Example-based Plastic Deformation of Rigid Bodies

Ben Jones
University of Denver

Nils Thuerey
Technical University of Munich

Tamar Shinar
University of California, Riverside

Adam W. Bargteil
University of Maryland, Baltimore County

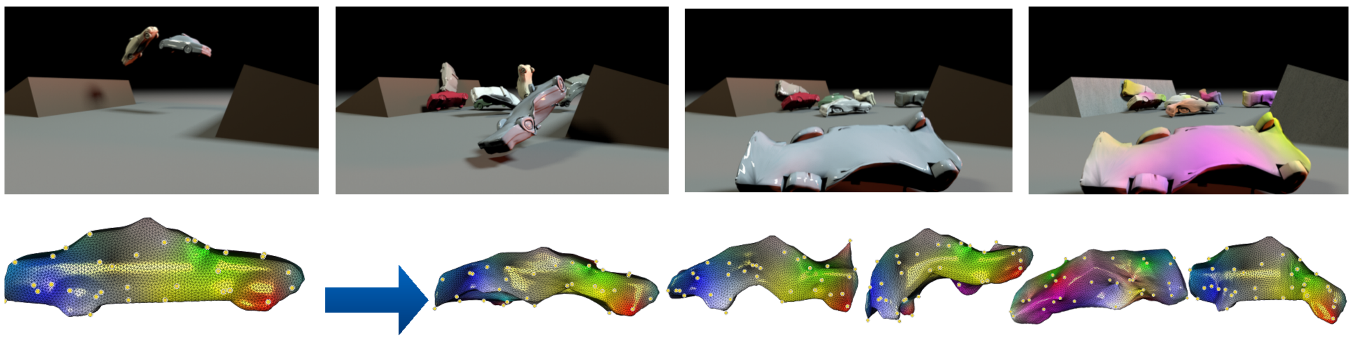

Figure 1: *Several stunt drivers crash in midair and their cars pile up on the ground. An artist provided a small set of example deformations (bottom). At runtime, our simulation maps collision impulses to deformations that match the style of the provided examples while remaining physically plausible. Colors of the right image indicate the spatially varying blend of example poses.*

Abstract

Physics-based animation is often used to animate scenes containing destruction of near-rigid, man-made materials. For these applications, the most important visual features are plastic deformation and fracture. Methods based on continuum mechanics model these materials as elastoplastic, and must perform expensive elasticity computations even though elastic deformations are imperceptibly small for rigid materials. We introduce an example-based plasticity model based on linear blend skinning that allows artists to author simulation objects using familiar tools. Dynamics are computed using an unmodified rigid body simulator, making our method computationally efficient and easy to integrate into existing pipelines. We introduce a flexible technique for mapping impulses computed by the rigid body solver to local, example-based deformations. For completeness, our method also supports prescoring based fracture. We demonstrate the practicality of our method by animating a variety of destructive scenes.

Keywords: Plasticity, Deformation, Example-based Simulation, Rigid Body Simulation, Skinning, Local Blending

Concepts: **•Computing methodologies → Simulation by animation; Physical simulation;**

SIGGRAPH '16 Technical Paper,, July 24 - 28, 2016, Anaheim, CA,
ISBN: 978-1-4503-4279-7/16/07
DOI: http://dx.doi.org/10.1145/2897824.2925979

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 figuratively breaks down when rigid bodies literally 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, however, because they typically 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 can't 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. 2010b; Budsberg et al. 2014].

We propose a similarly practical approach to animating plastic deformation in destructive scenes. Rather than relying on continuum mechanics and expensive elastoplastic simulations, in this paper we present an approach to plastic deformation that fits into traditional, and inexpensive, rigid body simulation. By design, our example-based approach leverages artist expertise and experience; 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 in example weights represents the key insight that allows our method to create simulation assets that respond locally and naturally to stimuli and demonstrate a rich space of run time deformations given only a handful of artist examples.

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

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.

2 Related Work

As noted above, methods based on continuum mechanics are commonly used to animate fracture and plastic deformations. The importance of these phenomena in computer animation is evident by the fact that a single year after Terzopoulos and colleagues [1987] demonstrated the first simulated elastic bodies Terzopoulos and Fleischer [1988] enhanced these models with plasticity and fracture. Since this seminal work a number of researchers have demonstrated plastic deformation and fracture of solid objects using spring-mass [Norton et al. 1991; Hirota et al. 1998; Hirota et al. 2000; Clavet et al. 2005; Levine et al. 2014], finite element [O'Brien and Hodgins 1999; O'Brien et al. 2002; Irving et al. 2004; Molino et al. 2004; Bargteil et al. 2007; Wojtan and Turk 2008; Wojtan et al. 2009; Parker and O'Brien 2009; Wicke et al. 2010; Clausen et al. 2013], point-based [Müller et al. 2004; Pauly et al. 2005; Müller et al. 2007; Martin et al. 2010; Müller and Chentanez 2011; Jones et al. 2014], and Eulerian [Fan et al. 2013] techniques. These methods have been quite successful and many of them run in real-time, however all these approaches build on techniques for simulating elastic bodies and necessarily require more computation than rigid body simulation, even for objects that demonstrate limited elastic deformations. In fact, because many of these methods are designed to animate large elastic deformations, they are particularly ill-suited to animating very stiff, nearly-rigid, materials. In practice, these approaches settle for larger than desired elastic deformations (e.g. [BeamNG 2016]), resort to coarse resolutions, sacrifice real-time computation, or employ model reduction through modal analysis (e.g. [Barbič and James 2005; Harmon and Zorin 2013; Yang et al. 2015] or leverage other pre-computation (e.g. [Liu et al. 2013; Bouaziz et al. 2014]) and sacrifice their ability to accurately handle online changes to the underlying physics, such as those that occur during fracture and/or plastic deformations. One approach to limiting computational cost is to treat objects as rigid bodies, but employ finite element simulation when plastic deformations might occur [Müller et al. 2001; Criswell et al. 2010a]. In our approach, we never resort to full finite element simulation, instead we use an artist's description of desired plastic deformations to guide the animation of plastic deformation.

More similar to our method is the classic approach of using rigid body simulation to animate fracture. This approach generally makes use of "pre-scoring," or predefined fracture patterns. Typically, the individual pieces in the pre-scored geometry are loaded into a rigid body simulator and constraints are added between adjacent pieces. At prescribed times or when internal stresses are large, the constraints are removed creating the fracture effect. In an interesting variation on this approach, Bao and colleagues [2007] used quasistatic finite element analysis to create fracture and dent patterns. In an alternative approach, Su and colleagues [2009] allowed the pre-scoring patterns to move over the geometry to align with collision events. This general approach has been well documented in the visual effects industry [Weinstein et al. 2008; Zafar et al. 2010; Criswell et al. 2010b; Budsberg et al. 2014]. Our implementation also includes this general approach to fracture.

