translations). We explicitly remove this by adding a row to $\mathbf{C}$ constraining the first particle to maintain its current embedded position. Note that the system is *not* invariant to global

rotations, so no special handling is required. Because the system changes structure as neighborhoods change or particles are split or merged, prefactoring is not possible.

While these embedded space positions provide a globally consistent reference configuration, there will still be discrepancy between the rest and embedded spaces. The vectors $\tilde{\mathbf{u}}_{ij}$ become the rest space vectors $\mathbf{u}_{ij}$ for the next timestep.

It is natural to compare our embedding to the elastic-energy minimization approach of Wicke and colleagues [2010]. Interestingly, our linear embedding does not capture some rotational changes to the rest space. However, world space dynamics are nearly identical compared to the more expensive nonlinear optimization used by Wicke and colleagues, as shown in Figure 3.2. This behavior is because rotations typically do not change the neighbors with the highest smoothing weights (those closest to the particle), and the rest-space vectors for those particles are preserved exactly. In our experiments, our linear solve was nearly twice as fast as the elastic-energy minimization approach of Wicke and colleagues.

Another difference between our approach and that of Wicke and colleagues [2010] is in how we store rest space. Wicke and colleagues store a single 3×3 matrix per tetrahedron, which they call a *plastic offset*, that maps from embedded space to rest space. Instead, we store rest space as about 30 rest-space neighbor vectors per particle. While our approach requires more memory, we found storing a single matrix per particle problematic as a simple least-squares fit introduced too much error into the mapping from embedded to rest space. If memory consumption is a significant concern, higher order fitting techniques and/or compression may prove effective.

3.2.5 Updating Neighborhoods

After plastic flow, a particle's neighbors may have moved far away, and no longer provide useful information about the deformation gradient. We therefore update each particle's neighborhood by finding the nearest neighbors in the embedded space. We expect that neighborhoods in embedded space are a good approximation of neighborhoods in rest space, and they can be queried efficiently using a KD-tree. The only information lost in this process is the difference between the rest space vector, $\mathbf{u}_{ij}$, and the embedded space vector, $\mathbf{e}_{ij}$, when a particle j moves out of particle i's neighborhood. When a new particle enters a neighborhood, we initialize $\mathbf{u}_{ij} = \mathbf{e}_{ij}$. This process does lose some information about the rest state (and, consequently, the internal stress), but the lost information is the least useful as it comes from the particles that are farthest away.

3.2.6 Particle Resampling

To improve computational efficiency and stability, we enable particle resampling in our simulator. We selectively split and merge particles when neighborhood sampling is either too dense, or too sparse. We quantify this by computing a basis matrix for each particle as

$$\mathbf{B}_i = \sum_j \frac{\mathbf{u}_{ij}\mathbf{u}_{ij}^T}{||u_{ij}||^4}, \tag{3.13}$$

where the index j runs over the neighbors of particle i, which may vary over time. Note that this is simply a weighted covariance matrix, where the 4th power in the denominator results in a $1/r^2$ falloff away from particle i.

We perform an eigendecomposition of this matrix and examine its eigenvalues to decide to split or merge particles. If the maximum eigenvalue is small, this indicates that there are too few particles nearby, and the particle should be split. We split the particle into two particles and place them along the eigenvector with the minimum eigenvalue. They are offset from the particle's original position by half the average distance to the particles in the neighborhood.

Splitting particles near the surface can potentially cause rendering artifacts on the object's surface, which we attempt to mitigate in two ways. First, we chose the middle eigenvalue as a splitting direction because for surface particles, the smallest eigenvalue is typically perpendicular to the surface, and the largest is already well sampled. The middle eigenvector corresponds to a direction tangent to the surface that is poorly sampled. Second, we reject splits that are likely to cause surface artifacts. We compute the distance from the original particle to the center of mass of its neighborhood. If either split particle is more than a factor of $\sqrt{2}$ away from the center of mass, the split is cancelled. Intuitively, this condition attempts to eliminate splits that are not tangent to the surface, as shown in Figure 3.3. We also cancel splits that would place a new particle inside of an obstacle. We classify particles as "surface particles" if the distance from the particle to its neighborhood center of mass is greater than a user-defined threshold, as interior particles likely have a uniformly distributed neighborhood, while surface particles will have a neighborhood skewed away from the surface.

If the minimum eigenvalue is too large, then there are too many particles nearby and the particle should be merged with its nearest neighbor. The merged particle is placed halfway between the two original particles. The split and merge thresholds are a user-specified parameter, but reasonable values can be chosen by examining minimum and maximum basis matrix eigenvalues from the object's initial configuration.

While much more complex resampling methods exist [Adams et al. 2007; Ando et al. 2012], we consider the ease of implementation of our technique to be a major benefit.

3.2.7 Implementation Details

For our weighting kernels, we use the standard SPH kernels of Müller and colleagues [2003]: The *poly6* kernel, Equation 3.14, for all weights except for viscosity, where we use the SPH viscosity kernel, Equation 3.15.

$$W_{\text{poly6}}(\mathbf{r}, h) = \frac{315}{64\pi h^9} \begin{cases} \left(h^2 - r^2\right)^3 & 0 \leq r \leq h \\ 0 & \text{otherwise} \end{cases} \tag{3.14}$$

$$\nabla^2 W(\mathbf{r}, h) = \frac{45}{\pi h^6} (h - r) \tag{3.15}$$

We initialize each particle's smoothing radius by finding its nearest 32 neighbors, and setting its radius to twice the average neighbor distance. During the simulation, a particle can have a maximum of 32 neighbors, which must be within its radius in embedded space. This constraint limits the number of neighbors and the number of non-zeroes in our matrices.

3.2.7.1 Rendering

Generating visually appealing surfaces from particle data for rendering is a difficult problem. In the examples in this paper, we used two methods: the skinning method of Bhattacharya and colleagues [2011], and a simple surface mesh embedding technique. When embedding a surface mesh, we update mesh vertex positions with the weighted average of the displacements of nearby particles from the first frame of animation. The weights are computed using the *poly6* kernel with a user-defined, constant support radius. This approach is sufficient for simulations with largely elastic deformations, but breaks down for large plastic flows. Including more robust surface tracking and using the embedded space to update weights are interesting directions for future work.

3.3 Results and Discussion

We have used our proposed method to simulate materials exhibiting a wide range of elastoplastic behavior as shown in our figures and included videos. All examples were run on a dual-2.8 GHz Intel Xeon processor machine using up to 12 cores. Trivially parallelizable loops were multithreaded using the Intel TBB library.

Figures 3.2 and 3.4 demonstrate the importance of the two components of our mapping to rest space: plastic offsets and embedded space. Without plastic offsets, the final world pose of the bar in Figure 3.2 would not include any twist. Without neighborhood updates, a

particle's neighborhood may become degenerate, leading to an ill-conditioned basis matrix and instability. By using embedded space to update neighborhoods, we maintain well sampled neighborhoods and increase stability (see Figure 3.4). Resampling also improves stability by merging particles that become extremely close in embedded space.

Figure 3.5 demonstrates our method's ability to animate a wide range of plastic parameters, including spatially varying plasticity.

Figures 3.6 and 3.7 compare the world space and embedded space deformations of a bunny dropped onto obstacles. The embedded space captures the key features of the global plastic deformation. The high-frequency details of the original mesh are preserved as the material flows plastically.

Figure 3.8 demonstrates our method's robust handling of uneven sampling. Our new volume estimate allows us to simulate objects with dramatically varying particle densities. Previous work has required expensive fully adaptive sampling techniques [Adams et al. 2007] or special handling of boundaries where regions of varying particle density meet [Solenthaler and Gross 2011].

As the "upset fowl" collides with the rigid obstacles in Figure 3.9, it undergoes significant elastic and plastic deformation. As it is crushed, it reacts elastically, lifting the upper block before succumbing to creep and flowing plastically.

As shown in Figure 3.10, our volume estimate closely tracks both the "ground truth" volume computed using the skinned, embedded space mesh, and the estimate used by SPH based approaches.

Timing results are shown in Table 3.1 and Table 3.2.

3.3.1 Limitations and Future Work

In practice, we found our method to be robust and stable for bulk motion. Typical failure cases are a single particle or a small group of particles drifting away from the bulk material, or getting caught on an obstacle (we colloquially refer to such particles as "jerk particles"). These artifacts are usually induced by extreme or violent deformations or sharp corners of rigid obstacles. In a production environment, these troublesome particles can be easily deleted before skinning or rendering. Using a robust least squares solve, or minimizing the $\mathcal{L}^1$ norm in our embedding, may eliminate these problems, albeit at significant additional cost.

In the current implementation, we allow particles to apply forces through solid obstacles, which can lead to unnatural and undesirable behavior near sharp corners. Also, we do not check for collisions between particles nearby in world-space, but far apart in embedded

space, which can result in interpenetration artifacts. We suspect both problems can be efficiently resolved with an additional spatial partitioning scheme in world space.

Although our simple resampling scheme works to improve stability, our splitting method can cause surface artifacts. Increasing the model resolution reduces these artifacts, but a more robust resampling or skinning approach is a possible direction of future work. Additionally, aggressively coarsening regions on the object interior could lead to improved performance while preserving surface detail.

It is common for materials to become brittle after work hardening and become prone to fracture. In our implementation, such topological changes happen by accident, when particle neighborhoods change and parts of the material stop interacting. By intentionally causing such topological changes, our method could be adapted to simulate ductile fracture. Relatedly, we do nothing in particular to address "fusing" in embedded space, but did not notice this to be a problem in our examples. One could address such issues using an embedded surface mesh and ray casting when computing neighborhoods. We also note that many previous techniques, both point-based [Gerszewski et al. 2009] and mesh-based [Bargteil et al. 2007], allow such fusing and sometimes consider it a "feature."

Because points behave nearly independently, this method is well suited to parallelization on multicore architectures or GPUs. Most computations only output results for a single particle, and read only from nearby particles. We employed multithreading to parallelize these independent loops on the CPU; however, these access patterns are also well suited for the memory hierarchy on current GPU architectures. Preliminary GPU implementation efforts resulted in an order of magnitude speedup for simple elastic examples.

We have presented a simple to implement point-based method for animating elastoplastic materials. By maintaining a globally optimal fit of the material's reference configuration we are able to simulate materials undergoing simultaneous and extreme elastic and plastic deformations. Like all point-based methods, our approach trades the numerical advantages of discretizing the domain into disjoint elements (better conditioning, sparser matrices, etc.) for the significant implementation advantage and simplicity of avoiding computing such a discretization altogether.

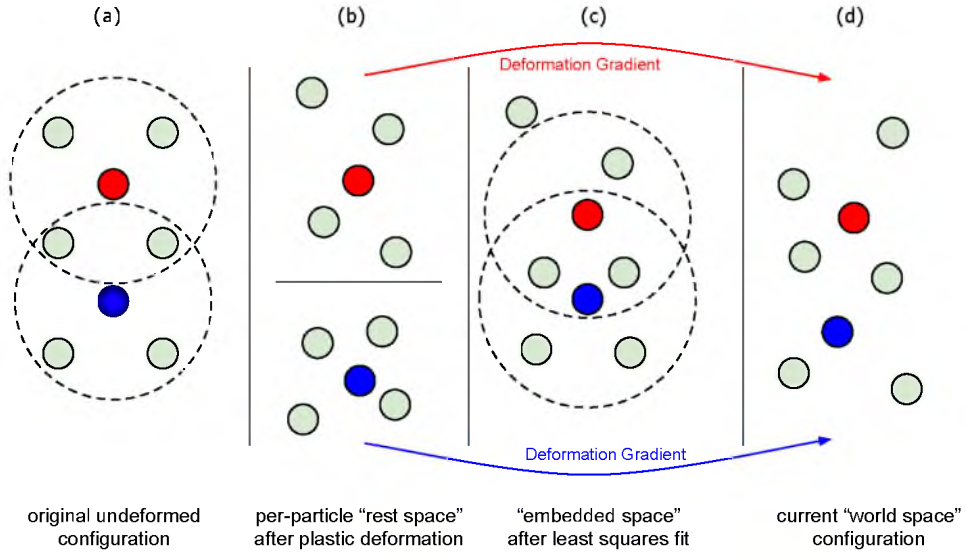

Figure 3.1. From the initial configuration (a), each particle's neighborhood undergoes plastic deformation, resulting in a new *rest space* configuration (b). However, the red and blue particles disagree about where their shared neighbors should be. Performing a least squares global fit, we obtain the *embedded space* configuration (c), which is used to update particle neighborhoods. Each particle's deformation gradient maps from its own rest space to the current *world space* configuration (d).

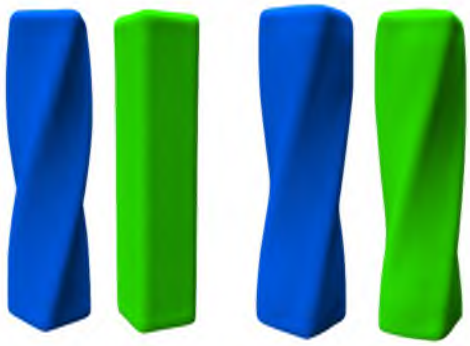

Figure 3.2. Comparison between our linear embedding (left) and nonlinear embedding (right) for a twisted plastic bar. The world space behavior (blue) is nearly identical, even though the linear embedding captures very little rotation. Though small, changes in the linear embedding did cause neighborhoods to change in this example.

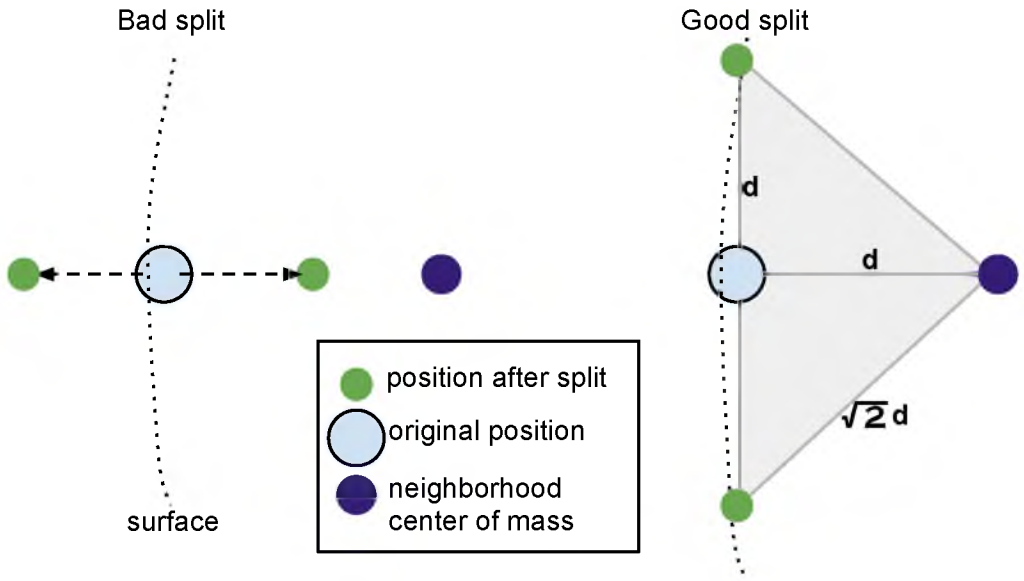

Figure 3.3. We cancel splits that are likely to cause popping artifacts near the object surface.

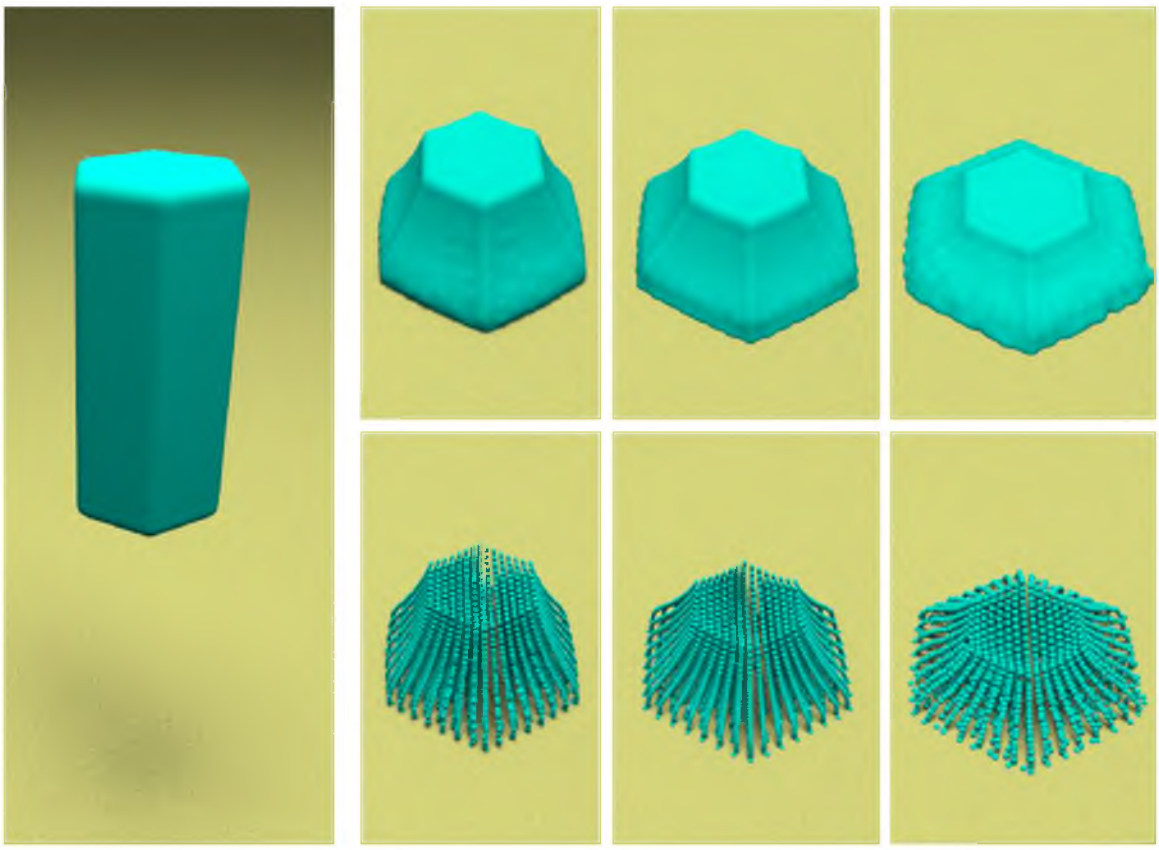

Figure 3.4. A plastic bar is dropped onto the ground. Without neighborhood updates (center left), the simulation becomes unstable after the object is significantly flattened. Our embedding allows neighborhood updates which improve stability (center right), but this also becomes unstable. Adding resampling (right) preserves stability through the entire scene.

