Breaking Good: Fracture Modes for Realtime Destruction

SILVIA SELLÁN, University of Toronto
JACK LUONG, California State University, Fresno and University of California, Los Angeles
LETICIA MATTOS DA SILVA, University of California, Los Angeles and Massachusetts Institute of Technology
ARAVIND RAMAKRISHNAN, University of Maryland at College Park and University of Toronto
YUCHUAN YANG, University of California, Los Angeles
ALEC JACOBSON, University of Toronto

Vibration modes = a shape's natural deformation patterns

Our **Fracture** modes = a shape's natural **breaking** patterns

Fig. 1. Drawing an analogy with the well-studied vibration modes, we define a shape's *fracture modes*, which we can precompute for realtime applications.

Drawing a direct analogy with the well-studied vibration or elastic modes, we introduce an object's *fracture modes*, which constitute its preferred or most natural ways of breaking. We formulate a sparsified eigenvalue problem, which we solve iteratively to obtain the n lowest-energy modes. These can be precomputed for a given shape to obtain a prefracture pattern that can substitute the state of the art for realtime applications at no runtime cost but significantly greater realism. Furthermore, any realtime impact can be projected onto our modes to obtain impact-dependent fracture patterns without the need for any online crack propagation simulation. We not only introduce this theoretically novel concept, but also show its fundamental and practical superiority in a diverse set of examples and contexts.

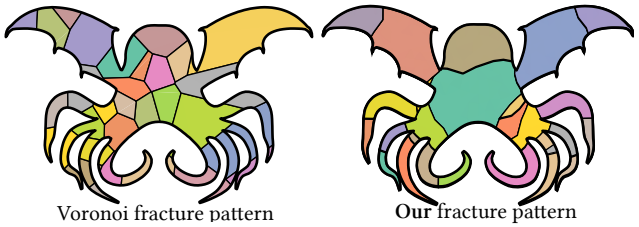
1 INTRODUCTION

The patterns and fragmentations formed by an object undergoing brittle fracture add richness and realism to destructive simulations. Unfortunately, existing methods for producing the most realistic fractures require hefty simulations too expensive for many realtime applications. An attractive and popular alternative is to rely on precomputed fragmentation patterns at the modeling stage that can be swapped in at run-time when an impact is detected. Existing prefracture methods use geometric heuristics that produce unrealistic patterns oblivious of an object's elastic response profile or structural weaknesses (see Figs. 2, 3 and 5). Geometric patterns alone also do not answer *which* fragments should break-off for a given impact at run-time, inviting difficult to tune heuristics or complete fracture regardless of the impact. As a result, video game studios may rely on artist-authored fragmentation patterns.

In this paper, we present a method for prefracturing which draws a direct analogy to a solid shape's elastic vibration modes. We compute a shape's *fracture modes*[1], which algebraically span the shape's natural ways of breaking apart. By introducing a continuity objective under a sparsity-inducing norm to the classic vibration modes optimization problem, we identify unique and orthogonal modes of fracture in increasing order of a generalized notion of frequency.

The first k fracture modes can by intersected against each other to define a prefracture pattern as a drop-in replacement to existing procedural methods (see Fig. 2). Furthermore, impacts determined at runtime can be efficiently projected onto the linear space of precomputed fracture modes to obtain impact-dependent fracture without the need for costly stress computation or crack propagation.

We demonstrate the superiority of our algorithm over existing procedural methods and evaluate its accuracy by qualitatively comparing to existing works in worst-case structural analysis. We showcase the benefits of our algorithm within an off-the-shelf rigid body simulator to produce animations on a diverse set of shapes and impacts. We show the potential of fracture modes for realtime applications with a prototypical interactive 2D application (see Fig. 4 and accompanying video).



Voronoi fracture pattern

Our fracture pattern

Fig. 2. Popular Voronoi-based prefracturing results in recognizable, unrealistic shapes. Our fracture modes break across weak regions.

[1]Not to be confused with the "three modes of fracture" [Irwin 1957].

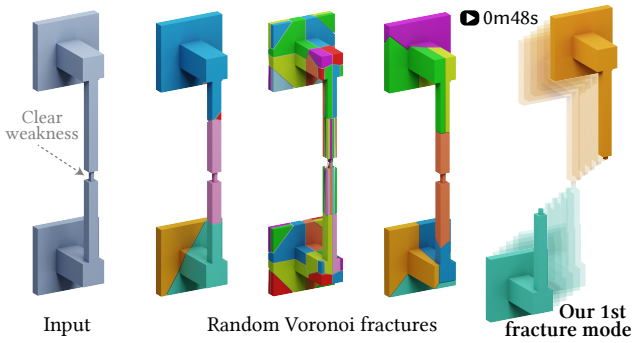


Fig. 3. Existing procedural prefracture algorithms (center) rely on randomness and do not account for the geometrically weak regions of an object, unlike our proposed fracture modes (right).