While enhancing rigid body simulation with fracture is both well-studied and common in practice, enhancing rigid body simulations with plastic deformations has remained relatively unexplored. One classic approach commonly used in video games simply replaces the underlying geometry when destructive events occur. Such approaches often employ finite state machines, impact accumulators, and secondary effects such as smoke or explosions to hide temporal discontinuities. With such tools talented artists can create very compelling animations of destruction. However, while these approaches may react differently to different stimuli, they are not able to generalize beyond the discrete examples created by the artist. In contrast, our approach locally blends multiple examples to create a rich space of novel deformations.

More recently, Patkar and colleagues [2014] explored plastic deformation of rigid bodies through denting and bending. Denting was accomplished with "dent maps" and artists were given control over the degree of denting ("dent distance") as well as the smoothness of the dents. Bending was accomplished by embedding articulated skeletons into the object geometry and allowing the joint angles to vary over time. In a similar approach, Budsberg and colleagues [2014] decomposed objects into a set of spheres and placed ball and socket joints between neighboring spheres. Plastic deformation was achieved by modifying the rest state of these joints. In contrast, our approach allows the artist to specify more general deformation rigs and we achieve local effects, analogous to dents, by allowing example weights to vary spatially over the object.

We are not the first to use linear-blend skinning as the underlying deformation model in a simulation. Gilles and colleagues [2011] and Faure and colleagues [Faure et al. 2011] used linear-blend skinning to reduce the number of degrees of freedom in the system, opting for user control over analytic or data-driven model reduction [Barbič and James 2005]. While they used Lagrangian mechanics to simulate elasticity, we cast our problem as an energy minimization to map impulses to plastic deformations. Similarly Fan and colleagues [2013] map impulses between disparate simulation techniques—in their case rigid body and Eulerian deformable.

Our approach also draws on the recent ideas behind "example-based simulation." Example-based, or data-driven, methods are common in computer graphics, but until recently had seen limited application to animating passive objects. Cloth animation is a notable exception because clothing is closely related to the underlying character's pose, which makes clothing an excellent candidate for combining data, skinning, and simulation. Kim and Vendrovsky [2008] demonstrated that the position of clothing could be well predicted given a small number of example poses and clothing configurations. They also noted that the interface for artists was particularly intuitive. If an artist didn't like the results, they simply changed the examples. Of course, such an approach only works when the number of examples is relatively small. More recently, other researchers have explored various data-driven approaches to cloth animation (e.g. [Wang et al. 2010; de Aguiar et al. 2010; Feng et al. 2010; Kavan et al. 2011]). Very recently Hahn and colleagues [2014] used data to learn how to choose an effective low-dimensional simulation subspace based on body pose and clothing state. They noted that even though they did not explicitly ask for

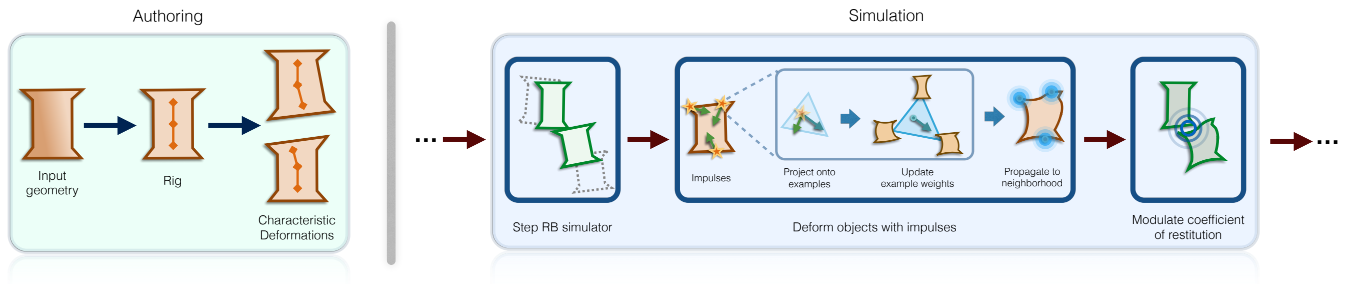
Authoring
Input geometry
Rig
Characteristic Deformations
Simulation
Step RB simulator
Impulses
Project onto examples
Update example weights
Propagate to neighborhood
Deform objects with impulses
Modulate coefficient of restitution

Figure 2: *Overview of the authoring and simulation process.*

sparse basis vectors, the vectors tended to be sparse, which resulted in novel poses in a similar way to our spatially varying example weights. Xu and colleagues [2014] also locally blend clothing examples based on body poses. In contrast, our blends are determined by impulses from a rigid body simulation rather than from the pose of an underlying skeleton.

Researchers have also been exploring example-based animation of elastic bodies. This approach was introduced by Martin and colleagues [2011], who employed an additional energy functional that draws the deformed world state of the elastic object toward an "example manifold" determined by a set of artist chosen examples. More recent work [Schumacher et al. 2012] improved performance and used examples to also guide plastic deformations. Koyama and colleagues [2012] proposed an alternative formulation based on a shape-matching framework [Müller et al. 2005]. One way to view their approach is that the rest shape of the object is changing over time—moving along the example manifold, which is defined through linear interpolation of example poses. Jones and colleagues [2013] adopted a similar approach and described additional mechanisms for moving along the example manifold. We also take a similar approach, but we discard the elastic component of the deformation and directly change the shape of the rigid body to the shape defined by the position on the example manifold. Our approach has two major advantages. First, because our deformations are computed from linear blend skinning rather than a continuum mechanics model, it is computationally cheaper, more suited to artistic authoring, and more flexible. Second, while previous work [Martin et al. 2011; Koyama et al. 2012] broke objects into regions and defined example manifolds for each region independently, we avoid breaking the object into pieces and simply allow the example weights to vary spatially across our object. This approach results in a rich and continuous space of deformations.

3 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 4. An overview of our method is shown in Figure 2.

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:

[1]In this discussion, we use the word "bone" generically to refer to the rig's degrees of freedom. Because we use bounded biharmonic weights [Jacobson et al. 2011] to deform our geometry we support bones, control handles, and cages.

- 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.

In the following discussion, we assume that the geometry of our object is modeled using a tetrahedral mesh with N vertices. We note, however, that tetrahedral meshes are not required by our approach; they just simplify some computations in our implementation (see Section 4). 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, $\mathbf{\Delta E} \in \mathbb{R}^{N\times E}$
- example transformations, $\mathbf{T} \in \mathbb{R}^{7B\times E}$.

$\mathbf{u}$, $\mathbf{W}$, and $\mathbf{T}$ are constant, while the remaining properties change during simulation. $\mathbf{T} \in \mathbb{R}^{7B\times E}$ stores a position and quaternion for each bone. The key to our method is using impulses collected from the rigid body simulator to induce changes in $\mathbf{E}$, which in practice is quite sparse. $\mathbf{u}$ and $\mathbf{x}$ are stored in the object's local coordinate system. Other material parameters are described below.