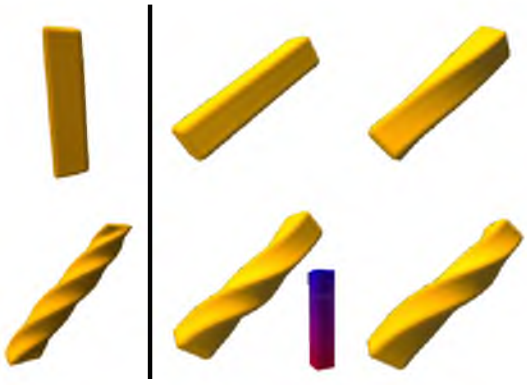

Figure 3.5. A bar is twisted and sheared (left), then released. The final world space configurations for (clockwise from top, left) elastic, slightly plastic, highly plastic, and varying plasticity materials. For the nonuniform bar, the plastic flowrate varies from high(red) to low(blue) along the bar.

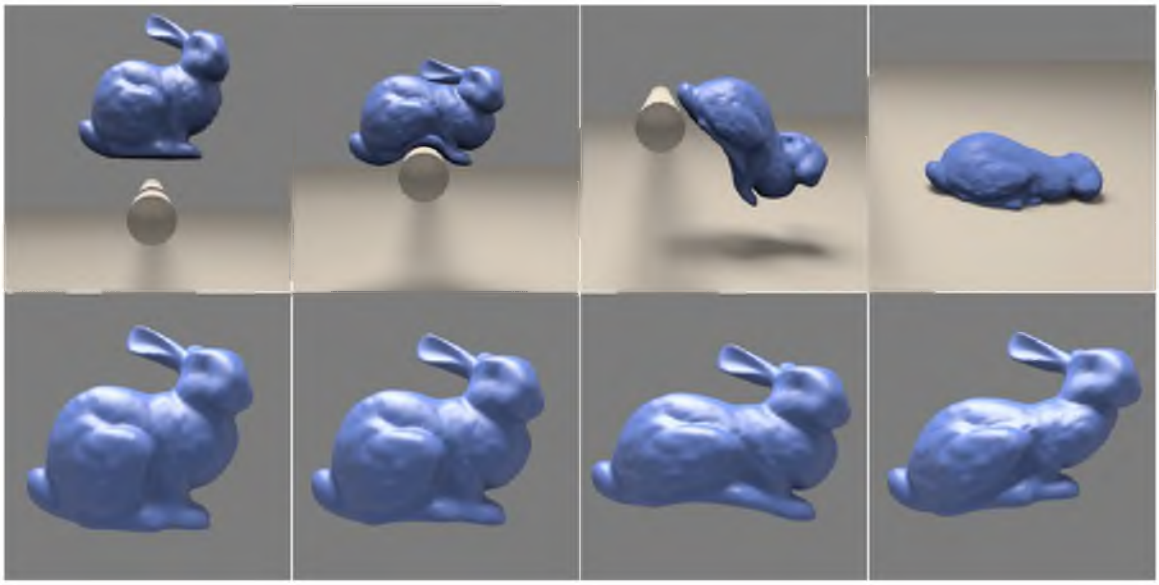

Figure 3.6. World space (above) and embedded space (below) of bunny dropped on a rigid bar.

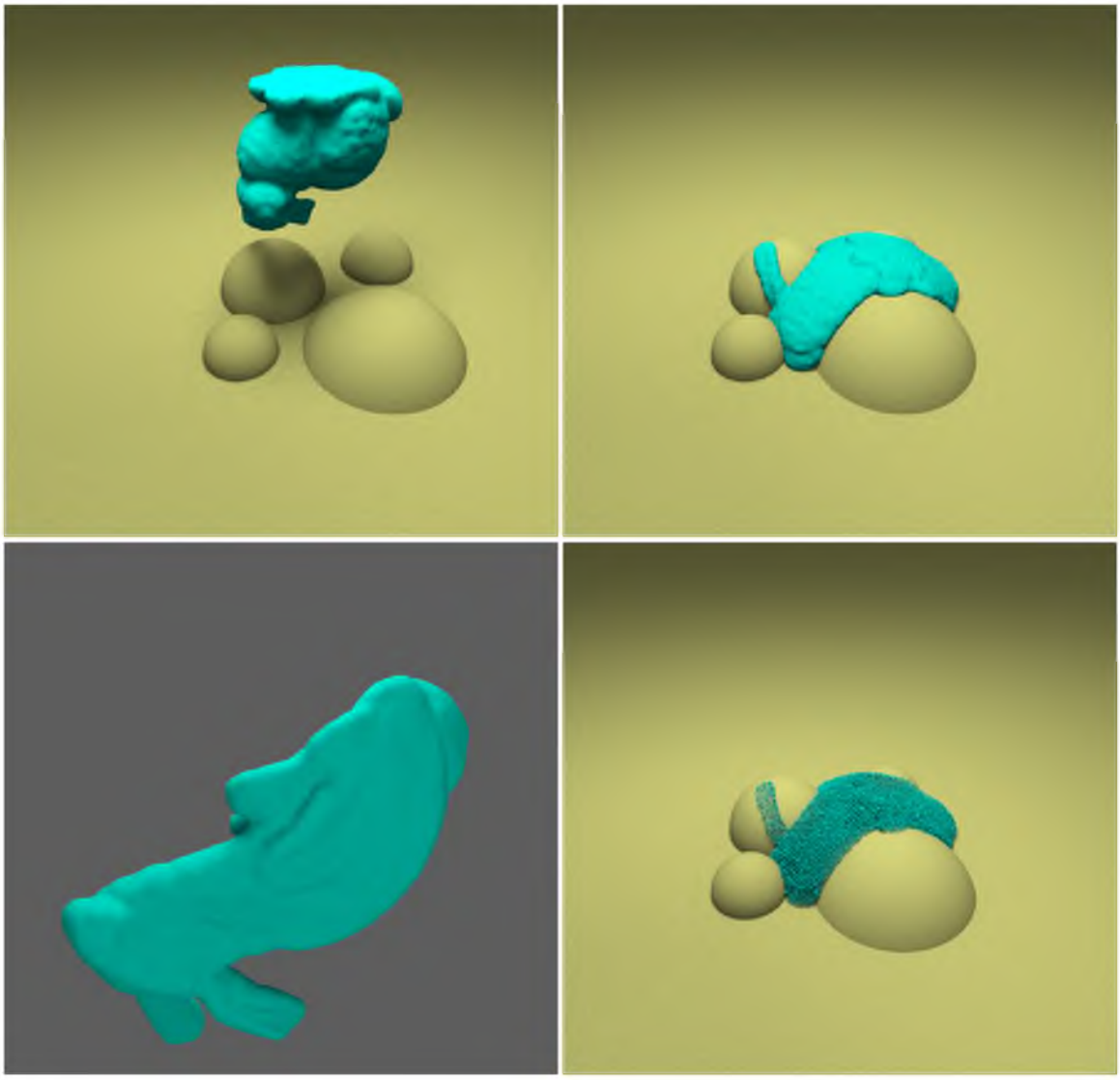

Figure 3.7. A bunny is dropped on a set of spheres. Clockwise from top left: Initial configuration, world (skinned), world(particles), and embedded spaces after impact.

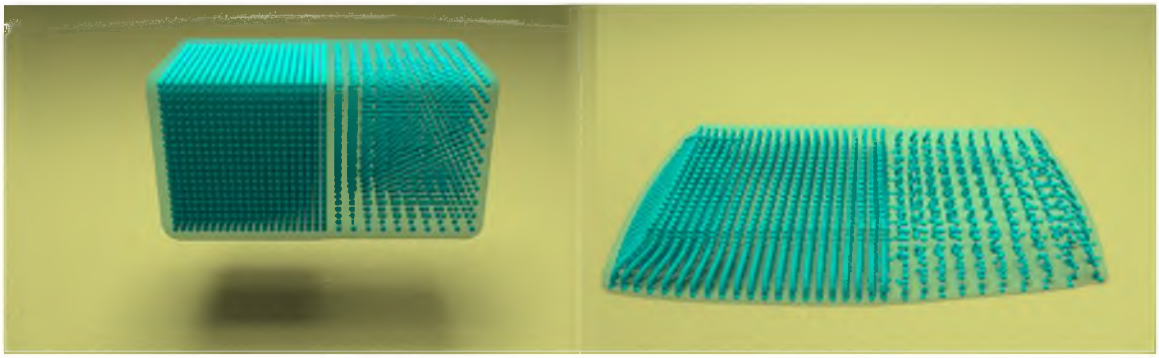

Figure 3.8. A plastic block with dramatically different sampling densities flows when dropped.

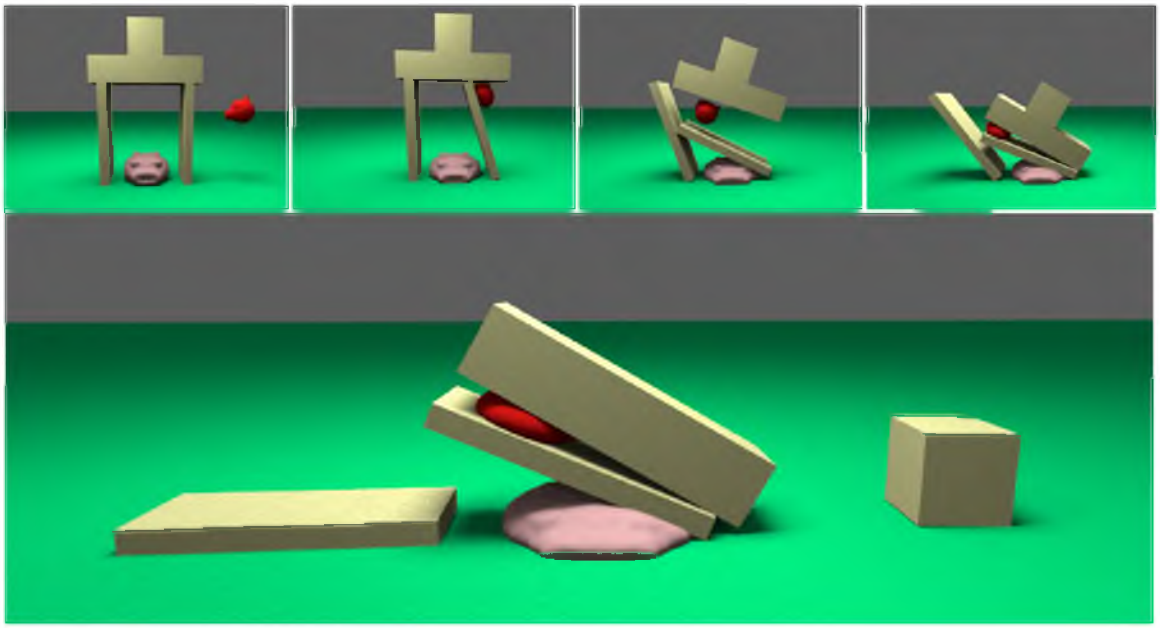

Figure 3.9. An "upset fowl" destroys a pig's house.

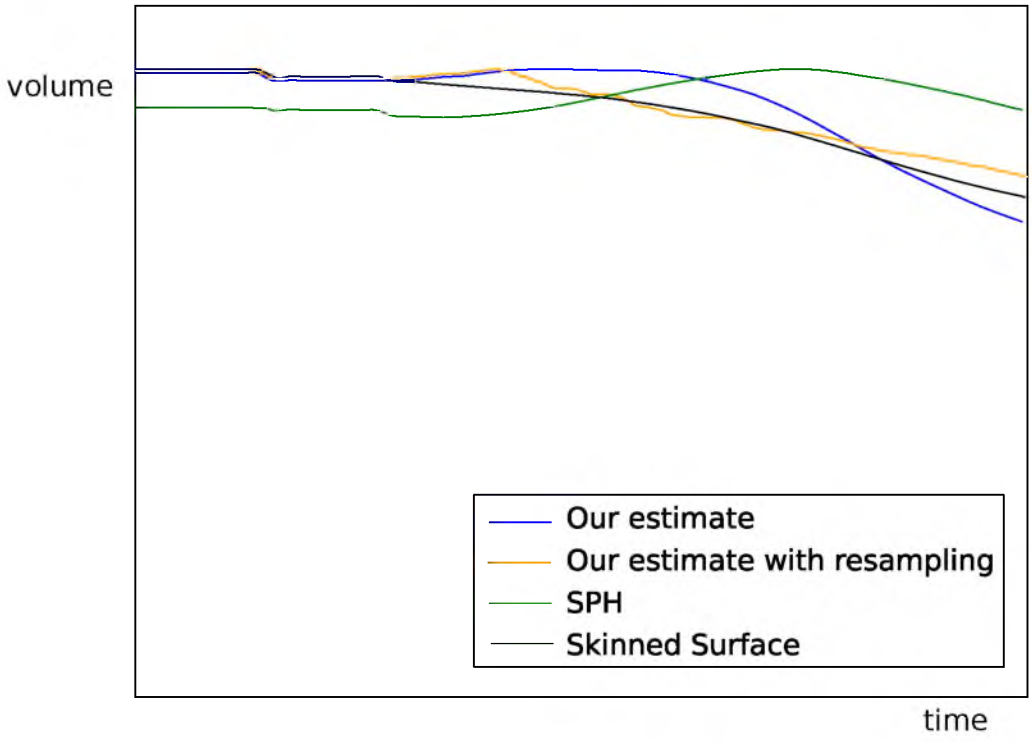

Figure 3.10. Estimated total volume computed using our new approach, SPH, and the skinned embedded space mesh for Figure 3.4.

Table 3.1. Timing results for pictured examples. Time is given in seconds to produce one 30 Hz frame of animation.

Example	#Particles	Δt	Sec/frame
Figure 3.2 linear	5,000	.001	2.36
Figure 3.2 nonlinear	5,000	.001	4.30
Figure 3.4	5,408	.001	1.2
Figure 3.7	36,675	.0004	278
Figure 3.8	10,470	.005	2.39
Figure 3.9	21,708	.0001	92

Table 3.2. Timing detail for Figure 3.4

Step	% computation time
Compute basis matrices	3.54
Viscosity solve	17.19
Compute forces	4.55
Explicit integration step	0.36
Embedding solve	41.70
Resampling	2.24
Neighborhood updates	22.91
Compute plastic offsets	3.73
Handle collisions	0.19

CHAPTER 4

DYNAMIC SPRITES: ARTISTIC AUTHORING OF INTERACTIVE ANIMATIONS

4.1 Introduction

Drawing pictures is one of the oldest and most fundamental forms of human communication. Ever since our ancestors began creating cave paintings in prehistoric times, humans have been making pictures to tell stories, explain ideas, and express emotions. Over the years, technical advances have provided artists with an ever expanding arsenal of tools and techniques for creating and editing images, and today, sophisticated software packages like Adobe Photoshop and GIMP include a variety of features that enable users to copy, paste, compose, morph, and otherwise manipulate their images. Unfortunately, while existing digital tools make it easier than ever to create high-quality *static* images, making drawings come "alive" through movement and interactive behaviors in cartoons, videos, or games is a very different and challenging task.

Turning a drawing into a dynamic, interactive entity typically involves several steps that require different types of expertise: a rigger defines articulation variables; an animator creates keyframes that specify how those variables change over time; and a programmer encodes the keyframed motions into behaviors. In some cases, simulation can provide a more automated alternative for generating motions and behaviors, but it often requires significant amounts of tuning to produce results that exhibit specific, desired characteristics. For professional film and game production, these steps can be distributed to separate teams of artists, riggers, animators, and programmers, which makes for a highly modular content creation pipeline. However, for more casual users, the gap in required skills and expertise between creating a drawing and making it come to life represents a significant barrier. This is one likely reason why it is much easier to find examples of high-quality static drawings (e.g., in online image repositories) than compelling dynamic content.