2 RELATED WORK

Fracture simulation has an extensive body of previous work; Muguercia et al. [2014] provide a thorough survey. Brittle fracture is characterized when little or no perceptible deformation occurs before fracture (i.e., when the object is otherwise rigid). Modeling the dynamic or quasistatic *growth* of brittle fracture patterns requires simulation at the timescale of cracktip propagation, modeling stress concentration and subsequent release. This process has been approximated, for example, using mass-springs [Hirota et al. 1998; Norton et al. 1991], finite elements [Kaufmann et al. 2009; Koschier et al. 2015; O'Brien and Hodgins 1999; Pfaff et al. 2014; Wicke et al. 2010], boundary elements [Hahn and Wojtan 2015, 2016; Zhu et al. 2015], and the material-point method [Wolper et al. 2020, 2019]. While any such method can eventually meet realtime demands by lowering the discretization fidelity (e.g., on low-res cage geometry [Parker and O'Brien 2009] or a modal subspace [Glondu et al. 2012]) or assuming large enough computational resources, we instead focus our attention to previous methods which achieve realtime performance via the well established workflow of prefracturing. This workflow sidesteps computationally expensive and numerically fragile remeshing operations. It fits tidily into the existing realtime graphics pipeline, where geometric resolutions and computational resources can be preallocated to ensure low latency and consistent performance.

Creating prefracture patterns manually requires skill and time, precluding fully automatic pipelines. Many commercial packages (e.g., Unreal Engine, Houdini) implement or suggest geometric prefracturing heuristics to segment a shape into solid subfragments. Voronoi diagrams of randomly scattered points [Oh et al. 2012; Raghavachary 2002; Schvartzman and Otaduy 2014] capture the stochastic quality of fracture, but result in overly regular and convex fragments with perfectly flat sides. Despite lacking realism, convexity can be an advantage for realtime collision detection, and approximate convex decomposition has been employed as prefracturing technique [Müller et al. 2013]. The regularity of these fragments can be augmented by randomly generated "cutter" objects (i.e., bumpy planes slicing though the input shape) or perturbed level set functions (see, e.g., [Museth et al. 2021]). These stochastic methods often miss obvious structural weaknesses (see Fig. 3) or result in implausible fragment shapes (see Fig. 5). While the defects

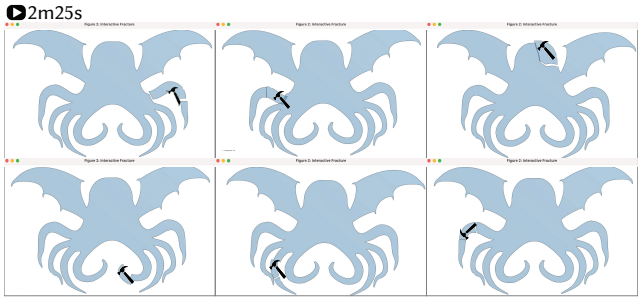


Fig. 4. Screenshots of our 2D interactive prototype, in which the user can select the impact position to obtain different breaking patterns.

of these methods can be hidden behind destruction dust or obscured by fast explosions, our fracture modes consistently produce non-convex fragments whose boundaries originate from minimal stress displacements of the shape. Beyond taking into account the physical elastic behavior of the geometry, our method can easily incorporate constraints to avoid fractures in certain areas. While already compatible with previous impact heuristics such as Euclidean distance thresholds [Müller et al. 2013], our fracture modes span a linear subspace onto which impacts can be cheaply projected to trigger fragment displacements at runtime.

2.1 Sparsified eigenproblems

Our approach for defining fracture modes lies within a broader class of optimization problems of the form:

$$\underset{\mathbf{X}^\top \mathbf{M}\mathbf{X}=\mathbf{I}}{\operatorname{argmin}} \quad \frac{1}{2}\operatorname{trace}\left(\mathbf{X}^\top \mathbf{L}\mathbf{X}\right) + \sum_{i=1}^{r} g(\mathbf{X}_i) \tag{1}$$

where $\mathbf{X}_i$ as the i^{th} column of $\mathbf{X} \in \mathbb{R}^{n\times k}$ is referred to as a modal vector or mode, $\mathbf{M}$ and $\mathbf{L}$ are positive semi-definite $n \times n$ matrices, and $g : \mathbb{R}^n \to \mathbb{R}$ is a sparsity-inducing norm, such as $g(\mathbf{x}) = \|\mathbf{x}\|_1$. When $g := 0$, this reduces to the common *generalized eigenvalue problem* whose solution satisfies $\mathbf{L}\mathbf{X} = \mathbf{M}\mathbf{X}\Lambda$, where the diagonal $k \times k$ matrix Λ contains the k smallest eigenvalues ($\Lambda_{i,i} = \mathbf{X}_i^\top \mathbf{L}\mathbf{X}_i$). For non-trivial g, we may continue to consider $\lambda_i = \mathbf{X}_i^\top L\mathbf{X}_i + g(\mathbf{X}_i)$ as describing the frequency of the i^{th} mode.

Ozolins et al. [2013] proposed the notion of compressed modes using a sparsity-inducing ℓ_1-norm to compute localized (sparse) solutions to Schrödinger's equation. Neumann et al. [2014] extended this idea to compressed eigenfunctions of the Laplace-Beltrami operator on 3D surfaces, advocating for an alternating direction method of multipliers (ADMM) optimization method. While ADMM's standard convergence guarantees [Boyd 2010] do not apply to non-convex problem such as Eq. (1), Neumann et al. [2014] demonstrate successful local convergence albeit with dependency on the initial guess and optimization path. Replacing the $\mathbf{X}^\top \mathbf{L}\mathbf{X}$ term with a data-term, Neumann et al. [2013] use a similar ADMM approach to create *sparse PCA* bases for mesh deformations.