3.1 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

$$\mathbf{x}_i = \sum_{b\in B} \mathbf{W}_{ib}\mathbf{T}_b\mathbf{u}_i, \tag{1}$$

where $\mathbf{T}_b$ is the current transformation of bone, b. To incorporate artist examples, we generalize the bone transformations, $\mathbf{T}_b$, to blends of the bone transformations in the examples. If transformations blended linearly, we could write,

$$\mathbf{x}_i = \sum_{b\in B} \mathbf{W}_{ib}\left(\sum_{e=1}^{E} \mathbf{E}_{ie}\mathbf{T}_{be}\right)\mathbf{u}_i. \tag{2}$$

Instead we interpolate the translational and rotational components separately, linearly blending the translations and interpolating rotations with QLERP [Kavan and Zara 2005],

$$\mathbf{x}_i = \sum_{b \in B} \mathbf{W}_{ib} \, \mathrm{QLERP}\left(\mathbf{E}_i, \mathbf{T}_b\right) \mathbf{u}_i. \tag{3}$$

To avoid extrapolation outside of the example space, we ensure that each entry of $\mathbf{E}$ is between 0 and 1, and that each row of $\mathbf{E}$ sums to 1, i.e. $\forall i \forall e, 0 \leq \mathbf{E}_{ie} \leq 1$ and $\forall i, \sum_e \mathbf{e}_{ie} = 1$. Thinking of the examples as defining an $(E-1)$-dimensional simplex, each row of $\mathbf{E}$ stores barycentric coordinates that describe how to combine the example transformations for vertex i. The first column of $\mathbf{E}$, which corresponds to the rest pose, is initialized to 1 and the rest of the entries are initialized to 0.

Notably, the barycentric coordinates vary spatially, but smoothly, over our object; providing us with two advantages over global approaches to blending examples. First, we can achieve the somewhat localized plastic deformations that we expect from colliding, otherwise rigid, bodies. Second, and more subtly, as shown by Jones and colleagues [2013], global blends of many different poses gives unintuitive results that poorly match the provided example poses; like mixing too many paints on a palette. In our approach, while many examples are often used to create the whole deformation, the deformation of any given vertex is typically a blend of just a few examples, avoiding “muddy” blends.

3.2 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 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 illustrated in the center part of Figure 2.

3.2.1 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

$$\mathbf{\Delta x}_i = \frac{\Delta t}{m} \max\left(||\mathbf{j}_i|| - \beta, 0\right) \frac{\mathbf{j}_i}{||\mathbf{j}_i||}, \tag{4}$$

where Δt is the timestep, m is the mass of the object, and β is a plastic yield threshold, which prevents 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,

$$\mathbf{J}_i = \frac{\partial \mathbf{x}_i}{\partial \mathbf{E}_{ie}}, \mathbf{J}_i \in \mathbb{R}^{3 \times E}. \tag{5}$$

Column e of this matrix is the change in skinned position for vertex i due to a change in example weight e. We compute these derivatives exactly using automatic differentiation [Stauning and Bendtsen 2007].

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

$$\mathbf{\Delta e}_i = \mathbf{J}_i^T \mathbf{\Delta x}_i, \tag{6}$$

where $\mathbf{\Delta e}_i$ is the change in example weights at vertex i and is a non-physical quantity. This approach could be viewed as a transformation to generalized coordinates. Another view is that we seek the solution, $\mathbf{\Delta e}$, to the following minimization problem

$$\min_{\mathbf{\Delta e}} \| \left(\mathbf{x}_i(\mathbf{e} + \mathbf{\Delta e}) - \mathbf{x}_i(\mathbf{e})\right) - \mathbf{\Delta x}_i \|^2, \tag{7}$$

where $\mathbf{x}_i(\mathbf{e})$ is the nonlinear function that gives the skinned position of vertex i for barycentric example weights $\mathbf{e}$. Equation (6) gives the gradient direction for this nonlinear optimization problem (assuming an initial guess of $\mathbf{\Delta e} = 0$ so that the first terms cancel). To solve the nonlinear optimization, we could take a step in this direction, update $\mathbf{J}_i$, and compute a new gradient direction. Instead, we have found it sufficient to take a single step of gradient descent with a carefully chosen step size. So that the plastic deformation is proportional to how much the impulse exceeds the plastic yield threshold, we choose the size of the step in the gradient direction to be $\alpha \|\Delta \mathbf{x}_i\|$, with α a user-defined scaling parameter. Specifically, we compute the change in example weights as

$$\mathbf{\Delta e}_i = \alpha \, ||\mathbf{\Delta x}_i|| \frac{\mathbf{J}_i^T \mathbf{\Delta x}_i}{\|\mathbf{J}_i^T \mathbf{\Delta x}_i\|}. \tag{8}$$

To avoid dividing by zero when an impulse is orthogonal to all the example deformations, we set $\mathbf{\Delta e}_i = \mathbf{0}$ if $\|\mathbf{J}_i^T \mathbf{\Delta x}_i\| \leq \epsilon$. We used $\epsilon = 10^{-8}$ and this code was not triggered in any of our experiments.

An alternative and appealing approach is to linearize $\mathbf{x}_i(\mathbf{e})$ in Equation (7), compute the pseudoinverse, $\mathbf{J}^+$, and compute

$$\Delta \mathbf{e}_i = \alpha \mathbf{J}^+ \Delta \mathbf{x}_i. \tag{9}$$

We initially tried this approach and found that $\mathbf{J}$ often has a wide Eigenspectrum, with many Eigenvalues arbitrarily close to zero. Consequently, results were highly sensitive to the choice of threshold used when deciding whether to invert Eigenvalues and we were unable to find good choices for individual examples, let alone a choice that worked well across examples. We experimented with a number of other approaches before settling on Equation (8), which works quite well in our experience. The accompanying video includes comparisons to both Equation (9) and the “natural” step-size for gradient descent,

$$\mathbf{\Delta e}_i = \alpha \mathbf{J}_i^T \mathbf{\Delta x}_i. \tag{10}$$

We found that making the stepsize proportional to $\|\mathbf{\Delta x}_i\|$ worked surprisingly well and is very fast to compute.