Sprite sheets, collections of static 2D drawings that depict representative poses for an object, like the example shown in Figure 4.1, offer a more cohesive, drawing-centric approach to creating dynamic images.

Producing motion with a sprite sheet involves cycling through drawings of the object. Thus, artists can change both the appearance and dynamic behavior of an object using traditional drawing tools by creating or editing various poses. Unfortunately, sprite sheets require many drawings to produce smooth animations since even small deformations to a pose require an entirely new drawing. Furthermore, even if these in-between poses can be generated automatically, sprite sheets alone encode neither the timing information that is critical for producing high-quality motions, nor the logic that defines how an object should behave in response to external stimuli. As a result, sprite sheets still require a significant amount of work to create and use. In contrast, we use physics to automatically provide timing and achieve generalization outside the hand-drawn examples.

In this work, we present an approach for transforming static drawings into dynamic sprites. To produce a dynamic sprite, the artist creates a pose manifold by drawing an object, deforming it into example poses, and specifying sets of these example poses that can be interpolated as the object moves. Our system uses example-based physical simulation to automatically move the object through this pose manifold based on forces applied from the external environment or user commands. In this way, we allow the artist to create dynamic objects and characters without specifying many in-between frames, adjusting animation curves, or writing code that defines object behavior.

The key feature of our approach is that it provides a set of explicit artistic controls over various characteristics of dynamic sprites. First, the example poses themselves give artists significant control over the appearance of the pose manifold.

However, combining several example poses at once can lead to a "muddy" manifold where interesting features of the individual examples get lost in the blended pose. Thus, we allow the artist to explicitly construct a simplicial complex (mostly line segments with the occasional triangle) over the set of example poses. Furthermore, since external physical forces alone may not induce sufficient motion within the pose manifold, our method provides several additional knobs for controlling the manner in which objects transition between the poses. For example, artists can tell the system to favor particular poses for specific object states, such as a stretched out pose when an object has high velocity, or a neutral pose that represents the object at equilibrium. The artist can also adjust how much energy an object retains after interactions (e.g. the bounciness of a ball sprite). Modifying these parameters allows artists to control the look and feel of motions and behaviors in a more direct, intuitive manner than fine-tuning physical properties like the elasticity or density of deformable objects.

For more complicated behaviors, arbitrary controllers can be used to traverse the pose manifold. Compared to controllers used in traditional physics-based character animation, our controllers are simpler because the sprites themselves can maintain important features of the motion, such as balance, and otherwise "bend" the laws of physics when desired.

We used our approach to generate a variety of sprites that exhibit different types of dynamic behavior, including a cartoony ball, bouncy characters, lively construction materials, and articulated ragdolls, either passively animated or controlled by a finite state machine. We also created three games that combine multiple sprites into interactive environments. Our results demonstrate how our approach facilitates the creation of dynamic objects from static drawings and gives artists intuitive and powerful control over not only the pose manifold, but also how the pose manifold is navigated.

4.2 Method

Our approach proceeds in two phases: an authoring phase, in which an artist designs a low-dimensional deformation rig, shown in Figure 4.2, then uses this rig to create example poses and defines a simplicial complex connecting these poses, as shown in Figure 4.3. Then, in the simulation phase, our system uses example-based simulation to create a dynamic, interactive animation like the ones shown in Figure 4.4. We note that though the details of our example-based simulation differ from previous work, our primary technical contribution is in how we provide explicit artistic control over the pose manifold and how it is navigated.

4.2.1 Authoring Phase

As the first step to creating stylized interactive animations, the artist specifies a low degree of freedom rig using the method of Jacobson and colleagues [2012]. Specifically, the artist places a small number (5-10) of bones and/or pins on the input shape. By manipulating this simple rig the artist creates a set of example poses that will guide the simulation. This rig also determines how the example poses will be interpolated. The artist additionally defines a simplicial complex over the example shapes that determines which shapes may be blended. Then the artist loads the simplicial complex into our example-based simulation and interactively tunes how the pose manifold is navigated by choosing what optional filters to apply and with what strengths.

4.2.2 Example-based Simulation

Our runtime environment builds on recent work that applies example-based simulation to shape matching/position-based dynamics [Schumacher et al. 2012; Koyama et al. 2012].

Deformable objects are modeled as a set of particles, $\mathbf{p}_i \in \mathcal{P}$ with positions, $\mathbf{x}_i$, and velocities, $\mathbf{v}_i$. The neighborhood of a particle $\mathcal{N}(p_i)$ is the set of particles in the k-ring (typically, $k = 3$) of $\mathbf{p}_i$ in a precomputed triangulation of the particle set. At its most basic level, our method applies a series of filters to the particle's positions and velocities to satisfy the goals of the artist. Our approach can also be cast in the prediction/correction framework: we *predict* particle positions with a forward Euler integrator and then *correct* them to achieve various goals; including pulling toward the pose manifold, removing interpenetration, and artistic goals, such as setting the global orientation. See Algorithm 2 for a summary of the simulation timestep. We now describe these operations in more detail.

Algorithm 2 Simulation Timestep

Apply body forces (e.g. gravity): $v \mathrel{+}= f/m * dt$
Forward-euler position update: $x \mathrel{+}= v * dt$
Compute desired rest shape
Interleave and Iterate
    Local-neighborhood shape match
    Global shape match
    Resolve collisions

Compute velocity
Global momentum adjustment
Correct global orientation

4.2.2.1 Compute Desired Rest Shape

Our runtime environment takes as input the artist-authored *pose manifold* described by a set of example poses, a low degree of freedom animation rig, a simplicial complex that describes which poses can be blended, and any other rules (such as an equilibrium pose) that guide navigation on the manifold. Then, during each simulation step we must select a pose on this pose manifold that will be used as a rest state for shape matching.

To interpolate poses within a simplex, we first factor each handle transform into translational, scale/shear, and rotational components using a polar decomposition. Translation and scale/shear terms are interpolated linearly, while rotations are combined using spherical linear interpolation. The final pose is generated by applying linear blend skinning to the interpolated transforms. Specifically, to blend two example poses with weights α and β we would have,

$$\mathbf{e}_i = \sum_{j=1}^{m} w_j(\mathbf{r}_i)\,\mathrm{Slerp}(\alpha\mathbf{R}_1, \beta\mathbf{R}_2)(\alpha\mathbf{S}_1 + \beta\mathbf{S}_2)\mathbf{r}_i + \alpha\mathbf{t}_1 + \beta\mathbf{t_2}, \tag{4.1}$$

where $\mathbf{e}_i$ is the particle's position in the pose selected from the pose manifold, $\mathbf{r}_i$ is the particle's position in the initial rest configuration where the weights were computed, $w_j(\cdot)$ gives the weight of the j^{th} handle at the specified position, and $\mathbf{R}$, $\mathbf{S}$, and $\mathbf{t}$ are the rotation, scale/sheer, and translations computed by the polar decomposition. Generalization to blending more poses is straightforward.

It remains to describe how we compute the weights (α and β above) we will use to combine the example poses. The straightforward solution is to simply project the current simulated shape onto the pose manifold. When external forces deform an object toward an example pose, this straightforward manifold projection generates animations that smoothly and plausibly transition between the user-provided examples. However, projection alone is not adequate for forces orthogonal to the pose manifold, or when other artistic control is desired. Thus, we have included additional control over how the simulation navigates the pose manifold, resulting in richer behavior. The weights are computed by performing the following operations:

1. Project current shape onto pose manifold
2. Adjust the equilibrium pose
3. Attract toward equilibrium pose
4. Apply energy adjustment
5. Apply velocity-based adjustment

Note that the first two elements have been incorporated into previous example-based simulation approaches, while the last two are critical to achieving our results.

Project current shape onto pose manifold: The goal of this step is to find interpolation weights in the simplicial complex, such that the skinned shape matches the current shape as closely as possible. To account for rotations we additionally compute a global, best-fit rotation. We alternate between solving for optimal weights with this rotation held fixed, and solving for the optimal rotation with weights held fixed. To compute optimal weights, we define our cost function as

$$\sum_{i,j} ||\mathbf{R}_g\mathbf{e}_{ij} - \mathbf{x}_{ij}||^2 \tag{4.2}$$

where i and j are connected particles, $\mathbf{R}_g$ is the current global rotation, $\mathbf{x}_{ij}$ is the vector between particles i and j in world space, and $\mathbf{e}_{ij}$ is the vector between i and j computed

via linear blend skinning with the current interpolation weights (see Equation 4.1). This projection yields barycentric coordinates, $\mathbf{m_p}$, in the simplicial complex that defines the manifold.

As noted by Jacobson and colleagues [2012], positions of nearby particles computed by linear blend skinning are highly correlated, so we adopt their "rotation clusters" optimization. Instead of summing over all particles i and j, we compute a smaller set of representative particles using k-means clustering and sum over them. We use a simple Newton solver to optimize the objective, using automatic differentiation to compute the gradient and Hessian [Fike and Alonso 2011]. To compute the optimal global rotation, we use the method of Sorkine and Alexa [2007].

Adjusting the equilibrium pose: As described thus far, our example-based framework has no notion of a *rest pose*—all poses in the pose manifold generate zero elastic energy. To address this limitation, we introduce the notion of an *equilibrium pose*, $\mathbf{q}$, in the example manifold. High-level planning and control strategies are incorporated into our method by allowing artists explicit control over how this equilibrium pose is chosen. In our prototype, complex behaviors are implemented using a 2-level finite state machine. The high-level state machine switches between character behaviors (walking, jumping, falling, etc.), and is controlled by user input and dynamic properties of the simulation, such as whether or not the character is standing on the ground. The lower-level state machine controls individual behaviors by adjusting the location of the equilibrium pose in the manifold, such as transitioning through the poses in a walk cycle.

Attract toward equilibrium pose: In the spirit of position-based dynamics, we use a first-order spring to attract toward the equilibrium pose:

$$\mathbf{m}_{eq} = \mathbf{m}_e + k_{eq} \left(\mathbf{q} - \mathbf{m}_e \right), \tag{4.3}$$

where $\mathbf{m}_{eq}$ is the pose after attracting to the equilibrium pose, k_{eq} is a stiffness, $\mathbf{m_e}$ the barycentric coordinates after energy adjustment, and $\mathbf{q}$ the barycentric coordinates of the equilibrium pose. By adjusting the stiffness of the manifold spring, the artist can control how closely the specified trajectory is followed, and how the motion is influenced by other other aspects of the simulation (e.g. energy adjustment).

Apply energy adjustment: Often, an artist will want the simulation to closely match the pose manifold, requiring very high material stiffness. In these cases, any energy from deformations orthogonal to the pose manifold is lost. To combat this, we apply some of this

lost energy in the pose manifold. Position-based dynamics does not have a explicit notion of energy; however, a good proxy for kinetic energy is

$$e = \sum_{i \in \mathcal{P}} m_i (\Delta \mathbf{x})^2, \tag{4.4}$$

where m_i is the mass of p_i and $\Delta \mathbf{x}$ is the change in a particle's position. We compute this energy when particles interact with other objects (e.g. during collisions). Our simulator then pushes the interpolation weights by an amount proportional to this energy,

$$\mathbf{m}_e = \mathbf{m}_p + k_e \mathbf{d}_e e, \tag{4.5}$$

where $\mathbf{m}_e$ is the pose after applying energy adjustment, $\mathbf{m}_p$ is the initial projection onto the manifold, k_e is a stiffness, and $\mathbf{d}_e$ is the direction of energy offset. This direction can be variably chosen as the direction away from the equilibrium ($\mathbf{m}_p - \mathbf{m}_{eq}$), the velocity in the manifold, or an explicitly specified direction. To avoid unwanted oscillations, we ignore energy contributions below a user-defined threshold.

The result is in-manifold deformation for shapes, even when experiencing orthogonal interactions, and a reduction of out-of-manifold deformation. Transferring energy normal to the manifold to a tangent direction may seem unphysical—*it is*. However, in practice we have found that this approach does an excellent job of preserving the artist's intent, expressed through the example shapes, as unphysical as this intent may be.

Apply velocity-based adjustment: In our framework, it is straightforward to use any information from the simulation state to guide navigation of the manifold. For example, in order to add cartoon-inspired stretch to a fast moving object, we attract interpolation weights to a manifold vertex corresponding to the stretched pose, $\mathbf{a}$, with a strength proportional to the object's speed, s.

$$\mathbf{m}_v = \mathbf{m}_{eq} + k_v s \left(\mathbf{a} - \mathbf{m}_{eq} \right), \tag{4.6}$$

where $\mathbf{m}_v$ is the pose after applying velocity adjustment, $\mathbf{m}_{eq}$ is the pose after being attracted to the equilibrium, k_v is a stiffness, and $\mathbf{a}$ is the pose being attracted to (e.g. the stretched pose).

4.2.2.2 Shape Matching

Once we compute the current rest pose of the object using linear blend skinning with appropriate interpolation weights, we are ready to compute dynamics. While any elastic simulation method could be used, we use shape matching [Müller et al. 2005] for its efficiency

and implementation simplicity. Furthermore, because shape-matching models elasticity with simple first-order dynamics, it fits in well with our framework of filtering of positions and velocities, allowing a greater degree of artistic control.

While the initial shape matching work [Müller et al. 2005] supported only global shape matching, more recently Rivers and James [2007] introduced a hierarchical approach. We take a middle ground and interleave global shape matching passes, which quickly correct any out-of-manifold deformation, and passes over local neighborhoods, which allow elastic deformations not described by example poses. For completeness, we briefly describe the shape matching approach. Given corresponding points in world space, $\mathbf{x}_i$, and in our example pose, $\mathbf{e}_i$, we solve for translations, $\mathbf{t}_x$ and $\mathbf{t}_e$, and rotation, $\mathbf{R}$, that minimize

$$\sum_i m_i \left(\mathbf{R}(\mathbf{e}_i - \mathbf{t}_e) - (\mathbf{x}_i - \mathbf{t}_x)\right)^2. \tag{4.7}$$

The translations correspond to the center of mass for each set of particles and the rotation is found by a polar decomposition. We can then compute goal positions, $\mathbf{g}_i$

$$\mathbf{g}_i = \mathbf{R}\left(\mathbf{e}_i - \mathbf{t}_e\right) + \mathbf{t}_x. \tag{4.8}$$

We include all particles in the global shape matches. For the local-neighborhood passes, we iterate over all the particles performing the shape match using only the particles in the local neighborhood, $\mathcal{N}(p_i)$, that is,

$$\sum_{j \in \mathcal{N}(p_i)} m_j \left(\mathbf{R}(\mathbf{e}_j - \mathbf{t}_e) - (\mathbf{x}_j - \mathbf{t}_x)\right)^2. \tag{4.9}$$

We then compute a particle's goal position by averaging over all neighborhoods that contain it.

$$\mathbf{g}_i = \frac{\sum_{j|i \in \mathcal{N}(p_j)} \mathbf{g}_{ij}}{\sum_{j|i \in \mathcal{N}(p_j)} 1}, \tag{4.10}$$

where $\mathbf{g_{ij}}$ is p_i's goal position when shape matching using $\mathcal{N}(p_j)$.

Additionally, we allow the artist to decompose an image into disjoint layers. To propagate constraints between layers, we add an additional constraint on a set of "pin" particles which join two layers. At initialization, for each pin, we find the triangle on the connecting layer in which it is contained. Then, we compute the barycentric coordinates of the pin with respect to its containing triangle. To enforce the constraint, we compute the world position of the stored barycentric coordinates with the current triangle vertex positions, and pull the pin particle toward it.

Collisions are handled by projecting overlapping particles out of objects, using the underlying triangle mesh to detect and resolve collisions.

4.2.2.3 Global Momentum Adjustment

Position-based dynamics has difficulty producing highly elastic, "bouncy" collisions. The inclusion of squashed poses in the pose manifold exacerbates this issue as they appear to be rest poses to the shape matching. However, such collisions are essential to lively, cartoon-style animation. To address this limitation, we add momentum directly to the system after collisions with the ground. Specifically, for each object we store the momentum we wish to add, $\mathbf{p}$, which is updated each timestep with a contribution from each colliding particle.

$$\mathbf{p} \leftarrow \mathbf{p} + m_i k_m \left(\mathbf{x}_{pro} - \mathbf{x}_{pen}\right) \tag{4.11}$$

where m_i is the particle mass, k_m is a scale, $\mathbf{x}_{pro}$ is the particle's (projected) position after resolving the collision, and $\mathbf{x}_{pen}$ is the penetrating particle position. Over time we add this momentum to the system, for each particle,

$$\mathbf{v}_i \leftarrow \mathbf{v}_i + m_i r \mathbf{p} / m_o^2, \tag{4.12}$$

where $\mathbf{v}_i$ is the particle's velocity, m_i its mass, r the rate at which we add the momentum, and m_o is the total mass of the object. Finally, we update $\mathbf{p}$

$$\mathbf{p} \leftarrow (1 - r)\mathbf{p} \tag{4.13}$$

This approach has two important features. First, by computing $\mathbf{p}$ per object rather than per particle, we avoid spatial discontinuities. Second, by adding momentum over time, we allow the collisions to occur over a finite time period, allowing the object to deform while it is on the ground, before jumping back into the air.