Brandt and Hildebrandt [2017] further extend this line of smooth, sparse modal decompositions by considering L to be the Hessian of an elastic energy. They propose an iterative mode-by-mode optimization. The current mode is optimized by sub-iterations consisting of

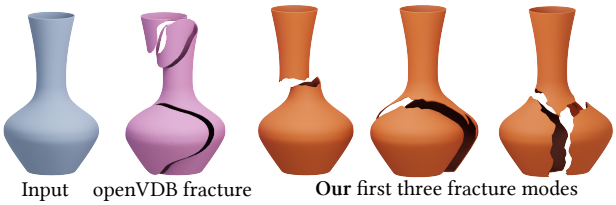


Fig. 5. Level-set methods (e.g., OpenVDB [Museth et al. 2021]) will produce non-convex, yet implausible fracture patterns unrelated to the structural integrity of the shape.

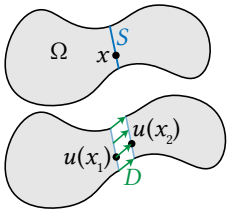
a quadratic program solve resulting from linearizing the constraints around the current iterant interleaved with normalization in order to approach a unit-norm vector. Despite the conspicuous downside that any sub-optimality of earlier modes is *locked in* possibly affecting the accuracy of later modes, this method enjoys performance and robustness improvements over the ADMM approach of Neumann et al. [2014]. Therefore, we follow suit with a similarly mode-by-mode fixed-point iteration approach. Unique to our method is that we do not consider the sparsity of the modal vector itself, but rather the sparsity of the mode's *continuity* over the domain.

3 FRACTURE MODES

Given an elastic solid object $\Omega \subset \mathbb{R}^d$ and a deformation map $u : \Omega \to \mathbb{R}^d$, we can formulate the object's total strain energy as

$$E_\Psi(u) = \int_\Omega \Psi(u, x)dx \tag{2}$$

where Ψ is the strain energy density function evaluated at points $x \in \Omega$ in the *undeformed* object.



Suppose we allow the deformation map u to fracture the object Ω into two disjoint pieces Ω_1 and Ω_2 along a *given* $(d-1)$-dimensional fracture fault S (see inset). Effectively, we're allowing u to be discontinuous at S. Consider x_1 and x_2 to be the undeformed positions of points infinitesimally on either side of a point $x \in S$ on the fracture fault. Then, the difference between $u(x_1)$ and $u(x_2)$ describes the pointwise *vector-valued discontinuity* at $x \in S$:

$$D(u, x \in S) = u(x_1) - u(x_2) \in \mathbb{R}^d, \tag{3}$$

where $D = \mathbf{0}$ would indicate continuity or absence of fracture.

We now consider that the set of admissible discontinuities is not just a single fracture fault, but a finite number of fault patches: $S = \{S_1, \ldots, S_p\}$. We assume that S comes from a, for now, arbitrary *over-segmentation* of Ω. This could be created with a high-resolution Voronoi diagram, by intersecting Ω with random surfaces, voxel boundaries, or by some *a priori* distribution of granular subobjects. We define the total *discontinuity energy* associated with u as

$$E_D(u) = \|D(u, S)\|_{2,1} := \sum_{i=1}^{p} \sqrt{\int_{S_i} D(u, x)^2 dx}. \tag{4}$$

We add this to the strain energy to form the *total energy*:

$$E(u) = E_\Psi(u) + \omega\, E_D(u), \tag{5}$$

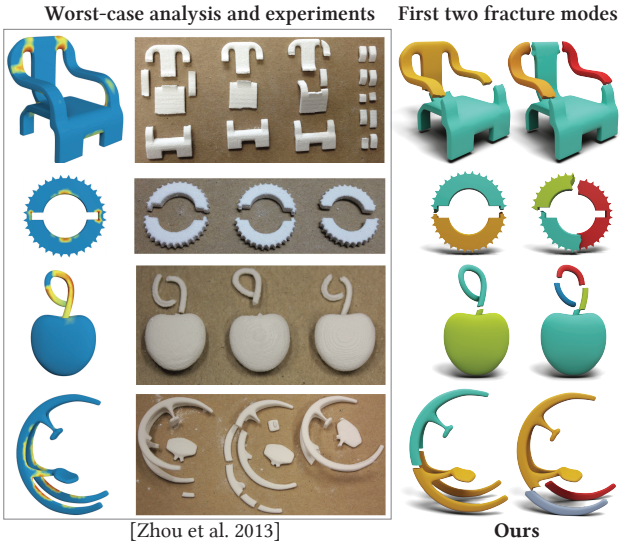


Fig. 6. Zhou et al. [2013] (left) identify the geometrically weakest regions of a given shape, which align with their real-world experiments (center). Our fracture modes (right) produce fracture patterns qualitatively similar.

where $\omega \in \mathbb{R}$ is a positive weight balancing the two terms. While a more common ℓ_1 norm would also encourage sparsity [Candes and Wakin 2008], the choice of $\ell_{2,1}$ norm ensures rotational invariance.