3.2.2 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 Euclidean 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

$$\mathbf{\Delta E}_j \mathrel{+}= \phi\left(\frac{||\mathbf{x}_j - \mathbf{x}_i||}{\gamma}\right) \mathbf{\Delta e}_i \tag{11}$$

where the $\gamma > 0$ is the radius of the kernel, ϕ, and represents a user tunable parameter. We use the cubic kernel

$$\phi(x) = \begin{cases} 2x^3 - 3x^2 + 1 & : x < 1 \\ 0 & : \text{otherwise.} \end{cases} \quad (12)$$

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

$$\mathbf{E} \mathrel{+}= (1 - \lambda)\, \mathbf{\Delta E}, \quad (13)$$

and then apply $\mathbf{\Delta E} \mathrel{*}= \lambda$, where $\lambda \in [0, 1)$ is a user-tunable parameter that should vary with the inverse of the timestep. This approach has several advantages. First, because

$$(1 - \lambda) \sum_{i=0}^{\infty} \lambda^i = 1 \quad (14)$$

the total accumulated deformation is independent of $\lambda \in [0, 1)$. Second, we need to store only a single accumulation buffer $\Delta\mathbf{E}$. Finally, this approach exposes a single, fairly intuitive parameter; for values of λ close to 0 the deformation happens very quickly, while larger values result in a slower fade-in—a more delayed response. 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 1. This normalization discards deformation that would cause us to extrapolate outside the example space.

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.

3.3 Dynamics

As outlined in Figure 2, object dynamics are computed using an unmodified rigid body simulator (BulletPhysics in our implementation [Coumans 2014]). We perform triangle based collision detection, using a coarse collision geometry. We log the impulses applied during each rigid body simulation timestep. In between timesteps, we use these impulses to deform the objects as described in Section 3.2.

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 effective coefficient of restitution, C_r, while objects are deforming. We found the following model worked well in practice: we set C_r to the minimum of the default value C_r^*; a slight increase over the previous timestep; and a decreased value after impact. Compactly,

$$C_r := \min\left(C_r^*, C_r + \mu\Delta t, \exp\left(-\nu ||\mathbf{\Delta E}||_f\right) C_r^*\right), \quad (15)$$

where μ and ν are user controls. The second term linearly increases C_r after collisions and avoids jittering artifacts. We experimented with several functions for the third, impact, term, such as a clamped linear falloff, and preferred the exponential function over the alternatives.

4 Authoring Simulation Assets

Algorithm 1 Authoring Pipeline

```
Load closed, manifold, surface mesh
Add and position desired handles
Automatically generate tetrahedral mesh
Automatically add additional handles
Automatically compute skinning weights
Iterate
    Use handles to create example pose
```

We leverage recent research contributions for rigging, and skinning to automate as much of the authoring pipeline as possible. An overview of the authoring process is given in Algorithm 1. Artists begin by loading a closed triangle mesh into our tool. Next, they position an initial set of control bones and handles within the mesh; these initial bones allow the artist to place bones in key areas that they plan to deform. Next, our authoring tool automatically generates a tetrahedral mesh that encloses and approximates the surface mesh. Then the artist can choose to automatically add an arbitrary number of additional bones from the surface of the tetrahedral mesh. Once the handles are positioned, skinning weights are automatically assigned to mesh vertices. The artist can then add as many example deformations as they wish.

We choose to use bounded biharmonic skinning weights [Jacobson et al. 2011], in part because of available open source code. These weights are computed through a constrained optimization, which requires a tetrahedral mesh of the object. As such, our pipeline currently makes use of the approximating tetrahedralization of Stuart et al. [Stuart et al. 2013]. However, our algorithm is completely oblivious to the particular choice of algorithm to compute the linear blend skinning weights. In our implementation, the tetrahedral mesh is also used to quickly generate the coarse collision geometry and to calculate approximate geodesic distances. We note that, unlike many methods that depend on tetrahedral meshes, these computations are not sensitive to mesh quality; our automatically generated meshes often contained inverted elements. Moreover, no part of our algorithm inherently requires a tetrahedralization; other auxiliary structures, e.g. a regular grid, could easily be used for these computations.

We found hand-placing bones in three-dimensional space to be somewhat onerous. Thus our tool will also add a specified number of bones chosen from the vertices of the tetrahedral mesh. The simple approach of creating a set of all vertices on the surface and iteratively removing one of the two that are closest in Euclidean distance worked surprisingly well and we did not consider additional approaches. For some examples we chose which of the two vertices to remove arbitrarily, for others we favored retaining the vertex closest to the bounding of the tetrahedral mesh. We found creating deformations with these bones to be especially intuitive. Of course artists may still add bones in regions of particular interest.

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.

Once the mesh is rigged and the characteristic deformations are created, the artist can perform a rigid body simulation. In addition to setting standard rigid body parameters, we expose six new parameters to artists: α, the scaling constant for mapping from displacements to changes in barycentric example-coordinates (Equation (8)); β, the threshold for plastic deformation (Equation (4)); γ

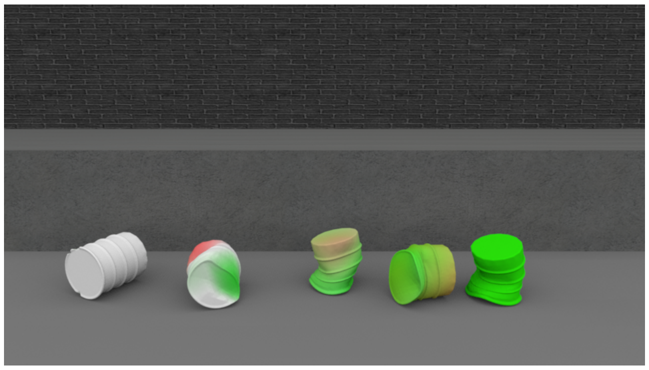

Figure 3: *The kernel radius, γ, controls how far deformations propagate. Smaller values generate denting behavior, while larger values result in deformations that more closely match the example poses.*

the deformation radius (Equation (11); λ, the plasticity rate (Equation (13)); and μ and ν, the parameters that control the coefficient of restitution (Equation (15)). α and β are straightforward to tune, for example by running a rigid body simulation and recording the maximum impulses. γ determines how far deformations propagate and is likewise straightforward to tune. The other parameters have more subtle effects, and we achieved reasonable results without much tuning. Since rigid body simulation is fast (most of our examples required less than one minute to run), the artist is able to interactively edit these parameters and even change the characteristic deformations to create the desired behaviors.

5 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 3 demonstrates the effect of varying the kernel radius, γ. In Figure 4 we show the range of deformations that can be achieved from barrels with identical material parameters and example poses as they fall through a Pachinko machine.

To demonstrate the potential of our approach to use arbitrary deformations as input to our system, we performed a thin-shell spring-mass simulation of our barrel model and used the output to create an example deformation for our system. Specifically, we set the positions of bones in our automatically generated rig to the positions of the corresponding vertices in the simulation output. The result is shown in Figure 5. Figure 6 demonstrates how, if a given set of examples doesn't produce the desired results, additional examples can be added to improve them.

We also animated five more complex scenes that may appear in movies or games. In the first, a car is pummeled by cannonballs before another car falls from the sky (see Figure 7). The deformations of the car change over time as the best example from the most recent impulse changes. When the second car falls from the sky, the first car assumes the flattened (red) pose almost everywhere. In the second, a reckless driver 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. Figure 8 shows how the example weights, $\mathbf{E}$, vary over the object meshes.