4.2.2.4 Orientation Correction

In addition to the shape control provided by example-based shape matching, artists may desire control over other aspects of the object's motion. For example, an artist may want to keep a character upright or aligned with its velocity. Providing such control is straightforward in our framework and requires only applying another filter to the set of particles. For the case of rotation control, we apply a first-order spring to the global orientation of the object about its center of mass,

$$\theta_a = \theta_b + k_{oc} \left(\theta_g - \theta_b\right), \tag{4.14}$$

where θ_b and θ_a are the global orientation before and after orientation control, respectively, k_{oc} is a stiffness, and θ_g is the goal orientation. This goal angle can be a particular value, a

scripted trajectory, or other user-defined criteria, such as the current velocity direction. The ordering of filters is important. For example, applying this filter before updating particle velocities leads to the computed velocities having an unwanted "spin" component.

4.2.2.5 Flipping Sprites

In most sprite-based applications, characters can turn around simply by reflecting the sprite about a vertical axis. Incorporating such a discontinuous change in our physics-based simulator requires care to maintain plausible elastic simulation. Since the velocity at the next timestep is computed as

$$\mathbf{v}^{n+1} = \frac{\mathbf{p}^{n+1} - \mathbf{p}^{n}}{dt} \tag{4.15}$$

after flipping the sprite, we adjust the previous positions, $\mathbf{p}^n$ so that velocities are not changed by flipping. Artistically, this gives a sense of weight to objects as they change direction; physically, it preserves linear momentum.

4.3 Results

To illustrate our authoring process we consider the classic animation task of creating a bouncing ball with squash and stretch. Figure 4.2 shows the simple rig we use to create the example poses in Figure 4.3. Figure 4.3 also visualizes the three line segments that make up the example manifold, which allows interpolation between the undeformed pose and either the squashed or stretched poses. In this example, the undeformed pose is set as the equilibrium pose. In addition, we use energy adjustment to move the selected example toward the squashed pose, which is otherwise largely ignored. Velocity-based adjustment pushes the selected example toward the stretched pose when the ball is moving quickly, and global orientation control aligns the pose with the velocity direction. Finally, global momentum adjustment allows the ball to bounce, appearing to "come alive" and move of its own volition. Figure 4.4 shows a variety of behaviors that can be achieved by changing the parameters.

We have also incorporated dynamic sprites into several simple games. In this context, the dynamic sprites enhance both visual complexity as well as gameplay compared to static objects or previous example-based approaches.

In our first game, players attempt to navigate a character vertically towards a finish line by jumping on a sequence of platforms. The player can control the rest pose of the character and apply left and right forces while in the air. With dynamic sprites, it is easy to create a variety of platform types, as shown in Figure 4.5, that look and behave differently based on the underlying examples and settings for the artistic controls.

For example, the brick platforms are mainly rigid and provide little vertical boost after impact, while the I-beams bend elastically when the player lands. The rope platforms are softer than the I-beams and do not spring back to their original shape as quickly.

In the second game, players position I-beams, catapults, and other objects to direct a passive object (e.g. a bouncy ball) that is dropped from above towards a target. Our dynamic sprites can be used to animate a variety of game objects ranging from sturdy brick obstacles, to an energetic catapult and launching pad. A sample level is shown in Figure 4.6.

In our third game, ragdoll figures are fired from a cannon toward a target while various boxes and I-beams serve as obstacles. Sample frames are shown in Figure 4.7.

We represent all game elements as dynamic sprites with different behaviors: the cannon contracts on firing; the character is drawn to an elongated, "superman" pose when moving quickly but assumes a fetal position upon impact; the I-beams wiggle elastically due to bent example poses; and energy adjustment draws the boxes to a variety of poses that are largely orthogonal to the external forces. As can be seen in the figure and accompanying video, our dynamic sprites generate richer deformations than shape matching or standard example-based physics, where the red poses are not activated. Unless a large number of particles are perturbed toward an example pose, the manifold projection does not move the current rest pose significantly through the manifold; the bulk of the material, which is undeformed, has lowest cost for the current rest pose, especially for the stiff materials we desire. The "rotation cluster" approximation we use for efficiency exacerbates this, since only the representative particles are considered during our manifold projection step. However, even when considering all particles during the projection step, it is difficult to trigger example poses through local collisions when using stiff materials.

In the final game, our ragdoll character is now actively controlled using a 2-level finite state machine (see Figures 4.8 and 4.9). The walking behavior state machine moves the equilibrium pose through 6 keyframe poses on a manifold with a loop topology. Unlike in traditional physics-based controllers, we did not need to consider balance and robustness of the character's locomotion; global orientation control keeps the character upright.

When the user wants to jump, first the character must crouch to prepare, and then can spring upward into a standing pose, similarly to the snake character in the platformer game. The character's inflatable jacket can also be used to shoot him further up in the air, and allow him to float down slowly. The floating behavior slowly blends from the inflated pose to the neutral pose. The position in the pose manifold also determines the magnitude of

a drag force on the man as he slowly floats to the ground. Figure 4.10 shows the ragdoll using a bouncy I-beam to help him float across a dangerous ball pit.

Our results exhibit many behaviors that might be considered artifacts in other physics-based approaches. However, many of these are both desirable, and easily controllable by an artist. For example, the "jiggliness" and excitability of the I-Beams in the cannon game are easily adjustable from passive and stiff, to energetic and oscillatory, as seen in the accompanying video. In the same way that cartoon drawings do not accurately reflect their real-world inspiration, our physics-inspired (nonphysical) behavior is a stylized exaggeration of real-world motion.

Our method does exhibit some behaviors that we consider to be artifacts. Some of the blended poses seem unnatural or undesirable. This can be alleviated to some extent by including more in-between poses in the manifold. Contact handling sometimes introduces oscillations near collisions, especially during resting contact. Finally, the global rotations applied by our orientation control can be visually jarring. A first-order spring-based control method may not be adequate to achieve smooth, subtle rotation control.

Timing information is shown in Table 4.1.

4.4 Conclusion

Our current system allows artists to turn sketches of example poses into dynamic, physical, reactive objects and actively controlled characters. We see several promising directions for future work. In our current pipeline, objects are posed using a warping system. This is not a requirement of the method; adding support for automatic registration (e.g. [Sýkora et al. 2009]) would allow us to use traditional sprite sheets as input. While we believe that our parameters are intuitive, it might still be interesting to investigate automatic tuning – e.g. changing momentum-preservation based on an artist-sketched bounce. Similarly, it may be difficult for users of our system to understand if a behavior they are seeing results from badly-tuned parameters or the need for more example poses; this is also a question that we may be able to answer algorithmically. Because we focus on sketched input, our technique works only in 2D; however, we believe that the pose manifold concept and the control methods we propose should generalize to 3D.

Overall, we find that our system sits at an interesting point in the design space of methods for creating dynamic content. By directly connecting artist-created poses with physical properties, dynamic sprites enable artists to create more interesting and detailed physics-based characters and objects with less effort than either traditional sprite sheets

(which sacrifice physical realism) or rigging and animation systems (which require several different areas of expertise).

Figure 4.1. Traditional sprite sheets capture all the poses a character or object can assume in a game. (Example from "Age of Umpires"; http://hockey.spacebar.org/. Copyright Tom Murphy VII, used with permission.)

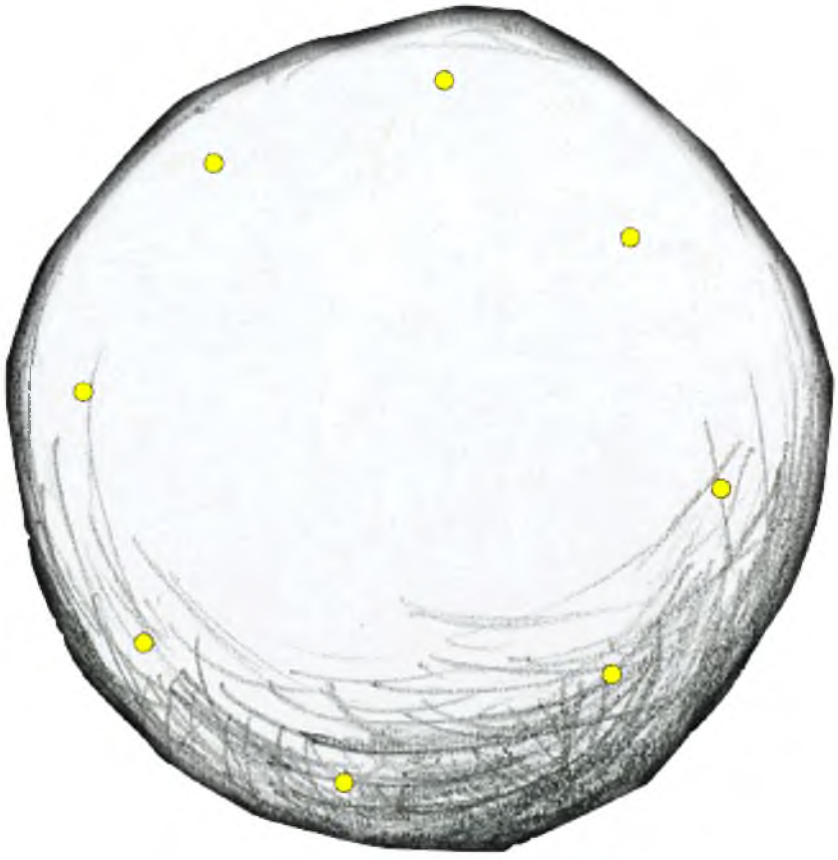

Figure 4.2. The rig used to generate poses of the cartoon ball. The yellow dots are control handles.

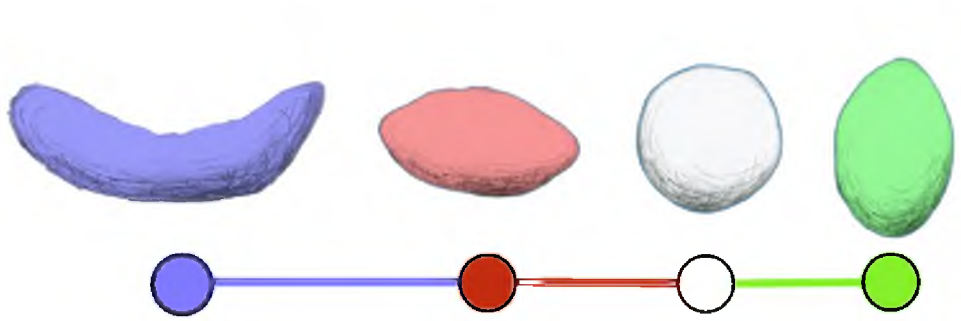

Figure 4.3. The example poses and the simplicial complex used to create a stylized bouncing ball.

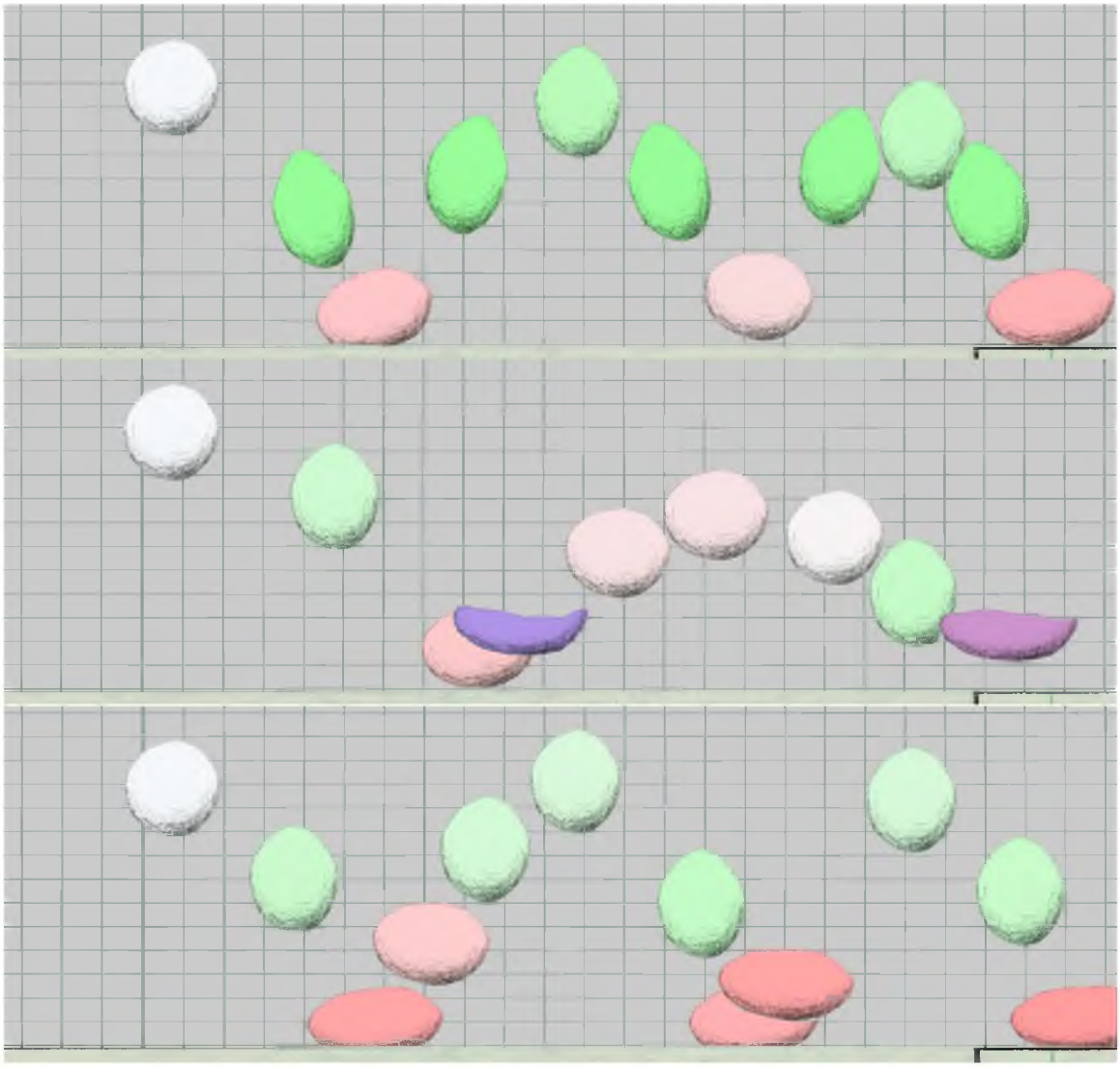

Figure 4.4. Three stylized behaviors generated by our system.

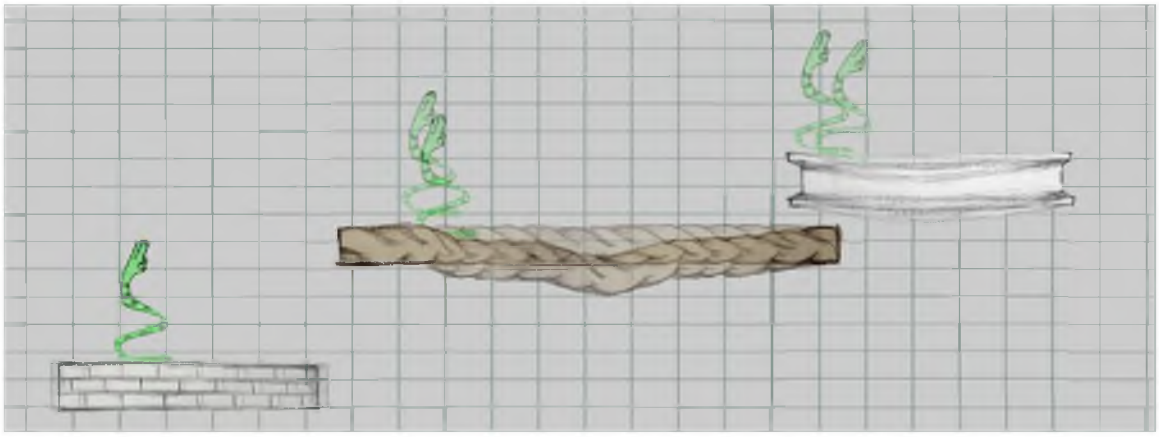

Figure 4.5. These dynamic sprites platforms cover a broad range of behaviors.