We can now define the k lowest energy *fracture modes* as the set of mass-orthonormal deformation maps $\{u^i\}_{i=1}^k$ that minimize their combined total energy; i.e.,

$$\{u^i\}_{i=1}^k = \underset{\{u^i\}_{i=1}^k}{\operatorname{argmin}} \sum_{i=1}^{k} E(u^i), \qquad \text{s.t.} \quad \int_\Omega (u^i)^\top \rho u^j dx = \delta^{i,j}, \tag{6}$$

where ρ is the local mass density and $\delta^{i,j}$ is the Kronecker delta. In general, for large enough ω, minimizers u^i will have exactly zero E_D on all but a sparse subset of fault patches in S, agreeing with the usual sparse coding theory [Candes and Wakin 2008].

3.1 Fracture Modes on Meshes

We will derive a discrete formulation of the variational problem in Eq. (6) for a 2D solid object, represented as a triangle mesh Ω with n vertices and m faces. In this construction, the mesh's p *interior* edges will correspond to admissible fracture faults $S_1, \cdots, S_p$. Everything that follows is straightforward to extend to 3D solids by exchanging triangles and interior edges for tetrahedra and interior faces.

Traditional piecewise-linear finite element method (FEM) would discretize the strain energy E_Ψ using hat functions $\varphi_i : \Omega \to \mathbb{R}$, $\forall i = 1, \ldots, n$ and associate a scalar function $u : \Omega \to \mathbb{R}$ with a vector $\mathbf{u} \in \mathbb{R}^n$ such that

$$u(x) = \sum_{i=1}^{n} \mathbf{u}_i \varphi_i(x). \tag{7}$$

Vector-valued $u : \Omega \to \mathbb{R}^d$ such as a deformation would be described coordinate-wise in the same way, via a vector $\mathbf{u} \in \mathbb{R}^{dn}$.

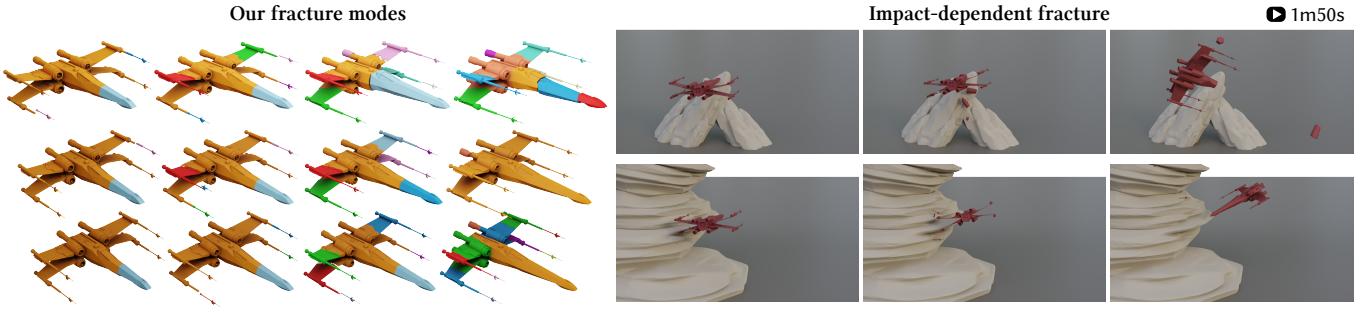
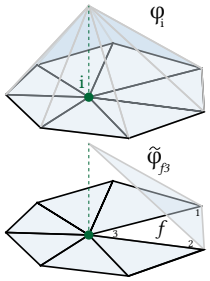


Fig. 7. The same prefractured modes can be used to simulate many different impacts; for example, a racing simulator may destroy the player's spaceship differently depending on its impact.



Hat functions are by construction continuous. Normally, this is a good thing, but we would like to have functions with arbitrarily large co-dimension one patches of discontinuities. Let us introduce the concept of an *exploded* mesh $\tilde{\Omega}$, with the same m faces as Ω and same carrying geometry, but where each vertex is effectively repeated for each incident triangle. Thus, $\tilde{\Omega}$ is composed of m combinatorially disconnected triangles and $3m$ vertices v_{cf} with $c = 1, 2, 3$ and $f = 1, \ldots, m$.

Hat functions $\tilde{\varphi} : \tilde{\Omega} \to \mathbb{R}$ defined on $\tilde{\Omega}$ reduce to barycentric coordinate functions extended with zero value outside the corresponding triangle (see inset). These trivially span the space of piecewise linear scalar *discontinuous* functions u via vectors $\mathbf{u} \in \mathbb{R}^{3m}$:

$$u(x) = \sum_{f=1}^{m} \sum_{c=1}^{3} \mathbf{u}_{cf} \tilde{\varphi}_{cf}(x). \tag{8}$$

We will use this basis for each coordinate of our vector-valued deformation map, captured in a vector of coefficients $\mathbf{u} \in \mathbb{R}^{(d+1)md}$, with the displacement at vertex c of face f selected by $\mathbf{u}_{cf} \in \mathbb{R}^d$.

We may now discretize both terms in our energy in Eq. (5). First, the integral strain energy follows the usual FEM discretization as a sum over each element.

$$E_\Psi(\mathbf{u}) = \sum_{f=1}^{m} \int_f \Psi(u, x) dx \,. \tag{9}$$

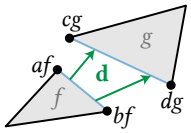
We defer the choice of Ψ until Section 3.7, and now only assume it can be approximated via Taylor expansion around the rest configuration by its Hessian matrix $\mathbf{Q} \in \mathbb{R}^{(d+1)md \times (d+1)md}$:

$$E_\Psi(\mathbf{u}) \approx \frac{1}{2} \mathbf{u}^\top \mathbf{Q} \mathbf{u} \,. \tag{10}$$

Our basis functions $\tilde{\varphi}$ will only allow fractures along the mesh's interior edges, therefore, the integral in Eq. (4) breaks into a contribution from each interior edge e:

$$E_e(u) = \sqrt{\int_e D(u, x)^2 dx} \,, \tag{11}$$

which we can compute exactly by two-point Gaussian quadrature (the integrand is a second-order polynomial).