We also created a scene with multi-car pile-up crashes shown in Figure 1. Stunt drivers use ramps to become airborne and collide in midair. The cars become progressively more and more damaged as they collide.

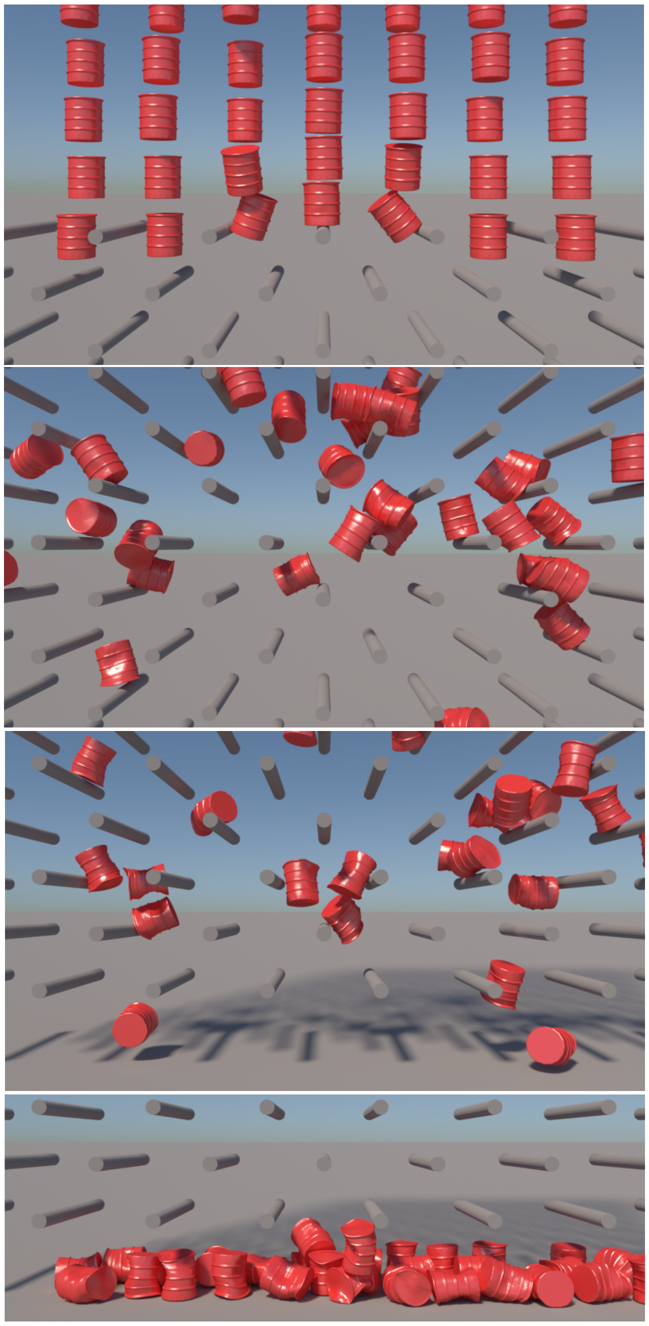

Figure 4: *Barrels deform as they fall through a Pachinko machine.*

We adopt the classic and practical pre-scoring approach to fracture. During the authoring phase, the artist breaks an object up into pieces, which are then loaded into the rigid body simulator with constraints that shared faces coincide. During simulation when a constraint force exceeds a threshold, the constraint is removed. In this case, we use the full volumetric tetrahedral mesh as collision geometry.

Figure 9 shows a spaceship crashing into a shipping yard, causing the ship and one of the cranes to break apart. Finally, we animate a scene from a space battle. A fleet of small ships crashes into a

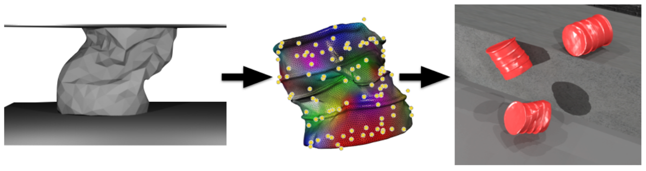

Figure 5: *Example deformations can be extracted from simulation data. We computed bone transforms matching this frame from mass-spring simulation. The resulting simulation uses only a single deformed example pose.*

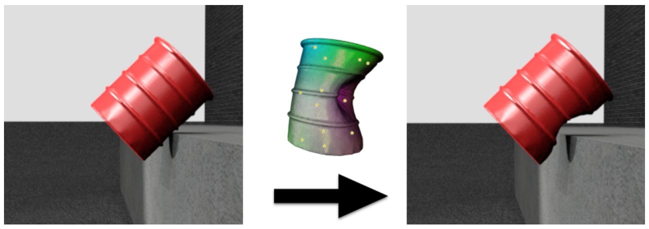

Figure 6: *The original example poses did not contain any deformations on the side of the barrel that hit the ledge. By adding an additional example, we can improve the quality of the resulting animation.*

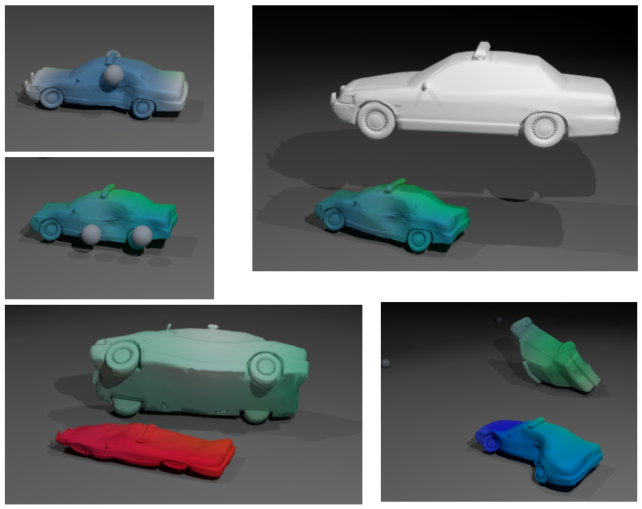

Figure 7: *A car is buffeted with cannonballs, and then a larger car is dropped onto it. The colors indicate example blend weights,* $\mathbf{E}$*; notice how they vary over the surface and change over time. In the bottom left frame, a flattened example deformation was included and the object largely assumes this example shape. In the bottom right frame, without the flattened example, a different shape emerges.*

larger one, causing significant 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 10.

Table 1 gives some representative statistics for our simulation assets.