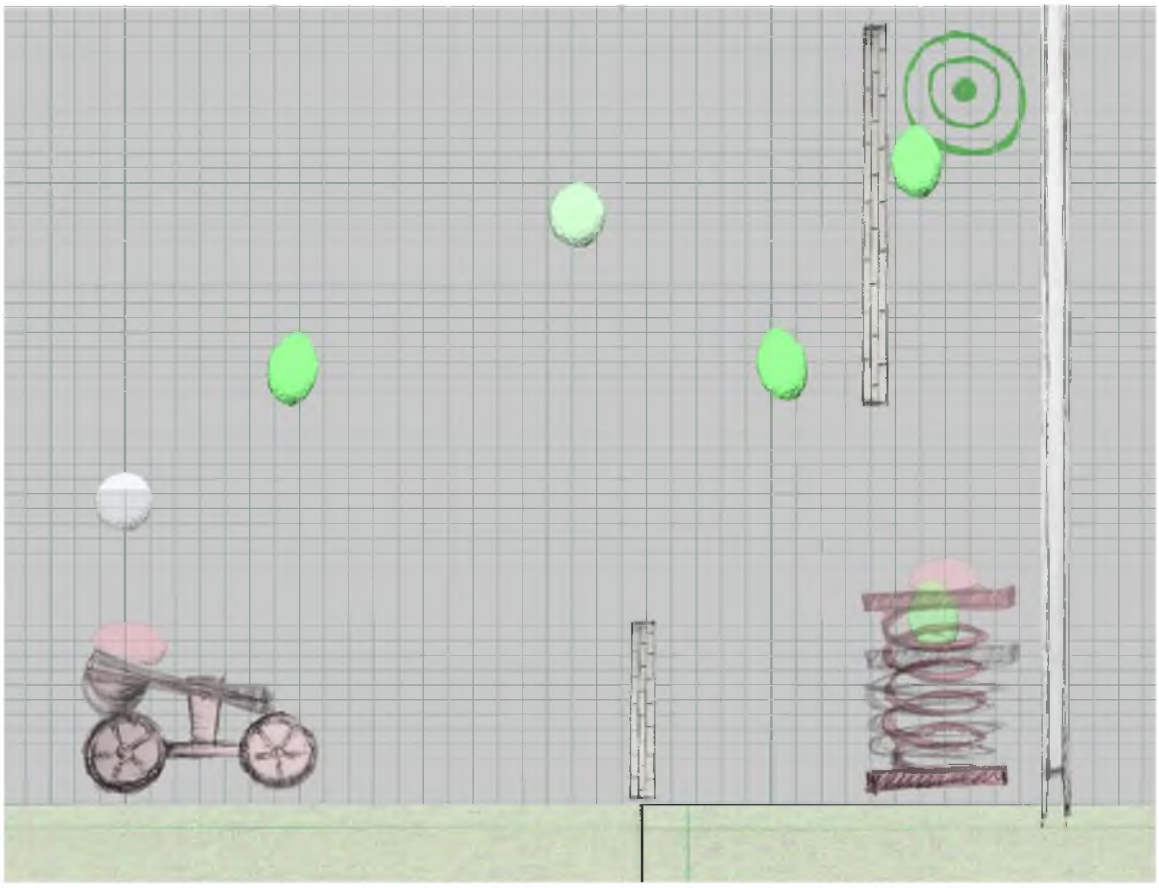

Figure 4.6. The user places the catapult and springy platform to help the ball reach the goal.

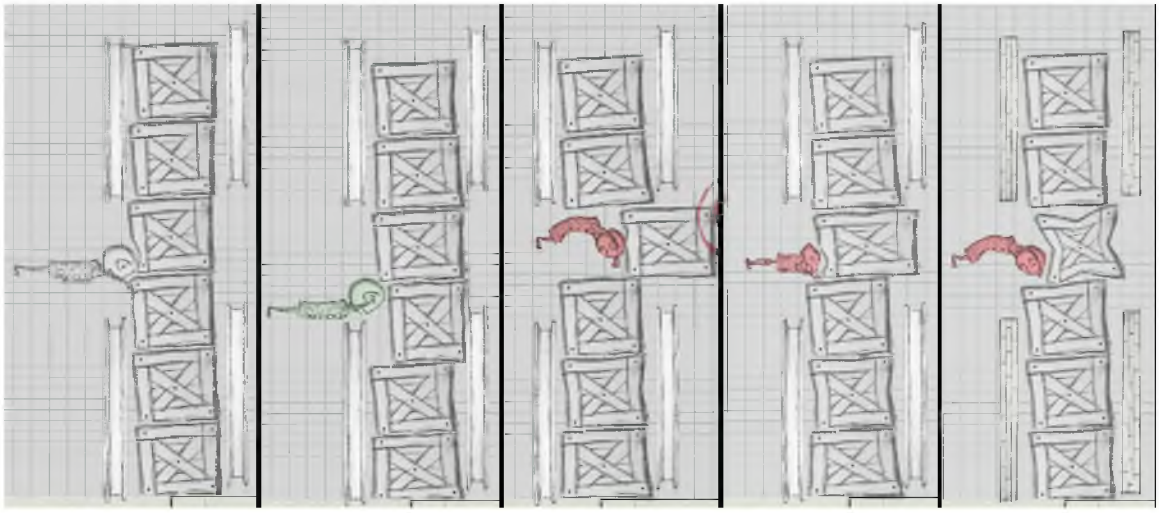

Figure 4.7. From left to right: Shape matching only, shape matching with examples, 3 different dynamic sprites.

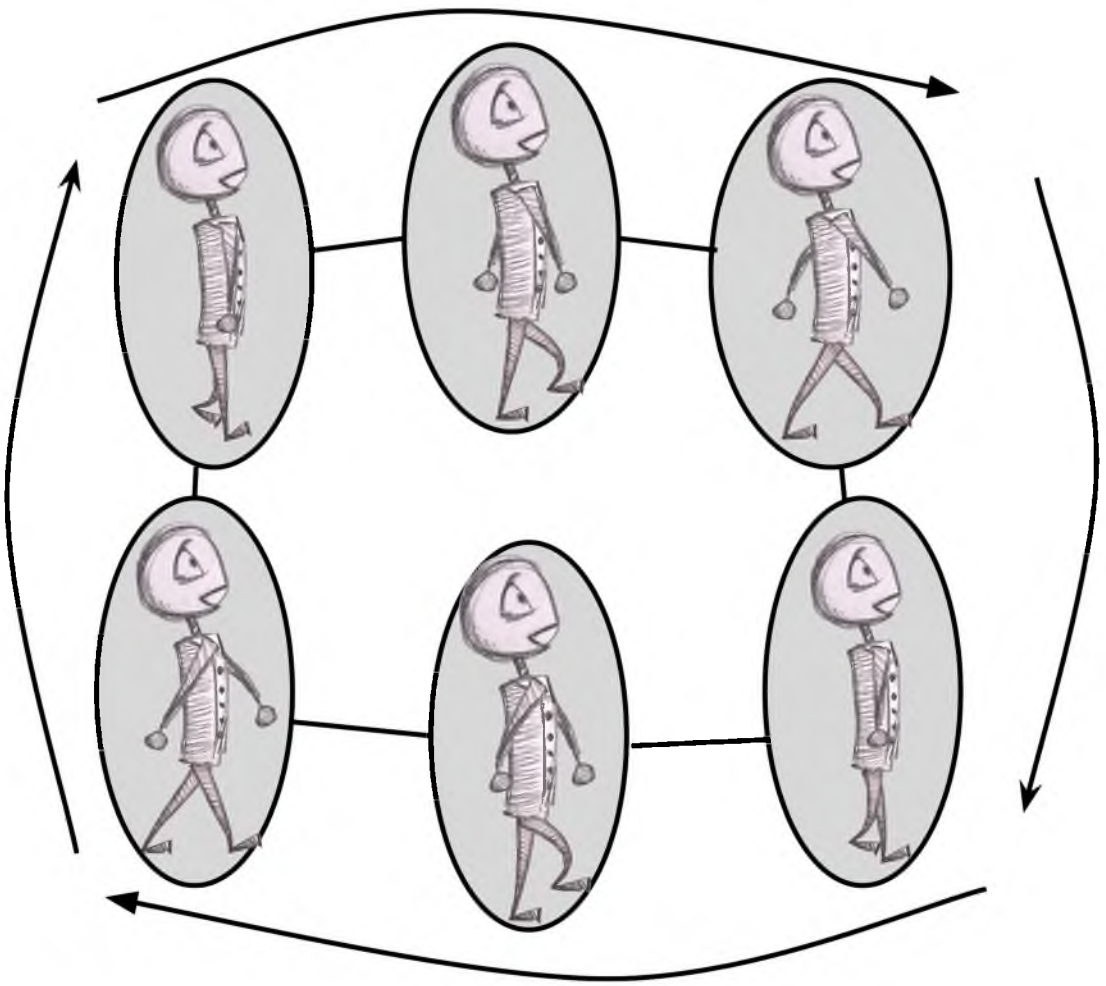

Figure 4.8. With 3 example poses and their reflections, a simple manifold, and a finite state machine, our system generates a lively, stylized walking behavior.

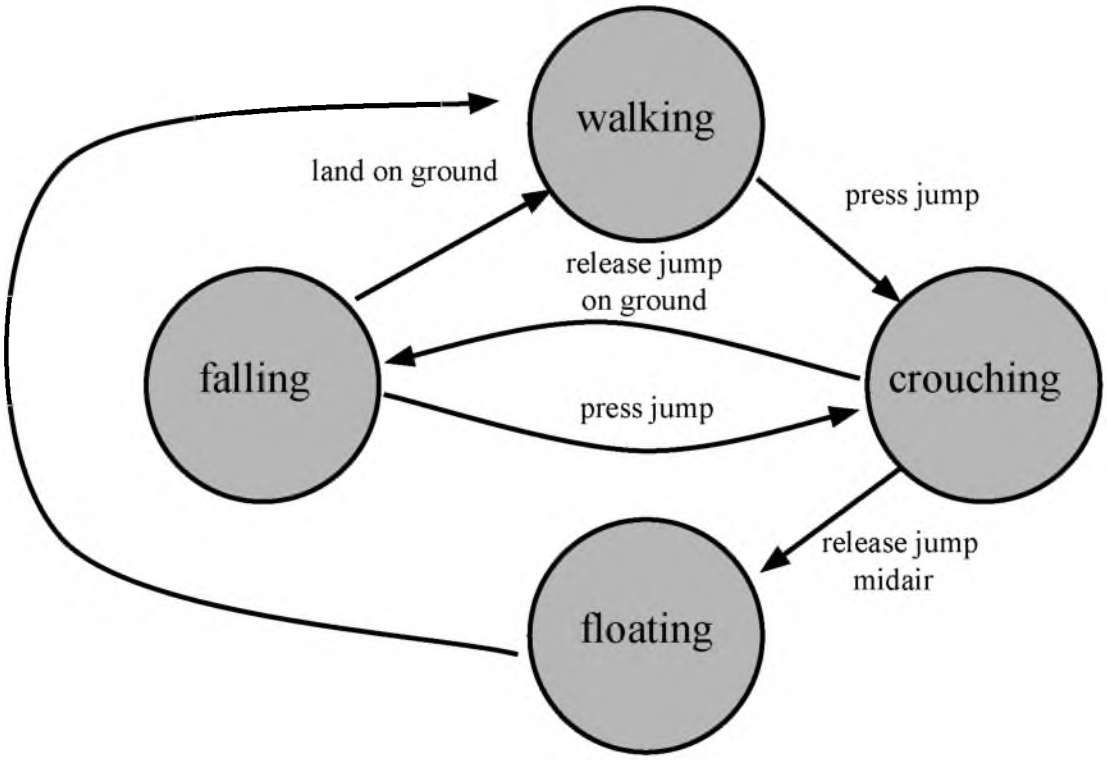

Figure 4.9. A simple state machine transitions between behaviors based on user input, the ragdoll's state.

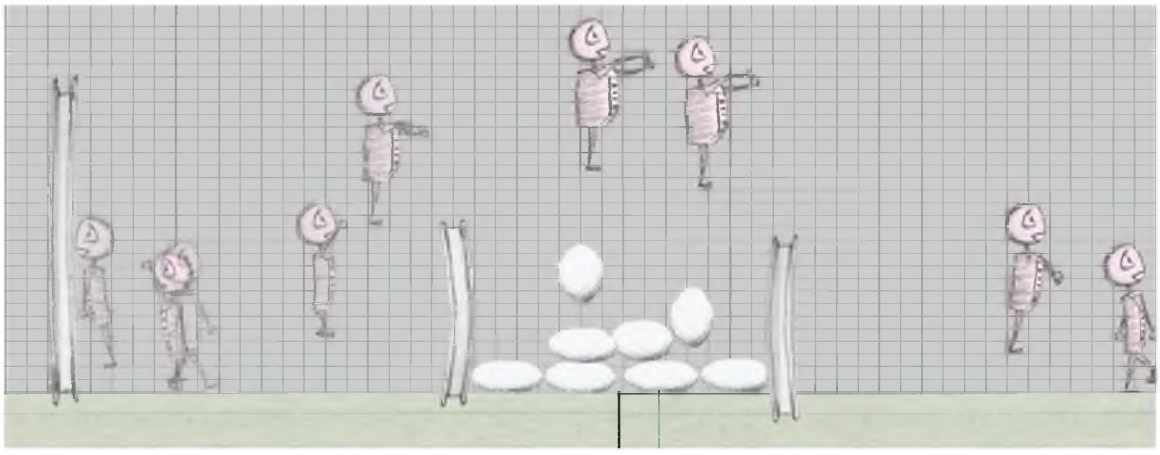

Figure 4.10. Our ragdoll uses a bouncy I-beam like a trampoline to gain enough sideways momentum to float over a pit of balls.

Table 4.1. Timing results for selected examples (ms per 60Hz frame).

	Platform Game	Puzzle Game	Launcher Game
Neighborhood Shape Matching	3.3	2.2	6.0
Global Shape Matching	0.3	0.1	0.1
Manifold Projection	5.0	8.9	2.0
Collisions	0.7	13.2	0.3
Other	1.2	1.0	0.5

CHAPTER 5

EXAMPLE-BASED PLASTIC DEFORMATION OF RIGID BODIES

5.1 Introduction

One of the greatest successes of physics-based animation is its widespread use for creating scenes containing large-scale destruction. The materials in these scenes are often man-made, carefully engineered and designed to be nearly rigid. Ensuring that building foundations remain stable, or that airplane wings maintain their shape in the presence of strong winds is vital to ensuring safety. The assumption of rigidity breaks down, however, when materials, well... break. During failure, man-made materials such as steel and concrete exhibit fracture and plastic deformation. For computer graphics applications, these failure cases are the most important – and most challenging – to animate.

Fracture is well-studied, both in engineering, and in computer graphics. Methods based on continuum mechanics and finite elements can produce realistic crack patterns and propagation. These methods are computationally expensive and in most cases model materials as elastic bodies. For mostly rigid objects, spending computational resources on elasticity calculations is essentially wasted effort, as most vibrations occur at frequencies that we cannot see; rigid body simulation can capture almost all of the important dynamics of the system. Consequently, geometric or artist-guided approaches to fracture are commonly employed in practice [Weinstein et al. 2008; Zafar et al. 2010; Criswell et al. 2010; Budsberg et al. 2014]. We adopt a similar approach to fracture: a nonphysical, artist-guided technique that integrates well with our plasticity model.

In comparison to fracture, plasticity is relatively poorly understood. Metalworkers have had some intuition about the properties of plasticity for centuries, heating, cooling, and folding the blades of their swords to maximize their strength. Analytical models, especially in graphics, are still based on heuristics. Measuring physical parameters is difficult because the materials must be destroyed during measurement. Also, in many engineering applications, analysts are concerned only with whether or not a material may fail; can they expect a particular component to withstand its environment or not? In computer graph-

ics, methods based on continuum mechanics are commonly used to animate elastoplastic materials. Again, this requires performing expensive, and frequently unnecessary, elasticity calculations. To ensure robustness, plasticity must either be limited, or the simulation domain must be periodically resampled or remeshed.

Rather than adapting engineering tools and techniques for use in graphics, we approach the problem from a different perspective: in this paper, we present a method specifically designed to animate destructive scenes, leveraging artist expertise and experience as much as possible. Our method is built on a new variant of linear blend skinning, example-based deformation, and rigid body dynamics. Artists create a simple rig for their simulated objects, then deform the object using this rig to create characteristic deformations. At simulation time, object dynamics are computed using an unmodified rigid body simulator. The objects are deformed by mapping impulses computed by the rigid body simulator to a spatially varying blend of the example deformations. This spatial variation allows us to create a wide range of deformations at run time with only two or three example poses.

The major contributions of this work are:

- An example-based deformation model based on linear blend skinning with a spatially varying blend of examples
- A method for mapping from discrete rigid body impulses to deformations
- A method for incorporating energy dissipation due to plastic deformation in system dynamics by modulating the coefficient of restitution
- A prescoring fracture approach that complements our deformation model.

The end result is a method that leverages existing artist expertise with rigging and skinning models; provides intuitive control over deformations by allowing artists to choose example deformations; and leverages common, efficient rigid body simulators to compute dynamics.

5.2 Method

To author assets that can be simulated using our technique, artists begin by rigging their model with bones.[1] Then, they use this rig to deform their input mesh into a set of characteristic example poses. We describe the particular methods used in the authoring phase of our pipeline in Section 5.3. An overview of our method is shown in Figure 5.1.

[1]In this discussion, we use the word "bone" generically to refer to the rig's degrees of freedom.

Objects in our system are modeled as rigid bodies, with a shape computed via modified linear blend skinning. Each object in our system stores standard rigid body properties:

- density, ρ, and mass, m (both constant during simulation)
- inertia tensor, $\mathbf{I}$
- linear position, $\mathbf{x}_{com}$, and momentum, $\mathbf{p}$
- orientation, $\mathbf{\Omega}$, and angular momentum, $\mathbf{L}$
- coefficient of restitution, C_r.

The geometry of our object is modeled using a tetrahedral mesh with N vertices. The skinned vertex positions are determined by the transformations of the B bones. The user provides a set of E example poses for the object, where each example contains a rotation and translation for each bone. To track skinning properties over time, each object stores

- undeformed mesh vertex positions, $\mathbf{u} \in \mathbb{R}^{N\times 3}$
- skinned mesh vertex positions, $\mathbf{x} \in \mathbb{R}^{N\times 3}$
- skinning weights, $\mathbf{W} \in \mathbb{R}^{N\times B}$
- example weights, $\mathbf{E} \in \mathbb{R}^{N\times E}$
- deformation accumulator, $\mathbf{\Delta E} \in \mathbb{R}^{N\times E}$.