For a given edge e with length l, corresponding to vertex pairs $\{af, bf\}$ *and* $\{cg, dg\}$ (see inset), this amounts to

$$E_e(\mathbf{u}) = \sqrt{\frac{l}{2} \left(\left\| \mathbf{d}\left(\tfrac{+1}{\sqrt{3}}\right) \right\|_2^2 + \left\| \mathbf{d}\left(\tfrac{-1}{\sqrt{3}}\right) \right\|_2^2 \right)}$$

where

$$\mathbf{d}(t) = \frac{1+t}{2}(\mathbf{u}_{af} - \mathbf{u}_{cg}) + \frac{1-t}{2}(\mathbf{u}_{bf} - \mathbf{u}_{dg}) \tag{12}$$

measures the pointwise discontinuity for the quadrature at parametric location $t \in [-1, 1]$ along the edge.

The full discontinuity energy associated with the map $u(x)$ is given by summing over every interior edge

$$E_D(\mathbf{u}) = \sum_{e=1}^{p} E_e(\mathbf{u}) \,. \tag{13}$$

Finally, we can define the k lowest-energy *discrete fracture modes* as column vectors of a matrix $\mathbf{U} \in \mathbb{R}^{(d+1)md \times k}$ satisfying

$$\underset{\mathbf{U}^\top \tilde{\mathbf{M}} \mathbf{U} = \mathbf{I}}{\operatorname{argmin}} \quad \frac{1}{2} \operatorname{trace}\left(\mathbf{U}^\top \mathbf{Q} \mathbf{U}\right) + \omega \sum_{i=1}^{k} \left(E_D(\mathbf{U}_i)\right) \tag{14}$$

where $\tilde{\mathbf{M}}$ is the possibly lumped FEM mass matrix defined on the exploded mesh $\tilde{\Omega}$.

3.2 Optimization

The definition of fracture modes on meshes involves solving the optimization problem in Eq. (14). While the objective term is convex, the orthogonality constraints are not. To proceed, we adapt the *Iterated Convexification for Compressed Modes* (ICCM) scheme proposed by Brandt and Hildebrandt [2017]. ICCM computes the modes sequentially, assuming the first $i-1$ columns of $\mathbf{U}$ have been computed. In the original ICCM formulation, the process for finding the i^{th} column, $\mathbf{U}_i$ proceeds by choosing a random unit-norm vector

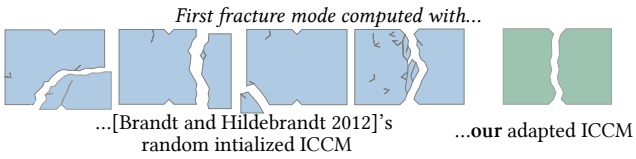


Fig. 8. We adapt the algorithm suggested by Brandt and Hildebrandt [2017] to use the eigenmodes of $\mathbf{Q}$ as initial guesses as opposed to random vectors.

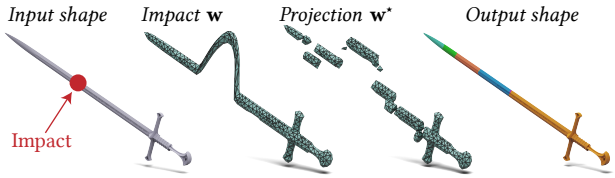


Fig. 9. When an impact is detected, we compute an *impact* vector $\mathbf{w}$ (center left) which we project onto our computed modes (center right) to obtain the final fracture pattern (right).

$\mathbf{c}$, then repeatedly solving

$$\mathbf{U}_i \leftarrow \underset{\mathbf{u}}{\operatorname{argmin}} \frac{1}{2}\mathbf{u}^\top \mathbf{Q}\mathbf{u} + \omega E_D(\mathbf{u}) \tag{15}$$

$$\text{subject to } \begin{bmatrix} \mathbf{U}_1^\top \\ \vdots \\ \mathbf{U}_{i-1}^\top \\ \mathbf{c}^\top \end{bmatrix} \tilde{\mathbf{M}}\mathbf{u} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} \tag{16}$$

and updating

$$\mathbf{c} \leftarrow \frac{\mathbf{U}_i}{\sqrt{\mathbf{U}_i^\top \tilde{\mathbf{M}}\mathbf{U}}}, \tag{17}$$

until convergence is detected by $\|\mathbf{U}_i - \mathbf{c}\|$ falling below some tolerance ε. By linearizing the (quadratic) norm constraint, the minimization problem Eq. (15) is a convex conic problem and solved with off-the-shelf techniques (see Appendix A).