Performance The performance of our method is closely tied to the performance of the underlying rigid body simulator. Depending on the scene, our deformation method adds 42% to 59% overhead per timestep on average (excluding the scene in Figure 10). However, our approach is generally slower than typical rigid body simulations, largely due to the fact that we use non-convex triangle meshes as collision geometry. Because our meshes change over time, it is impractical to employ the common trick of using approximate collision geometry made up of simpler primitives (e.g., a set of convex hull primitives), as these would need to be frequently rebuilt. Triangle meshes lead not only to more expensive collision detection, but to reduced timesteps; with framerate timesteps we observed artifacts, such as tangled meshes, in the rigid body simulation. Thus, while we did not observe any issues with our deformation model at framerate timesteps, our simulations take several timesteps per frame.

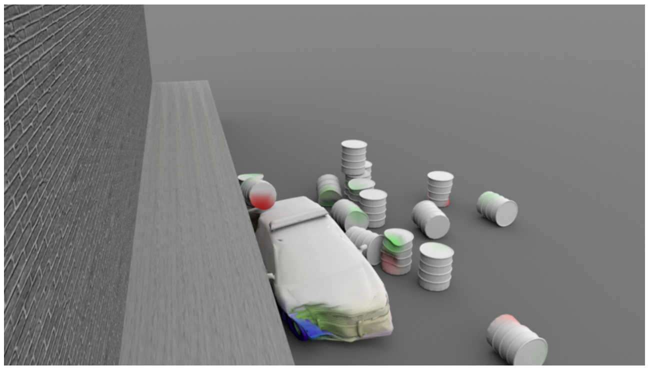

Figure 8: *Colors represent the matrix,* $\mathbf{E}$*, showing how the example weights vary over the objects. White vertices are undeformed; colored vertices correspond to the deformed examples.*

Table 1: *Simulation Asset Statistics*

Object	N	B	E
Barrel (Figure 4)	1790	12	3
Car (Figures 1 and 8)	4695	32	6
Crane (Figure 9)	1939	14	3
Big Spaceship (Figure 10)	4178	64	6
Little Spaceship (Figure 10)	1188	12	3

Despite this, several of our examples run in real-time, while the rest are fast enough to allow artists to interactively tune parameters.

The examples in this paper were run on a 2013 Macbook Pro with an integrated GPU. The projection and propagation were parallelized on the CPU while skinning was performed on the GPU. We made straightforward optimizations, such as avoiding re-skinning objects when $\Delta\mathbf{E}$ becomes negligible, but we did not fully optimize our code. Detailed timing results are shown in Table 2. Figure 11 shows the per-frame computational cost of rigid body simulation and deformation computations for the scene in Figure 4.

Table 2: *Timing breakdown for pictured examples. Timings below are the total time spent for each scene, in seconds*

Scene	Δt	RB Solver	Deformation	Average FPS
Figure 1	4e-3	54.2	6.8	54.4
Figure 8	5e-3	17.3	8.8	13.79
Figure 4	4e-3	64.7	47.8	10.6
Figure 9	2e-3	381	160	1.1
Figure 10	4e-3	43.2	53.1	6.2
Figure 7	4e-3	0.6	2.4	197.1

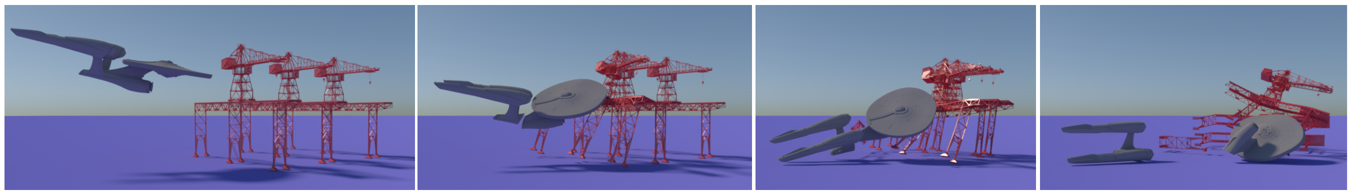

Figure 9: *A spaceship crashes through a shipping yard.*

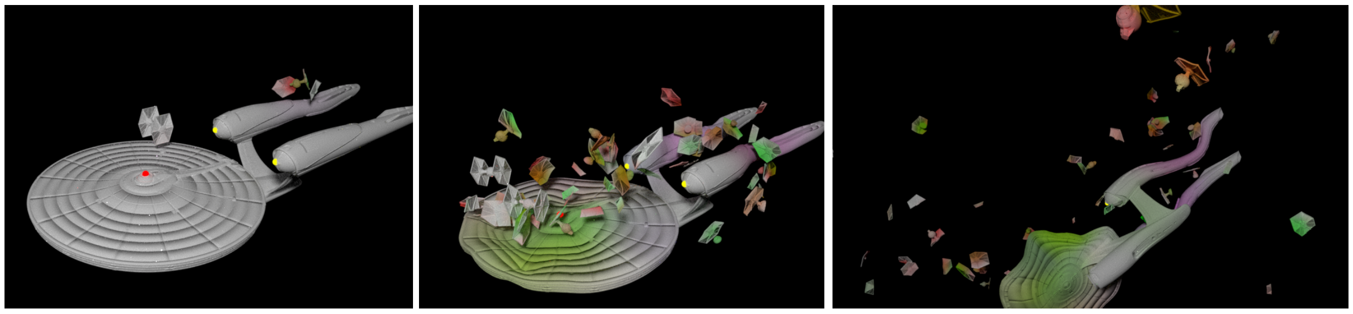

Figure 10: *A small fleet of spaceships crashes into an enemy vessel.*

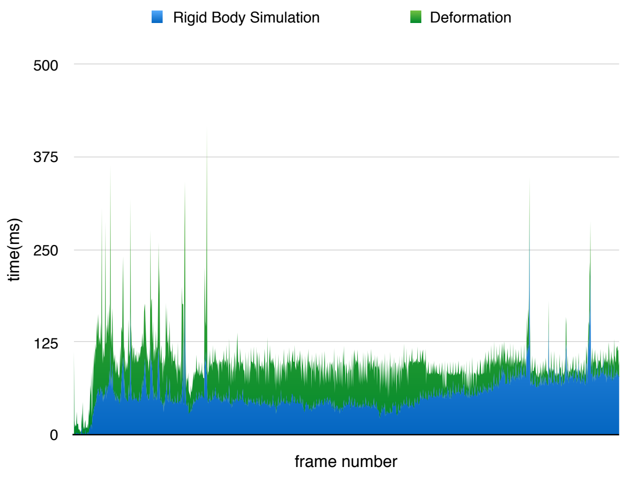

Figure 11: *Per-frame timing breakdown of the scene in Figure 4 (ms per 60 Hz frame).*

Artist Experience To help evaluate the usability of our method, we asked a digital art student to create a scene using an early version of our tools. We provided the artist with a version of the scene in Figure 9 with only rigid bodies and worked with him to add deformations using our approach. Over a two hour session, the artist authored example deformations and we worked together to tune parameters to achieve the final result. With a more polished user interface, we expect this task could have been accomplished in less than one hour (iterating with our prototype requires editing text files, rerunning the simulation, and reloading the output viewer).