$\mathbf{u}$ and $\mathbf{W}$ are constant, while the remaining properties change during simulation. Other material parameters are described below.

5.2.1 Skinning

Our method supports stylized deformation of object geometry while producing plausible local deformations by using a modified version of linear blend skinning. In traditional linear blend skinning, skinned vertex positions are computed as

$$\mathbf{x}_i = \sum_{b\in B} \mathbf{W}_{ib}\mathbf{T}_b\mathbf{u}_i, \tag{5.1}$$

where $\mathbf{T}_b$ is the current transformation of bone, b. To incorporate artist examples, we restrict bone transformations, $\mathbf{T}_b$, to interpolations of the bone transformations in the provided examples, $\mathbf{T}_{be}$. We store barycentric coordinates $\mathbf{e} \in \mathbb{R}^E$ that describe how to combine the example transformations. We split each transformation into a translation and rotation; translations are combined using linear interpolation and rotations are combined using QLERP [Kavan and Zara 2005].

Unfortunately, skinning the whole object using a single set of barycentric weights has two major drawbacks. First, the example deformations are global, while we expect plastic

deformation due to collision or other interactions to be somewhat localized. Second, as discussed in Chapter 4, blending many different poses simultaneously gives unintuitive results that poorly match the provided example poses.

Our solution to these problems is to allow the barycentric weights, $\mathbf{e}$, to vary spatially over the object, storing the weights for each mesh vertex in the matrix $\mathbf{E}$. This approach naturally supports local deformations, solving the first problem. We found that this approach also solves the second problem in most of our examples because interactions in the simulation tend to locally induce deformations that blend a small number of examples, avoiding "muddy" blends.

5.2.2 Impulse-based Deformation

In traditional deformable body simulation, each vertex has its own positional degrees of freedom and elastoplastic response is computed by analyzing stresses. Since we model our objects as rigid, this approach is unsuitable. Instead, we map the impulses generated by the rigid body constraint solver to deformations. Specifically, an impulse $\mathbf{j}_i$ at vertex i is mapped to a change in the barycentric coordinates in row i of the matrix, $\mathbf{E}$. To ensure smooth deformations, we propagate this change to nearby vertices using a smoothing kernel. An overview of this process is shown in Figure 5.2.

5.2.2.1 Projection

To compute the deformation at a single vertex, we seek to find a change in barycentric coordinates, $\mathbf{\Delta e}$, that would move the vertex in the direction of the applied impulse. We first map the impulse to a desired change in position by

$$\mathbf{\Delta x}_i = \alpha \ \max\left(||\mathbf{j}_i|| - \beta, 0\right) \hat{\mathbf{j}}_i, \tag{5.2}$$

where α is a scaling parameter and β is a threshold magnitude, preventing deformation, for example, during resting contact.

Next we compute the change in example weights that best matches this desired change in position. We can construct a Jacobian matrix whose columns represent the change in skinned position of vertex i with respect to change of example weights,

$$\mathbf{J}_i = \frac{\partial \mathbf{x}_i}{\partial \mathbf{E}_{ie}}, \mathbf{J}_i \in \mathbb{R}^{3\times E}. \tag{5.3}$$

Column e of this matrix is the change in skinned position for vertex i due to a change in example weight e. Using Equation 5.1, we can see

$$\frac{\partial \mathbf{x}_i}{\partial \mathbf{E}_{ie}} = \sum_b \mathbf{W}_{ib} \frac{\partial \left(\mathbf{T}_b \mathbf{u}_i\right)}{\partial \mathbf{E}_{ie}} = \sum_b \mathbf{W}_{ib} \left(\frac{\partial \mathbf{t}_b}{\partial \mathbf{E}_{ie}} + \frac{\partial \mathbf{R}_b \mathbf{u}_i}{\partial \mathbf{E}_{ie}} \right), \tag{5.4}$$

where the transformation, $\mathbf{T}_b$, has been separated into its translational, $\mathbf{t}_b$, and rotational, $\mathbf{R}_b$, parts.

Using a Jacobian to map from one coordinate space to another is a common strategy. For example, in character animation, the principal of virtual work maps forces in Cartesian space to torques in joint space by applying the Jacobian transpose [Pratt et al. 2001]. We could employ a similar strategy by computing a mapping using

$$\mathbf{\Delta e}_i = \mathbf{J}_i^T \mathbf{\Delta x}_i, \tag{5.5}$$

which could be viewed as a transformation to generalized coordinates. If $\mathbf{J}_i$ is full-rank and orthonormal, this change of basis works well. However, for most sets of example deformations, $\mathbf{J}_i$ is likely to have neither of these features. Small, or zero, singular values in $\mathbf{J}_i$ cause some impulses to result in little or no deformation; if there are large singular values or if the columns of $\mathbf{J}_i$ form an overcomplete basis, the result is deformations that are too large. The pseudo-inverse does not completely address these issues.

Instead, we advocate a mapping such that the magnitude of $\mathbf{\Delta e}_i$ is proportional to $\mathbf{\Delta x}_i$, regardless of the direction of the impulse, while still ensuring that the deformation is plausible. To accomplish this, we compute the SVD of the Jacobian, $\mathbf{J}_i = \mathbf{U\Sigma V}^T$. In our setting, the matrices computed by the SVD have an intuitive meaning: the matrix $\mathbf{U} \in \mathbb{R}^{3\times3}$ is a rotation matrix that encodes the preferred world space directions that trigger deformation. The matrix $\mathbf{V}$ encodes the deformation modes at that vertex, while $\mathbf{\Sigma}$ encodes their relative weights. Using this intuition, we compute the change in example weights as

$$\mathbf{\Delta e}_i = ||\mathbf{\Delta x}_i|| \mathbf{V} \frac{\left(\mathbf{\Sigma U}^T \mathbf{\Delta x}_i\right)}{||\left(\mathbf{\Sigma U}^T \mathbf{\Delta x}_i\right)||}. \tag{5.6}$$

Because of the normalization (and unit magnitude columns in V), the change is alway proportional to the desired change. The direction and weighting between modes is taken into account by the matrices $\mathbf{\Sigma}$ and $\mathbf{U}$. In the case that the impulse is orthogonal to the columns of $\mathbf{J}_i$, this approach will change the direction of the deformation to match the examples. If the columns of $\mathbf{J}_i$ form an overcomplete basis, this approach computes weights that balance between the different examples.

5.2.2.2 Propagation and Application

Once we compute $\mathbf{\Delta e}$ for the vertices where impulses have been applied, we propagate the change to nearby vertices using a smoothing kernel. However, we must be careful when choosing a distance metric. Distances should be shape aware, which rules out Euclidian

distances. Because our skinning weights are smooth by design, we considered computing weights between coordinates in skinning weight space. However, we found that distances between points in this space were close to a step function in our examples. Approximate geodesic distances, computed using a few passes of Dijkstra's algorithm, work well, and are used in all of our examples. We update each row of the matrix $\mathbf{\Delta E}$ by

$$\mathbf{\Delta E}_j \mathrel{+}= \phi\left(\frac{||\mathbf{x}_j - \mathbf{x}_i||}{\gamma}\right)\mathbf{\Delta e}_i \tag{5.7}$$

where the γ is the radius of the kernel, ϕ, and represents a user tunable parameter. We use the cubic kernel

$$\phi(x) = \begin{cases} 2x^3 - 3x^2 + 1 & : x < 1 \\ 0 & : \text{otherwise.} \end{cases} \tag{5.8}$$

We emphasize that this smoothing is applied to the example weights not to the rigid body impulses. Smoothing the impulses would lead to "dent-like" behavior while smoothing the example weights more closely follows the artist's intent.

Large impulses could result in large, instantaneous deformations. We address the resulting discontinuities by deforming our objects over time, rather than at the instant of impact. Specifically, we update the barycentric matrix by

$$\mathbf{E} \mathrel{+}= \lambda\mathbf{\Delta E}, \tag{5.9}$$

and then apply $\mathbf{\Delta E} \mathrel{*}= (1 - \lambda)$, where $\lambda \in [0, 1]$. This geometric series falls off quickly to negligible values, but allows plastic deformations to occur over several frames of animation. Please see the video for a comparison between several values of λ. To ensure that the rows of $\mathbf{E}$ remain barycentric, we clamp negative values to 0 and then scale each row so that its entries sum to one. This normalization discards deformation that would cause us to extrapolate outside our example space.

This delayed plasticity is related to the concept of plastic *creep*. When a material is held in a deformed state, plastic deformation accumulates over time, even though such deformation would be (almost) entirely elastic if the deformation occurs over a short period of time. Silly putty, for example, exhibits this behavior: it will bounce elastically, but well deform plastically if held against the ground. Since objects in our system are treated as rigid and experience instantaneous impulses at a small (and changing) set of contacts rather than sustained deformations, we cannot directly model creep. However, like creep, out approach accumulates plastic deformation over time.

Once we have updated example weights, we compute new skinned vertex positions, $\mathbf{x}$. We translate and rotate $\mathbf{x}$ so the local center of mass is at the origin, and is aligned with

the principal axes of inertia, as is required by our rigid body simulator. We also update the moment of inertia to account for changes in mass distribution.

5.2.3 Dynamics

As outlined in Figure 5.1, object dynamics are computed using an unmodified rigid body simulator (BulletPhysics in our implementation [Coumans 2014]). Each timestep is split into two phases, deformation and time integration. Before the deformation step, we save the rigid body state $(\mathbf{x}_{com}, \mathbf{p}, \mathbf{\Omega}, \mathbf{L})$ of each object. In the deformation phase, the simulator is stepped forward. The impulses computed during this step are used to deform the objects as described in Section 5.2.2.

Before the time integration step, the rigid body state is reset to its saved value, but the the shape is updated to account for the deformation step. The time integration step simply calls the rigid body simulator to get new positions and momenta.

This two-step scheme has two convenient features: first, it requires no modifications to the rigid body simulator; and second, the rigid body solver is given a full timestep to resolve any new collisions that may be caused by deformation.

5.2.3.1 Restitution Modification

During a violent collision, part of the kinetic energy of the system is dissipated as plastic deformation. This important aspect of plasticity is exploited by auto manufacturers, for example, who engineer components to crumple, reducing impact on passengers. We model this phenomenon by modulating the coefficient of restitution, C_r, while objects are deforming. We found that the scaling function,

$$C_r = \min\left(C_r^*, C_r + \mu\Delta t, \exp\left(-\nu\frac{||\mathbf{\Delta E}||_f}{n}\right)C_r^*\right), \tag{5.10}$$

where C_r^* is the default coefficient of restitution, and μ and ν are user controls, provides suitable damping. The second term ensures that C_r increases slowly after collisions, avoiding jittering artifacts. We experimented with several functions for the third term, such as a clamped linear falloff, and preferred the exponential function over the alternatives.

5.2.4 Fracture

Our implementation uses a prescoring approach to fracturing. The artist separates the input tetmesh into pieces using cutting planes, and we record which tetrahedra are split. For each piece, the collision geometry consists of the faces of the uncut tetrahedra, the triangulated cutting surfaces, and the triangulated, clipped faces of the cut tetrahedra as

shown in Figure 5.3. In our implementation, we restrict cuts to be planar and use Triangle to compute the cut boundary and triangulate its interior in 2D space [Shewchuk 1996]. In principle, however, any technique that produces a triangulated cutting surface, such as Voronoi-based fracture, could be used. The cut tetrahedra are duplicated to simplify skinning and constraint generation, but are not used for collision detection directly.

In our initial implementation, we did not split the cut tetrahedra, assigning each tet to a piece based on which side of the cutting planes its barycenter was on. Unfortunately, this approach caused problems for the collision solver in Bullet, which generated unreasonably large impulses to separate pieces after fracture. We suspect this is due to the dramatically varying normals of the tetrahedra faces, as well as the large number of contacts near the fracture plane. Using the volumetric tetrahedra for collision detection (rather than a surface triangle mesh) did not solve this issue, and ran significantly slower. Our cutting approach also improves the final rendered appearance compared to the semiregular jaggedness of the uncut tetrahedra.

To incorporate fracture into object dynamics, we generate point-to-point constraints for each tetmesh vertex shared by two pieces. We use Bullet's constraint threshold feature to automatically break them when enforcement impulses violate a threshold. We only allow the constraints to be broken during the deformation phase of the timestep. The masses of these vertices are divided between the shared pieces. Skinned positions of the vertices on the cutting surface are computed by barycentric interpolation of the enclosing tetrahedra, though if coarse tetmeshes are used, the vertices could use interpolated skinning and example weights, then compute positions via linear blend skinning. The only modification required to the deformation procedure is to consider the constraints when computing approximate geodesic distances: active constraints are considered zero-length edges between vertices.

5.3 Authoring Simulation Assets

We leverage recent research contributions for tetrahedral meshing, rigging, and skinning to automate as much of the authoring pipeline as possible. Artists begin by creating a closed triangle mesh. Next, they position a set of control bones and handles within the mesh. Once the handles are positioned, we automate the process of assigning skinning weights to mesh vertices by using bounded biharmonic weights [Jacobson et al. 2011]. These weights are computed through a constrained optimization, which requires a tetrahedral mesh of the object. Because we expect skinning weights to vary smoothly across the mesh, we compute a coarse, enclosing, approximating tetrahedralization using the approach of Stuart

and colleagues [Stuart et al. 2013], rather than using a conforming tetrahedral mesh as Jacobson and colleagues did. The meshing algorithm begins by tetrahedralizing a body centered cubic lattice that encloses the triangle mesh, then iteratively adjusts vertices of the mesh to match surface geometry. We ensure that the control handles specified by the artist correspond to vertices in the tetrahedralization by initially snapping the nearest vertices in the lattice to the handle locations and treating these vertex locations as constrained during the optimization.

Once the mesh is rigged, the artist manipulates the control handles to produce characteristic deformations. We use the fast automatic skinning transformations approach of Jacobson and colleagues [2012] to automate rotational degrees of freedom.

5.4 Results and Discussion

To demonstrate the effect of the material parameters in our system, we modify the parameters in a simple scene with 3 barrels being dropped on a loading dock. As seen in the accompanying video, it is possible to create widely varying results by tuning these parameters. Figure 5.4 demonstrates the effect of varying the kernel radius, γ.

We also animated three more complex scenes that may appear in movies or games. In the first, a reckless drive crashes into a stack of barrels and a wall, wrecking the car and barrels. The barrels all have the same material properties but display a wide variety of deformations due to the different impulses applied. Sample frames from the animation and the example poses are shown in Figure 5.5. Figure 5.6 shows how the example weights, $\mathbf{E}$, vary over the object meshes.

In the second, a bridge collapses under the weight of five shipping containers falling onto it, as shown in Figure 5.7. The bridge was prefractured with constraints automatically broken during the simulation. Our deformation model produces plausible deformations for both the containers and the bridge, even though they vary greatly in scale. While generating this example, the artist was unsatisfied with the bridge deformation in the preliminary results, so he added an additional example pose, resulting in significantly improved results. The ability to iterate quickly, including changing example deformations after seeing preliminary results, is a key advantage of our approach.

Finally, we animate a scene from a space battle. A fleet of small ships crashes into a larger one, causing minor damage to their foe, but completely destroying their fleet. The wings bend, twist, and break apart due to the impacts. Sample frames are shown in Figure 5.8.

5.4.1 Performance

The speed of our method is closely tied to the performance of the underlying rigid body simulator. Because each of our timesteps requires two calls to the rigid body simulator as well as our deformation computation, our simulator is 2.5 to 5 times slower than rigid body simulation, depending on the scene. For scenes with many interacting objects, such as the barrel pyramid in the car crash scene, collision detection dominates run time. The most expensive parts of our deformation model are computing approximate geodesic distances and performing skinning. In our implementation, these are multithreaded on the CPU, but a GPU-based implementation is likely to provide a significant speed boost, as both are very data parallel computations. These are computational bottlenecks because they affect every vertex of our volumetric mesh (or in the case of approximate geodesic distances, a subset that depends on kernel radius, γ). The car crash scene was the slowest scene due to the complex collision configuration. The average simulation time for that example was 1.5 seconds per 60 Hz frame on a Macbook Pro. For that scene, 43% of computation time was spent performing rigid body simulation, while 55% was spent computing deformation and skinning. The other scenes simulated at more than one frame per second.

While our runtimes would not be acceptable for games, by reducing mesh resolution and offloading computation to the GPU, we expect our method could run at interactive rates.

5.4.2 Limitations and Future Work

Because the space of deformations used in our system is so large, depending on the provided examples and parameters, objects may deform into self-intersecting configurations. This is most common when the plasticity scaling parameter, α, is large and the kernel radius, γ, is small. Because the mesh is only used for collision detection and rendering, these interpenetrations do not cause stability issues during simulation. Treating objects as "two-sided" during rendering further mitigates this problem.

Integrating our approach into a full featured modeling application would allow for greater flexibility. For example, users could paint various parameters over the mesh to create weak or strong areas. Such tools could also provide more flexibility for fracturing objects.