We found that random initializations for $\mathbf{c}$ not only introduce non-determinism, but can also sometimes result in a large number of inner iterations and sub-optimal local minima (see Fig. 8). Instead, when computing $\mathbf{U}_i$ we initialize $\mathbf{c}$ with the i^{th} continuous eigenvectors of $\mathbf{Q}$ (defined on the unexploded mesh). These k initial vectors can be precomputed at once using a sparse eigen solver (such as `eigs` in MATLAB or ARPACK [Lehoucq et al. 1998]). We outline our complete fracture mode computation algorithm in Algorithm 1.

3.3 Impact-dependent fracture

By construction, the columns of $\mathbf{U}$ form an orthonormal basis of the lowest-energy k-dimensional subspace of possible fractures of Ω. This key feature means we can precompute an object's fracture modes right after its design. Then, inside an interactive application we can project any detected impact onto our modes to obtain *impact-dependent realtime fracture* (see Fig. 9).

If a collision is detected between Ω and another object, with contact point p and normal $\vec{n}$, we can define the exploded-vertex-wise impact vector $\mathbf{w} \in \mathbb{R}^{(d+1)md}$. Ideally, $\mathbf{w}$ would be the displacements

Algorithm 1: Fracture Modes via Adapted ICCM

```
Let Q be a PSD matrix, k ∈ ℕ
C ← eigenvectors(Q, M, k)
for i = 1, ..., k do
    c ← C_i
    repeat
        U_i ← solve Eq. (15) (see Appendix A)
        c ← U_i / sqrt(U_i^⊤ M̃ U_i)
    until ||U_i − c|| ≤ ε
return U
```

determined by an extremely short-time-duration simulation of elastic shock propagation. In lieu of being able to computing this in realtime, we use an approximation based on distance to smear the impact into the object:

$$\mathbf{w}_{cf} = G(\|p - v_{cf}\|)\,\vec{n}\,, \quad \forall c = 1,2,3\,,\ f = 1,\ldots,m \tag{18}$$

where G is a filter that vanishes as its argument grows; in our case, a Gaussian density function with standard deviation 0.05. Then, we project $\mathbf{w}$ onto our modes to obtain our *projected impact*

$$\mathbf{w}^\star = \sum_{i=1}^{k} \mathbf{U}_i \mathbf{U}_i^\top \tilde{\mathbf{M}}\mathbf{w} \tag{19}$$

Let $\tilde{\Omega}_{\mathbf{w}^\star}$ be the exploded mesh $\tilde{\Omega}$ as deformed by the map $w^\star$. For any two vertices v_{af}, v_{cg} that are coincident in $\tilde{\Omega}$ (i.e., that came from the same original vertex in Ω), we will *glue* (i.e., *un-explode*) them if their deformation maps differ by less than some tolerance, $\|\mathbf{w}^\star_{af} - \mathbf{w}^\star_{cg}\| < \sigma$. This results in a new *fractured* mesh $\Omega^\star$, whose fracture pattern depends meaningfully on the nature of the impact and which we can output to the simulation.

3.4 Efficient implementation for real-time fracture

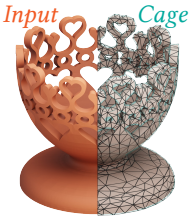
In 3D, our fracture mode computation needs a tetrahedralization of the input's interior, but practical realtime applications prefer to work with triangle surface meshes for input and output. Fortunately, the sparsity inducing discontinuity norm results in fracture modes which are continuous across most pairs of neighboring tetrahedra. It is unnecessary to keep the entire tetrahedral mesh at runtime. Instead, we can determine the connected components determined by neighboring tetrahedra whose shared face's discontinuity term is below σ across all k modes. The boundary of each component is a *solid* [Zhou et al. 2016] triangle mesh of a fracture fragment. Since the impact projection described above is linear, we can pre-restrict the projection to vertices on the boundary of these fragments, discarding all internal vertices and the tetrahedral connectivity.

3.5 Simple Nested Cages

In practice, the input model may be very high-resolution, not yet fully modeled when fractures are precomputed, or too messy to easily tetrahedralize. Like many simulation methods before us, we can avoid these potential performance, workflow and robustness problems by working with a tetrahedralization coarse cage nesting the input model. The NESTED CAGES method of Sacht et al. [2015]

produces tight fitting cages, but suffers long runtimes, potential failure, and may result in a surface mesh which causes subsequent tetrahedralization (e.g., using TetWild [Hu et al. 2018] or TetGen [Si 2015]) to fail.

Therefore, in Algorithm 2 we introduce a very simple caging method inspired by the level-set method of Ben-Chen et al. [2009]. As an example, Nested Cages crashes after a minute on the input mesh in the inset, while our simple algorithm produces a satisfying output cage after 85 seconds.



Algorithm 2: Simple nested cages via binary search

Let $V_{\text{in}}, F_{\text{in}}$ be the vertex and face lists of the input mesh, and m_{target} the desired number of output faces.

Binary Search on offset amount d

- V_{mc},F_{mc} ← marching-cubes(distance to V_{in},F_{in} minus d)
- V_{d},F_{d} ← decimate V_{mc},F_{mc} to m faces
- V_{u},F_{u} ← self-union V_{d},F_{d} via [Zhou et al. 2016]
- V_{t},T_{t}, F_{t} ← tetrahedralize V_{u},F_{u} via [Si 2015]
- **if** *any step failed* **or** V_t,F_t *does not strictly contain* V_{mc},F_{mc} **then**
  - increase d
- **else**
  - decrease d

return V_t,T_t

Like Nested Cages, the output cage will strictly contain the input, but also by construction we ensure that this cage can be successfully tetrahedralized (not just in theory). In a sense, this method provides a different point on the Pareto frontier of tightness-vs-utility. Each step is a fairly standard geometry processing subroutine with predictable performance, and one may even consider using it as an initialization strategy for Nested Cages to improve tightness in the future. We run a max of 10 search iterations, lasting between 5 and 20 seconds each in our examples.