The feedback we received form the artist was generally positive. About our deformation model, he said "for non-physical deformation, it really looks like physics." He also liked that he could author a "worst-case-scenario" deformation and have our simulator deform continuously toward that pose, even supporting minor deformations with the style of the provided example. He noted that this would be useful for authoring a single asset and reusing it many times, as he did for the cranes in the scene. He also liked the ability to add additional examples after seeing where objects collide during a simulation and have the new examples blend naturally with existing poses. The short runtimes of the method allowed him to iterate quickly to tune parameters and refine the simulation.

We did encounter some problems during the experience. When the example poses contained large rotations and objects underwent strong impulses, our system sometimes created self-intersections in the mesh. This was mitigated by adjusting the kernel radius, γ, and modifying the example pose. Without experience using our simulator, tuning simulation parameters was challenging at first; however, after some experimentation, he began to gain intuition for how adjusting them affected the resulting deformations.

Limitations and Future Work Like all example-based methods, the quality of our results is closely tied to the quality of the input examples. Our rigs based on bounded biharmonic weights are quite powerful, supporting bones, control handles, and cages. But, linear blend skinning is a reduced deformation model that, in the hands of novice users, tends to produce smooth deformations that appear more like volumetric modeling clay than the steel thin shells of which real-world objects are made. The detail present in Figure 5, where the example deformations are created from a spring-mass system, suggests that this is not an inherent limitation of linear blend skinning rigs. However, our automatically generated rig was only able to approximate the shapes in the simulation output. A very promising area of future work is improved automatic rig generation. A method that, given a set of arbitrary example deformations, produces a rig that can approximate them well would be very valuable. Such a technique would not only allow simulation output to be used as examples, but would free artists to use any tool they like to create examples. Kavan and colleagues [2010] might provide some guidance in this direction.

Relatedly, 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. It would also be very interesting to integrate our approach into production pipelines and to explore the "layering" approaches that are typically used. Our method should fit very well into this framework, allowing artists to add additional dents and folds, perhaps through methods such as wrinkle meshes [Müller and Chentanez 2010]. Such layering approaches is another avenue that would help address the lack of high frequency detail in our examples. Of course, it would also be interesting to see secondary effects like dust, smoke, and explosions.

Because the space of deformations used in our system is so large 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.

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 (8) would strike a balance between the input examples. It would be interesting to explore ways of computing the example 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.

Our method targets extreme destructive scenes. Scenes that involve deformation that occurs slowly over time or that contain many small impulses, would require enhancing our plasticity model to handle creep and perhaps a different approach to the plastic yield condition in Equation (4).

The underlying examples have a complex relationship with the final animation. For example, changing an example deformation could change the trajectories in the resulting rigid body simulation. More direct artistic control over the deformation is an interesting area of 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, easily implemented, and integrates smoothly into existing pipelines.

Acknowledgments

The authors are especially thankful to the anonymous reviewers for their time and helpful comments. Their efforts helped dramatically improve this work. This work was supported in part by National Science Foundation awards IIS-1314896 and IIS-1314813 and by the ERC project ERC-2014-StG-637014 *realFlow*. We want to thank P. Jutzi for the car model, Ron Romero for work on our pipeline, and digital art student Daniel Blair for 3D-modeling assistance and for experimenting with our prototype system.

References

ABE, Y., AND POPOVIĆ, J. 2006. Interactive animation of dynamic manipulation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 195–204.

BAO, Z., HONG, J.-M., TERAN, J., AND FEDKIW, R. 2007. Fracturing rigid materials. *IEEE Trans. Vis. Comput. Graph. 13*, 2 (Mar.), 370–378.

BARBIČ, J., AND JAMES, D. L. 2005. Real-time subspace integration for St. Venant-Kirchhoff deformable models. *ACM Trans. Graph. 24*, 3, 982–990.

BARGTEIL, A. W., WOJTAN, C., HODGINS, J. K., AND TURK, G. 2007. A finite element method for animating large viscoplastic flow. *ACM Trans. Graph. 26*, 3, 16.

BEAMNG, 2016. http://www.beamng.com. accessed January 19, 2016.

BOUAZIZ, S., MARTIN, S., LIU, T., KAVAN, L., AND PAULY, M. 2014. Projective dynamics: Fusing constraint projections for fast simulation. *ACM Trans. Graph. 33*, 4 (July), 154:1–154:11.

BUDSBERG, J., ZAFAR, N. B., AND ALDÉN, M. 2014. Elastic and plastic deformations with rigid body dynamics. In *ACM SIGGRAPH Talks*, 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. *ACM Trans. Graph. 32*, 2 (Apr.), 17:1–15.

CLAVET, S., BEAUDOIN, P., AND POULIN, P. 2005. Particle-based viscoelastic fluid simulation. In *Proccedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 219–228.

COUMANS, E., 2014. Bullet physics library. http://bulletphysics.org/.

CRISWELL, B., LENTINE, M., AND SAUERS, S. 2010. Avatar: Bending rigid bodies. In *ACM SIGGRAPH 2010 Talks*.

CRISWELL, B., SMITH, J., AND DEUBER, D. 2010. Transformers 2: Breaking buildings. In *ACM SIGGRAPH Talks*.

DE AGUIAR, E., SIGAL, L., TREUILLE, A., AND HODGINS, J. K. 2010. Stable spaces for real-time clothing. *ACM Trans. Graph. 29*, 4, 106:1–106:9.

FAN, Y., LITVEN, J., LEVIN, D. I. W., AND PAI, D. K. 2013. Eulerian-on-lagrangian simulation. *ACM Trans. Graph. 32*, 3, 22:1–22:9.

FAURE, F., GILLES, B., BOUSQUET, G., AND PAI, D. K. 2011. Sparse meshless models of complex deformable solids. *ACM Trans. Graph. 30*, 4, 73:1–73:10.

FENG, W.-W., YU, Y., AND KIM, B.-U. 2010. A deformation transformer for real-time cloth animation. *ACM Trans. Graph. 29*, 4, 108:1–108:9.

GILLES, B., BOUSQUET, G., FAURE, F., AND PAI, D. K. 2011. Frame-based elastic models. *ACM Trans. Graph. 30*, 2, 15:1–15:12.

HAHN, F., THOMASZEWSKI, B., COROS, S., SUMNER, R. W., COLE, F., MEYER, M., DEROSE, T., AND GROSS, M. 2014. Subspace clothing simulation using adaptive bases. *ACM Trans. Graph. 33*, 4, 105:1–105:9.

HARMON, D., AND ZORIN, D. 2013. Subspace integration with local deformations. *ACM Trans. Graph. 32*, 4 (July), 107:1–107:10.

HIROTA, K., TANOUE, Y., AND KANEKO, T. 1998. Generation of crack patterns with a physical model. *The Visual Computer 14*, 3, 126–137.