Our examples deliberately use a small number of example deformations, reducing the burden on the artist. If more examples were used, the basis formed by the columns of $\mathbf{J}_i$ would be overcomplete and the mapping in Equation 5.6 would strike a balance between the input examples. It would be interesting to explore ways of computing the examples weights in such underconstrained scenarios that satisfy secondary goals, such as smoothness of the

example weights or favoring using a single example over an average of all examples. Abe and Popović [2006] might provide some guidance in this direction.

The objects in our results are all treated as volumetric solids. Even objects, like the barrels, that might be better approximated as thin shells have their interior volume meshed. While this approach worked well for our examples, extending our techniques to thin shells remains an interesting area for future work. Relatedly, we make no effort to preserve the volume of our objects. Even if the artist's examples maintain volume, interpolations of the examples may not. It would be interesting to consider volume preservation as a secondary goal when the artist's examples form an overcomplete basis.

Due to the smoothness of linear blend skinning, it is difficult to achieve multiple scales of deformation simultaneously. For example, while the bridge our example demonstrates large-scale bending and twisting, it lacks smaller scale deformations such as denting or crumpling. Effects shots in film are often created by layering multiple simulations, so additional deformations could be added as a postprocess; however, incorporating an additional surface deformation model may produce more plausible results.

One major limitation of our method is that, like most other simulation techniques, the simulation must be adjusted by changing material parameters and initial conditions. While our example-based approach and intuitive parameters make it easier to author a simulation, it may still be necessary to tune parameters and rerun simulations multiple times to achieve a desired result. Allowing users to more directly adjust simulation output is an interesting direction for future work.

In summary, we have presented a technique for animating the failure of near-rigid man-made materials. Our primary contribution is an example-based plasticity model based on linear blend skinning that leverages rigid body simulation for dynamics. Our method is fast, artist friendly, and integrates easily into existing pipelines.

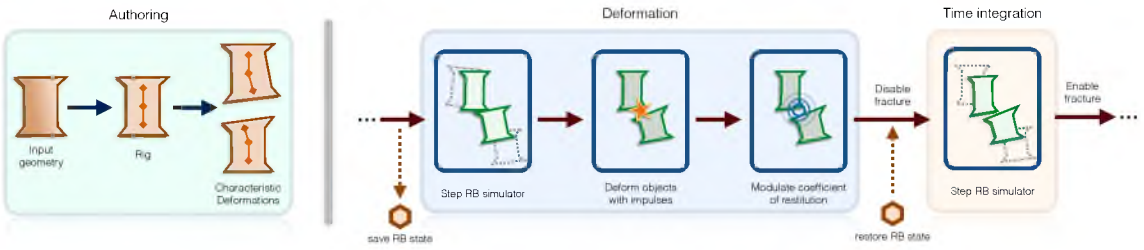

Figure 5.1. Overview of the authoring and simulation process.

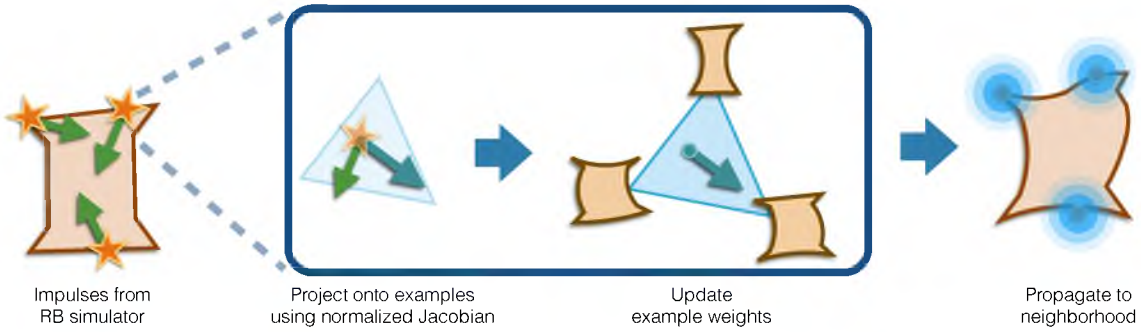

Figure 5.2. Overview of the deformation process.

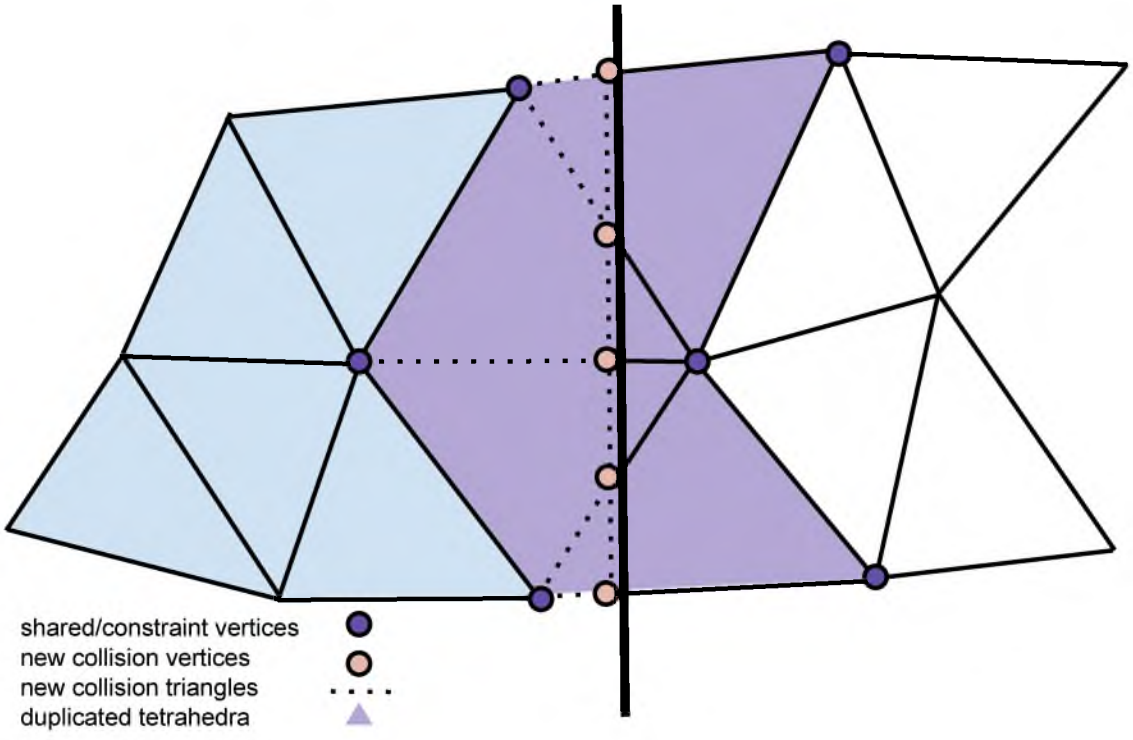

Figure 5.3. A fracture plane creates two new pieces. The surface triangles of the cut tetrahedra are clipped against the cutting plane. Constraints are created between the two pieces at shared vertices.

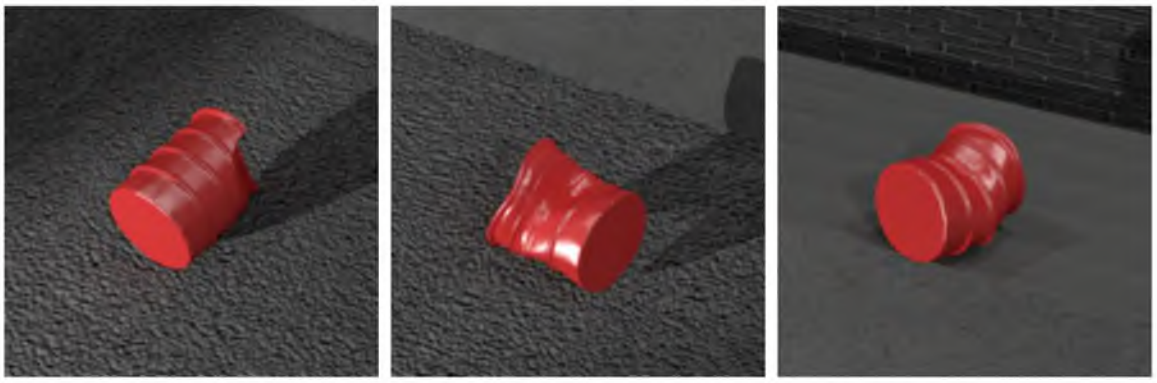

Figure 5.4. The kernel radius, γ, controls how far deformations propagate. Smaller values generate denting behavior, while larger values result in deformations that more closely match the example poses.

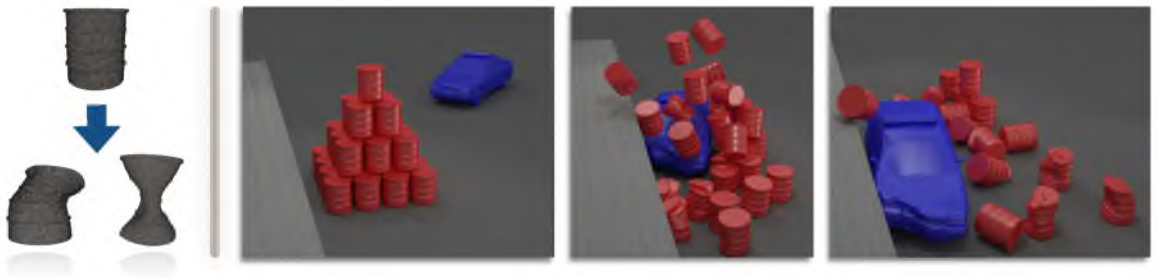

Figure 5.5. A reckless driver crashes his car into a stack of barrels. An artist provided example deformations of the barrel (left). At runtime, our simulator maps collision impulses to deformations that match the style of the provided examples while remaining physically plausible.

Figure 5.6. Colors represent the matrix, **E**, showing how the example weights vary over the objects. White vertices are undeformed; red and green correspond to the two input examples.

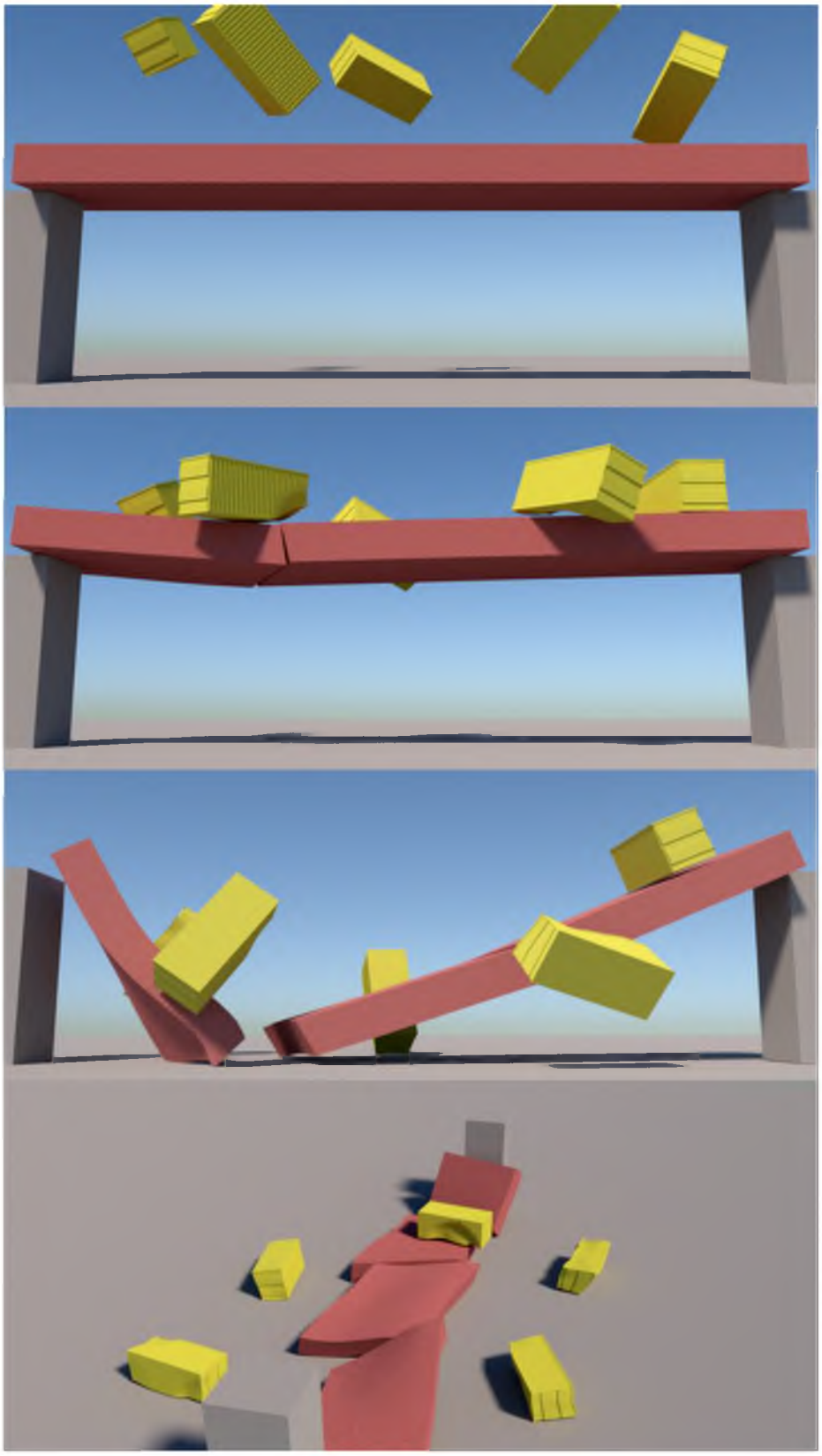

Figure 5.7. A bridge collapses as shipping containers fall onto it.

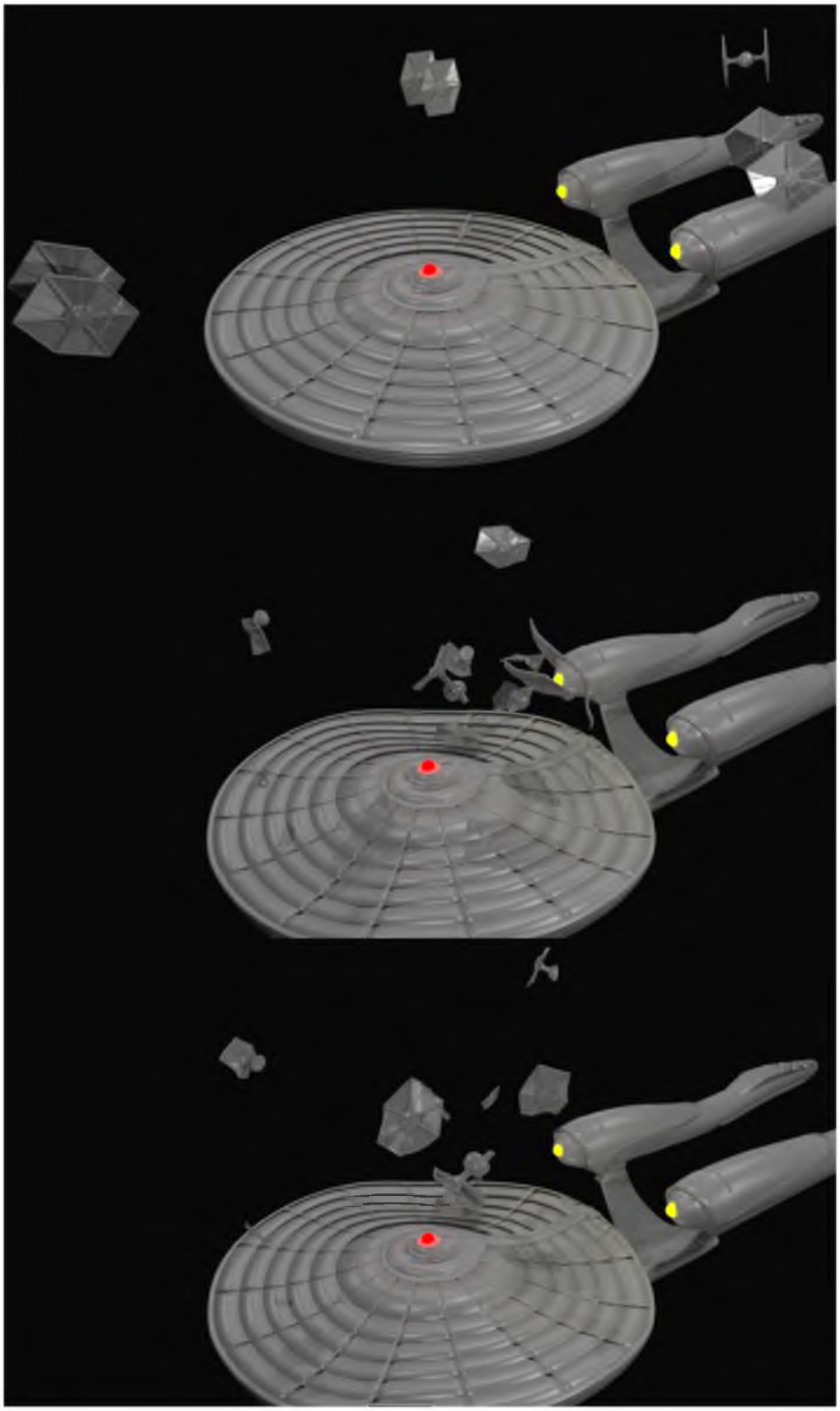

Figure 5.8. A small fleet of spaceships crashes into an enemy vessel.