Fracture modes and solid fragment components on the cage's tetrahedralization can be transferred to the true input geometry by intersecting each connected component against the input mesh. In this way, the exterior surface of each fragment component is exactly a subset of the input mesh.

3.6 Smoothing internal surfaces

By our construction, the fracture boundaries will follow faces of the tetrahedral mesh used for their computation. This reveals aliasing with frequency proportional to the mesh resolution. We may optionally alleviate this by treating each extracted per-tet component membership as a one-hot vector field, which we immediately average onto (unexploded) mesh vertices stored as a matrix $\mathbf{Z} \in [0,1]^{n \times |\text{components}|}$, so that $\mathbf{Z}_{i,j} \in [0,1]$ is viewed as the likelihood that vertex i belongs to component j. We apply implicit Laplacian smoothing with a time step of λ to columns of $\mathbf{Z}$:

$$\mathbf{Z} \leftarrow (\mathbf{M} + \lambda \mathbf{L})^{-1}(\mathbf{M}\mathbf{Z}), \tag{20}$$

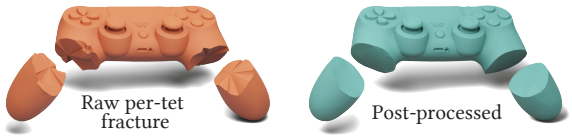


Fig. 10. We use the upper envelope extraction algorithm by Abdrashitov et al. [2021] to obtain smoothed fracture faults.

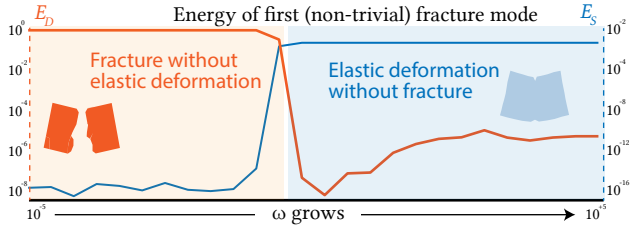


Fig. 11. As our energy weight changes, our output modes sharply transition from only deforming to only fracturing. This gives us the additional insight that fractures occur only within the nullspace of the strain energy, and has the additional effect of making ω a simple parameter to set.

where $\mathbf{M}$, $\mathbf{L}$, are the mass and Laplacian matrices, respectively. The resulting $\mathbf{Z}$ continue to contain fractional values in $[0,1]$ corresponding to a smoothed likelihood. We now *re*-extract piecewise-linear (triangle mesh) component boundaries by computing the upper-envelope (tracking the argmax) using the implementation of Abdrashitov et al. [2021]. While essentially still using the same tetrahedral mesh, utilizing smoothing and piecewise-linear interpolation greatly reduces aliasing artifacts (see Fig. 10).

3.7 Choice of strain energy

So far the only requirement on the strain energy density Ψ is that we can construct its second-order approximation near the rest configuration represented by the (positive semi-definite) Hessian matrix $\mathbf{Q}$. We now investigate the effect of choices of $\mathbf{Q}$ and in particular the relationship with the balancing weight ω.

To make our investigation concrete, take Ψ to be the linear elastic strain density, so that $\mathbf{Q}$ is the common linear elasticity stiffness matrix. By sweeping across values of ω we see a sharp change in the first (and all) fracture mode's behavior with the discontinuity energy dominating over the strain energy and then sharply swapping (see Fig. 11). When the discontinuity energy is effectively zero, then we have simply recovered the usual linear elastic vibration modes (albeit in a convoluted way). When the strain energy is effectively zero, then we not only start to see sparse fractures, but we also see that each fracture fragment undergoes its own zero-strain energy transformation. That is, each fragment undergoes a *linearized* rigid transformation, the only motions in the null space of the strain energy. This is quite interesting as it implies that the precise choice $\mathbf{Q}$ is irrelevant and only its null space matters.

With this in mind, we consider whether all linearized rigid transformations should be admissible. Since we ultimately care about the fracture pattern created by the modes, we observed *qualitatively* that the scaling induced by linearized rotations resulted in small elements breaking off and expanding between fragment boundaries to reduce the discontinuity energy. Rather than attempt to identify these as outliers, we found a simpler solution is to work with