HIROTA, K., TANOUE, Y., AND KANEKO, T. 2000. Simulation of three-dimensional cracks. *The Visual Computer 16*, 7, 371–378.

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*, 131–140.

JACOBSON, A., BARAN, I., POPOVIĆ, J., AND SORKINE, O. 2011. Bounded biharmonic weights for real-time deformation. *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. *ACM Trans. Graph. 31*, 4, 77:1–77:10.

JONES, B., POPOVIĆ, J., MCCANN, J., LI, W., AND BARGTEIL, A. 2013. Dynamic sprites. In *Proceedings of the ACM SIGGRAPH Conference on Motion in Games*.

JONES, B., WARD, S., JALLEPALLI, A., PERENIA, J., AND BARGTEIL, A. W. 2014. Deformation embedding for point-based elastoplastic simulation. *ACM Trans. Graph. 33*, 2, 21:1–21:9.

KAVAN, L., AND ZARA, J. 2005. Spherical blend skinning: A real-time deformation of articulated models. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ACM Press, 9–16.

KAVAN, L., SLOAN, P.-P., AND O'SULLIVAN, C. 2010. Fast and efficient skinning of animated meshes. *Comput. Graph. Forum 29*, 2, 327–336.

KAVAN, L., GERSZEWSKI, D., BARGTEIL, A., AND SLOAN, P.-P. 2011. Physics-inspired upsampling for cloth simulation in games. *ACM Trans. Graph. 30*, 4, 93:1–93:9.

KIM, T.-Y., AND VENDROVSKY, E. 2008. Drivenshape: a data-driven approach for shape deformation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Eurographics Association, 49–55.

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*, 19–24.

LEVINE, J. A., BARGTEIL, A. W., CORSI, C., TESSENDORF, J., AND GEIST, R. 2014. A peridynamic perspective on spring-mass fracture. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*.

LIU, T., BARGTEIL, A. W., O'BRIEN, J. F., AND KAVAN, L. 2013. Fast simulation of mass-spring systems. *ACM Trans. Graph. 32*, 6, 214:1–214:7.

MARTIN, S., KAUFMANN, P., BOTSCH, M., GRINSPUN, E., AND GROSS, M. 2010. Unified simulation of elastic rods, shells, and solids. *ACM Trans. Graph. 29*, 39:1–39:10.

MARTIN, S., THOMASZEWSKI, B., GRINSPUN, E., AND GROSS, M. 2011. Example-based elastic materials. *ACM Trans. Graph. 30*, 4, 72:1–72:8.

MOLINO, N., BAO, Z., AND FEDKIW, R. 2004. A virtual node algorithm for changing mesh topology during simulation. *ACM Trans. Graph. 23*, 3, 385–392.

MÜLLER, M., AND CHENTANEZ, N. 2010. Wrinkle meshes. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 85–92.

MÜLLER, M., AND CHENTANEZ, N. 2011. Solid simulation with oriented particles. *ACM Trans. Graph. 30*, 4 (July), 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 *Proceedings of the Eurographics Workshop on Computer Animation and Simulation*, 113–124.

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*, 141–151.

MÜLLER, M., HEIDELBERGER, B., TESCHNER, M., AND GROSS, M. 2005. Meshless deformations based on shape matching. *ACM Trans. Graph. 24*, 3, 471–478.

MÜLLER, M., HEIDELBERGER, B., HENNIX, M., AND RATCLIFF, J. 2007. Position based dynamics. *J. Vis. Comun. Image Represent. 18*, 2, 109–118.

NORTON, A., TURK, G., BACON, R., GERTH, J., AND SWEENEY, P. 1991. Animation of fracture by physical modeling. *The Visual Computer 7*, 4, 210–219.

O'BRIEN, J. F., AND HODGINS, J. K. 1999. Graphical modeling and animation of brittle fracture. In *The Proceedings of ACM SIGGRAPH*, 137–146.

O'BRIEN, J. F., BARGTEIL, A. W., AND HODGINS, J. K. 2002. Graphical modeling and animation of ductile fracture. *ACM Trans. Graph. 21*, 3, 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*, 156–166.

PATKAR, S., AANJANEYA, M., BARTLE, A., LEE, M., AND FEDKIW, R. 2014. Efficient Denting and Bending of Rigid Bodies. In *Eurographics/ ACM SIGGRAPH Symposium on Computer Animation*.

PAULY, M., KEISER, R., ADAMS, B., DUTRÉ;, P., GROSS, M., AND GUIBAS, L. J. 2005. Meshless animation of fracturing solids. *ACM Trans. Graph. 24*, 3, 957–964.

PRATT, J., CHEW, C.-M., TORRES, A., DILWORTH, P., AND PRATT, G. 2001. Virtual model control: An intuitive approach for bipedal locomotion. *The International Journal of Robotics Research 20*, 2, 129–143.

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*, 1–8.

STAUNING, O., AND BENDTSEN, C., 2007. Fadbad++, flexible automatic differentiation using templates and operator overloading in c++. http://www.fadbad.com.

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*.

SU, J., SCHROEDER, C., AND FEDKIW, R. 2009. Energy stability and fracture for frame rate rigid body simulations. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 155–164.

TERZOPOULOS, D., AND FLEISCHER, K. 1988. Modeling inelastic deformation: Viscoelasticity, plasticity, fracture. In *The Proceedings of ACM SIGGRAPH*, 269–278.

TERZOPOULOS, D., PLATT, J., BARR, A., AND FLEISCHER, K. 1987. Elastically deformable models. *SIGGRAPH Comput. Graph. 21*, 4, 205–214.

WANG, H., HECHT, F., RAMAMOORTHI, R., AND O'BRIEN, J. 2010. Example-based wrinkle synthesis for clothing animation. *ACM Trans. Graph. 29*, 4, 107:1–107:8.

WEINSTEIN, R., PETTERSON, F., AND CRISWELL, B. 2008. Destruction system. In *ACM SIGGRAPH Talks*, 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. *ACM Trans. Graph. 29*, 49:1–49:11.

WOJTAN, C., AND TURK, G. 2008. Fast viscoelastic behavior with thin features. *ACM Trans. Graph. 27*, 3, 1–8.

WOJTAN, C., THÜREY, N., GROSS, M., AND TURK, G. 2009. Deforming meshes that split and merge. *ACM Trans. Graph. 28*, 3, 76:1–76:10.

XU, W., UMENTANI, N., CHAO, Q., MAO, J., JIN, X., AND TONG, X. 2014. Sensitivity-optimized rigging for example-based real-time clothing synthesis. *ACM Trans. Graph. 33*, 4, 107:1–107:11.

YANG, Y., LI, D., XU, W., TIAN, Y., AND ZHENG, C. 2015. Expediting precomputation for reduced deformable simulation. *ACM Trans. Graph. 34*, 6, 243:1–243:13.

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 Talks*, 25:1–25:1.