CHAPTER 6

CONCLUSION

Though originally developed for engineering applications, physical simulation has proven to be a powerful tool for animation. In this dissertation, we showed that by reevaluating design decisions made for engineering applications, we can create tools that are easier for artists to work with and enable the creation of animations not feasible with existing techniques.

We first developed a point-based method for animating elastoplastic solids. Our technique, *deformation embedding*, allows us to animate objects undergoing extreme elastic and plastic deformations without volumetric meshing. This reduces requirements for artists modeling assets to be simulated: as long as we can reliably tell inside from outside, we can sample its interior with points and animate it using our simulator. Since artists often have experience working with point primitives, for example particle systems, they can use familiar tools to postprocess simulation results.

Next we developed *Dynamic Sprites*, a tool for creating stylized, animated objects suitable for interactive applications. Using example-based simulation as a starting point, we developed a flexible set of controls that allow artists to create stylized, exaggerated, and even nonphysical animations. Traditional simulation tools for engineering were certainly not developed with these types of applications in mind, so our approach required somewhat drastic changes to traditional simulation techniques. However, our technique still leverages physics to naturally handle some aspects of timing and collision response; this foundation is what makes our approach suitable for interactive applications containing unpredictable inputs and interactions.

Finally, we developed a method for animating destruction that focuses computational resources on the most visually important features: rigid motion, plasticity, and fracture. Our example-based plasticity model leverages artist expertise in rigging and skinning allowing intuitive artistic control over deformations in the resulting animations. Dynamics are computed using an unmodified rigid body simulator, so our technique is simple to implement, easy to integrate with existing pipelines, and computationally efficient. Our results show

that our method can be used to animate the types of destructive scenes common in film.

6.1 Future Directions

While we have demonstrated that our methods can be used to create stylized animations, we played the role of both researcher and artist for the examples in this dissertation. To develop methods that are optimized for artists and appropriate for production use, we must bring artists into the loop to validate and critique our designs. In particular, user studies can help us to understand which features of our techniques are improvements over existing tools, which features are regressions, and which artist challenges remain unaddressed.

A simple user study would entail asking artists to create an animation using both existing tools and our prototypes and comparing their experiences with each, for example using a concept art sketch as input. Qualitative results would be based on interviews both during and after the artist completes the task, while quantitative comparisons would be based on data such as the total time taken and the number of iterations required with various tools. To mimic production-like challenges, we could require the final animations to satisfy some "director constraints" such as the final location of objects.

In addition to validating the techniques in this thesis, there are a number of exciting further research opportunities for artist-guided animation tools. During the past decade, the internet has exploded as a place to share creative works as evidenced by the popularity of services like Flickr, Youtube, Reddit, and Vine. Armed with even the simplest content creation tools (for example a tool for putting funny text on a picture of a cat), users can create and share content that can inform, entertain, and inspire. By creating tools that let these users create interactive works, we can enable an entirely new way for people to express themselves. While the work presented in this dissertation is a start to making these tools a reality, there are still significant challenges to overcome.

Allowing users to provide example shapes is an effective means of artistic control; however, open questions about how best to use these shapes for simulation remain. Modal analysis has proven to be a valuable dimensional reduction technique for elastic simulation, speeding up integration by reducing the number of necessary degrees of freedom. This technique could prove to be useful for artist-guided simulation where the important degrees of freedom are hand-crafted, rather than computed. However, most graphics applications use linear modal analysis, assuming small deformations, whereas artists are likely to want extreme deformations where this assumption breaks down. Investigate how artist created modes and analytic or data-driven modes can be used to create a compact, expressive set of degrees of freedom that can be simulated efficiently is an exciting research direction.

While the work in this dissertation used example deformations as a means for artists to specify desired behavior, there may be other inputs that are more intuitive and expressive for artists. In particular, deformations do not provide useful information about how the object's global position and orientation should change over time. In our work with *Dynamic Sprites*, we allowed the user to control this via a set of simulation parameters, but allowing a user to specify characteristic trajectories in addition to characteristic deformations would help artists specify object behaviors in an intuitive way. Again, iteration with trained artists is necessary in order to validate that we are providing the most intuitive controls.

As 3D display technology improves and becomes inexpensive, there will be a strong demand for interactive, animated content for these devices. We may be able to leverage emerging input technologies such as hand and finger tracking to create such 3D content. Authoring tools that allow users to describe motions they wish to see, with their own, real-world motions, will greatly lower the barriers to entry for generating compelling animated content.

REFERENCES

Abe, Y. and Popović, J. (2006). Interactive Animation of Dynamic Manipulation. In: *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '06. Vienna, Austria, pp. 195–204.

Adams, B., Pauly, M., Keiser, R., and Guibas, L. J. (2007). Adaptively Sampled Particle Fluids. In: *ACM Trans. Graph.* 26.3, p. 48.

Ando, R., Thurey, N., and Tsuruno, R. (2012). Preserving fluid sheets with adaptively sampled anisotropic particles. In: *IEEE Trans. Vis. Comp. Graph.* 18.8, pp. 1202–1214.

Bao, Z., Hong, J.-M., Teran, J., and Fedkiw, R. (2007). Fracturing Rigid Materials. In: *IEEE Transactions on Visualization and Computer Graphics* 13.2, pp. 370–378.

Baraff, D. and Witkin, A. (1998). Large Steps in Cloth Simulation. In: *Computer Graphics (Proceedings of SIGGRAPH 98)*. Annual Conference Series, pp. 43–54.

Bargteil, A. and Jones, B. (2014). Strain Limiting for Clustered Shape Matching. In: *Proceedings of the ACM SIGGRAPH Conference on Motion in Games*.

Bargteil, A. W., Wojtan, C., Hodgins, J. K., and Turk, G. (2007). A Finite Element Method for Animating Large Viscoplastic Flow. In: *ACM Trans. Graph.* 26.3, p. 16.

Bergou, M., Mathur, S., Wardetzky, M., and Grinspun, E. (2007). TRACKS: Toward Directable Thin Shells. In: *ACM Trans. Graph.* 26.3, 50:1–50:10.

Bhattacharya, H., Gao, Y., and Bargteil, A. W. (2011). A Level-set Method for Skinning Animated Particle Data. In: *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. Vancouver, British Columbia.

Bridson, R., Marino, S., and Fedkiw, R. (2003). Simulation of Clothing with Folds and Wrinkles. In: *The Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 28–36.

Budsberg, J., Zafar, N. B., and Aldén, M. (2014). Elastic and Plastic Deformations with Rigid Body Dynamics. In: *ACM SIGGRAPH 2014 Talks*. SIGGRAPH '14. Vancouver, Canada, 52:1–52:1.

Clausen, P., Wicke, M., Shewchuk, J. R., and O'Brien, J. F. (2013). Simulating Liquids and Solid-Liquid Interactions with Lagrangian Meshes. In: *ACM Trans. Graph.* 32.2. Presented at SIGGRAPH 2013, 17:1–15.

Clavet, S., Beaudoin, P., and Poulin, P. (2005). Particle-based Viscoelastic Fluid Simulation. In: *The Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 219–228.

Cole, B. (2011). Kali: High Quality FEM Destruction in Zack Snyder's Sucker Punch. In: *ACM SIGGRAPH 2011 Talks*. ACM, p. 40.

Coumans, E. (2014). *Bullet Physics Library*. `http://bulletphysics.org/`.

Criswell, B., Smith, J., and Deuber, D. (2010). Transformers 2: Breaking Buildings. In: *ACM SIGGRAPH 2010 Talks*.

Fike, J. and Alonso, J. (2011). The Development of Hyper-dual Numbers for Exact Second-derivative Calculations. In: *AIAA paper* 886.

Gerszewski, D., Bhattacharya, H., and Bargteil, A. W. (2009). A Point-based Method for Animating Elastoplastic Solids. In: *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. New Orleans,Louisiana.

Goktekin, T. G., Bargteil, A. W., and O'Brien, J. F. (2004). A Method for Animating Viscoelastic Fluids. In: *ACM Trans. Graph.* 23.3, pp. 463–468.

Goktekin, T. G., Reisch, J., Peachey, D., and Shah, A. (2007). An Effects Recipe for Rolling a Dough, Cracking an Egg and Pouring a Sauce. In: *ACM SIGGRAPH 2007 sketches*, p. 67.

Hegemann, J., Jiang, C., Schroeder, C., and Teran, J. M. (2013). A Level Set Method for Ductile Fracture. In: *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. ACM, pp. 193–201.

Irving, G., Teran, J., and Fedkiw, R. (2004). Invertible Finite Elements For Robust Simulation of Large Deformation. In: *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 131–140.

Jacobson, A., Baran, I., Popović, J., and Sorkine, O. (2011). Bounded Biharmonic Weights for Real-time Deformation. In: *ACM Trans. Graph.* 30.4, 78:1–78:8.

Jacobson, A., Baran, I., Kavan, L., Popović, J., and Sorkine, O. (2012). Fast Automatic Skinning Transformations. In: *ACM Trans. Graph.* 31.4, 77:1–77:10.

Jakobsen, T. (2001). Advanced Character Physics. In: *Game Developers Conference*, pp. 383–401.

Kavan, L. and Zara, J. (2005). Spherical Blend Skinning: A Real-time Deformation of Articulated Models. In: *2005 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pp. 9–16.

Kavan, L., Gerszewski, D., Bargteil, A., and Sloan, P.-P. (2011). Physics-inspired Upsampling for Cloth Simulation in Games. In: *ACM Trans. Graph.* 30.4, 93:1–93:9.

Koyama, Y., Takayama, K., Umetani, N., and Igarashi, T. (2012). Real-time Example-based Elastic Deformation. In: *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '12, pp. 19–24.

Liu, T., Bargteil, A. W., O'Brien, J. F., and Kavan, L. (2013). Fast Simulation of Mass-spring Systems. In: *ACM Transactions on Graphics (TOG)* 32.6, p. 214.

Macklin, M. and Müller, M. (2013). Position Based Fluids. In: *ACM Transactions on Graphics (TOG)* 32.4, p. 104.

Macklin, M., Müller, M., Chentanez, N., and Kim, T.-Y. (2014). Unified Particle Physics for Real-time Applications. In: *ACM Transactions on Graphics (TOG)* 33.4, p. 153.

Martin, S., Kaufmann, P., Botsch, M., Grinspun, E., and Gross, M. (2010). Unified simulation of elastic rods, shells, and solids. In: *ACM Trans. Graph.* 29 (4), 39:1–39:10.

Martin, S., Thomaszewski, B., Grinspun, E., and Gross, M. (2011). Example-based Elastic Materials. In: *ACM Trans. Graph.* 30.4, 72:1–72:8.

Müller, M. and Gross, M. (2004). Interactive Virtual Materials. In: *The Proccedings of Graphics Interface*, pp. 239–246.

Müller, M., Charypar, D., and Gross, M. (2003). Particle-Based Fluid Simulation for Interactive Applications. In: *The Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 154–159.

Müller, M. and Chentanez, N. (2011). Solid Simulation with Oriented Particles. In: *ACM Trans. Graph.* 30.4, 92:1–92:10.

Müller, M., McMillan, L., Dorsey, J., and Jagnow, R. (2001). Real-Time Simulation of Deformation and Fracture of Stiff Materials. In: *Computer Animation and Simulation 2001.* Proceedings of the Eurographics Workshop, pp. 113–124.

Müller, M., Dorsey, J., McMillan, L., Jagnow, R., and Cutler, B. (2002). Stable Real-Time Deformations. In: *The Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 49–54.

Müller, M., Keiser, R., Nealen, A., Pauly, M., Gross, M., and Alexa, M. (2004). Point Based Animation of Elastic, Plastic and Melting Objects. In: *The Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 141–151.

Müller, M., Heidelberger, B., Teschner, M., and Gross, M. (2005). Meshless Deformations Based on Shape Matching. In: *ACM Trans. Graph.* 24.3, pp. 471–478.

Müller, M., Heidelberger, B., Hennix, M., and Ratcliff, J. (2007). Position Based Dynamics. In: *J. Vis. Comun. Image Represent.* 18.2, pp. 109–118.

Müller, M., Chentanez, N., and Kim, T.-Y. (2013). Real Time Dynamic Fracture with Volumetric Approximate Convex Decompositions. In: *ACM Transactions on Graphics (TOG)* 32.4, p. 115.

O'Brien, J. F. and Hodgins, J. K. (1999). Graphical Modeling and Animation of Brittle Fracture. In: *The Proceedings of ACM SIGGRAPH 99*, pp. 137–146.

O'Brien, J. F., Bargteil, A. W., and Hodgins, J. K. (2002). Graphical Modeling and Animation of Ductile Fracture. In: *ACM Trans. Graph.* 21.3, pp. 291–294.

Parker, E. G. and O'Brien, J. F. (2009). Real-Time Deformation and Fracture in a Game Environment. In: *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 156–166.

Pauly, M., Keiser, R., Adams, B., Dutré; P., Gross, M., and Guibas, L. J. (2005). Meshless Animation of Fracturing Solids. In: *ACM Trans. Graph.* 24.3, pp. 957–964.

Pfaff, T., Narain, R., Joya, J. M. de, and O'Brien, J. F. (2014). Adaptive Tearing and Cracking of Thin Sheets. In: *ACM Transactions on Graphics* 33.4, xx:1–9.

Pratt, J., Chew, C.-M., Torres, A., Dilworth, P., and Pratt, G. (2001). Virtual Model Control: An Intuitive Approach for Bipedal Locomotion. In: *The International Journal of Robotics Research* 20.2, pp. 129–143.

Rivers, A. R. and James, D. L. (2007). FastLSM: Fast Lattice Shape Matching for Robust Real-time Deformation. In: *ACM Trans. Graph.* 26.3.

Ruilova, A. (2007). Creating Realistic CG Honey. In: *ACM SIGGRAPH 2007 posters*. San Diego, California, p. 58.

Schumacher, C., Thomaszewski, B., Coros, S., Martin, S., Sumner, R., and Gross, M. (2012). Efficient Simulation of Example-based Materials. In: *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 1–8.

Shewchuk, J. R. (1996). Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In: *Applied Computational Geometry: Towards Geometric Engineering*. Ed. by M. C. Lin and D. Manocha. Vol. 1148. Lecture Notes in Computer Science. From the First ACM Workshop on Applied Computational Geometry, pp. 203–222.

Solenthaler, B. and Gross, M. (2011). Two-scale Particle Simulation. In: *ACM Trans. Graph.* 30.4, 81:1–81:8.

Sorkine, O. and Alexa, M. (2007). As-rigid-as-possible Surface Modeling. In: *Proceedings of the fifth Eurographics symposium on Geometry processing*. Barcelona, Spain, pp. 109–116.

Stuart, A., Levine, J., Jones, B., and Bargteil, A. (2013). Automatic Construction of Coarse, High-Quality Tetrahedralizations that Enclose and Approximate Surfaces for Animation. In: *Proceedings of the ACM SIGGRAPH Conference on Motion in Games*. Dublin,Ireland.

Su, J., Schroeder, C., and Fedkiw, R. (2009). Energy Stability and Fracture for Frame Rate Rigid Body Simulations. In: *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '09. New Orleans, Louisiana, pp. 155–164.

Sýkora, D., Dingliana, J., and Collins, S. (2009). As-rigid-as-possible Image Registration for Hand-drawn Cartoon Animations. In: *Proceedings of International Symposium on Non-photorealistic Animation and Rendering*, pp. 25–33.

Terzopoulos, D. and Fleischer, K. (1988). Modeling Inelastic Deformation: Viscoelasticity, Plasticity, Fracture. In: *Proceedings of ACM SIGGRAPH*, pp. 269–278.

Terzopoulos, D., Platt, J., Barr, A., and Fleischer, K. (1987). Elastically Deformable Models. In: *Proceedings of ACM SIGGRAPH*. Vol. 21, pp. 205–214.

Weinstein, R., Petterson, F., and Criswell, B. (2008). Destruction System. In: *ACM SIGGRAPH 2008 Talks*. Los Angeles, California, 71:1–71:1.

Wicke, M., Ritchie, D., Klingner, B. M., Burke, S., Shewchuk, J. R., and O'Brien, J. F. (2010). Dynamic Local Remeshing for Elastoplastic Simulation. In: *ACM Trans. Graph.* 29 (4), 49:1–49:11.

Wojtan, C. and Turk, G. (2008). Fast Viscoelastic Behavior with Thin Features. In: *ACM Trans. Graph.* 27 (3), 47:1–47:8.

Zafar, N. B., Stephens, D., Larsson, M., Sakaguchi, R., Clive, M., Sampath, R., Museth, K., Blakey, D., Gazdik, B., and Thomas, R. (2010). Destroying LA for "2012". In: *ACM SIGGRAPH 2010 Talks*, 25:1–25:1.

Zhou, Y., Lun, Z., Kalogerakis, E., and Wang, R. (2013). Implicit Integration for Particle-based Simulation of Elasto-Plastic Solids. In: *Computer Graphics Forum*. Vol. 32. 7. Wiley Online Library, pp. 215–223.