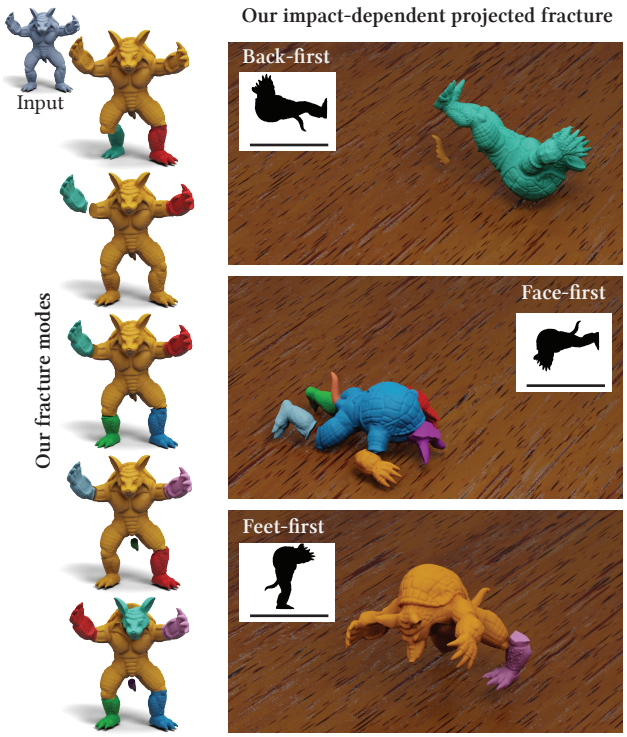


Fig. 12. Our precomputed fracture modes identify the geometrically weakest regions of a shape, and are activated or not on runtime depending on the nature of the impact.

a strain energy that only admits translational motions in its null space, namely, $\Psi(u,x) = \|\nabla u(x)\|^2$. The Hessian of this energy is simply the cotangent Laplacian matrix $\tilde{\mathbf{L}} \in^{(d+1)m\times(d+1)m}$ repeated for each spatial coordinate:

$$\mathbf{Q} = \mathbf{I}_d \otimes \mathbf{L}. \tag{21}$$

This choice of $\mathbf{Q}$ is used in all our examples.

4 TIMING & IMPLEMENTATION DETAILS

We have implemented our main prototype in MATLAB, using GPTOOLBOX [Jacobson et al. 2016]. We used MOSEK [ApS 2019] to solve the conic problem in Eq. (15). We report timings conducted on a 2020 13-inch MacBook Pro with 16 GB memory and 2.3 GHz Quad-Core Intel Core i7 processor. To produce our animations, we follow a traditional HOUDINI [SideFX 2020] fracture simulation workflow, exchanging the usual Voronoi or openVDB fracture nodes for our own fractured meshes. Our impact projection step could be fully integrated into Houdini's rigid body simulator at a minimal performance cost. Only for simplicity in prototyping, we chose not to do this and instead compute our final mesh $\Omega^\star$ in MATLAB taking into account the animation's impact and load it into HOUDINI directly to show a prototype of what our algorithm would look like integrated in a rigid body simulation.

Our algorithm's only parameters are the tolerances ε and σ, which we fix at $\varepsilon = 10^{-10}$ and $\sigma = 10^{-3}$. While σ could be used as a global artistic parameter to decide how easy a given object can fracture,

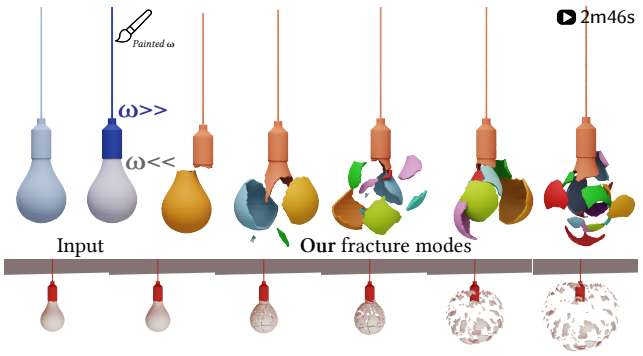


Fig. 13. Our ω parameter can be painted into our input to signal areas that shouldn't fracture; for example, in objects with different materials.

it would have the same effect as a scaling of the impact vector $\mathbf{w}$, and we prefer the use of the latter for having a clearer physical meaning. As we discuss in Section 3.7, our parameter ω does not actually have an effect in the output as long as it is small enough for us to be within the zero-deformation fracture realm (see Fig. 11). We fix $\omega = 0.01$ in all our results.

Our proposed algorithm works in two steps. First, we precompute a given shape's fracture modes. This step takes place offline, following Algorithm 1. The computational bottleneck of this section of our algorithm is the conic solve detailed in Eq. (15). Each mode takes between 20 seconds and two minutes to compute in our meshes, which have between 12,000 and 40,000 vertices.

Secondly, our impact projection step as detailed in Section 3.3 is the only part of our algorithm that happens at runtime. The complexity of this step is dominated by the projection step in Eq. (19), which is $O(k\tilde{n})$, where k is the number of precomputed modes and $\tilde{n}$ is the number of vertices in the boundary of the connected components described in Section 3.4. All other elements of our projection step are $O(p)$, where p is the number of connected components (in our example between 10 and 100) and $p << n$ so they can be disregarded from the complexity discussion. In our examples, $\tilde{n}$ is between 1,000 and 10,000 and we compute between $k = 20$ and $k = 40$ fracture modes, meaning our full runtime step requires between 0.1 and 1 million floating point operations, putting it well within realtime requirements, even if one greatly increases $\tilde{n}$, k or the number of objects on scene (note this projection step only needs to be ran when a collision is detected, and not at every simulation frame). Our unoptimized, CPU implementation takes between 0.01 and 0.1 seconds to carry out this step on our laptop.

5 EXPERIMENTS & COMPARISONS

Our proposed fracture modes naturally identify the regions of a shape that are geometrically weak, as opposed to existing procedural prefracture algorithms. We make this explicitly clear in Figs. 3 and 16, where existing prefracture work fails to identify even the most obvious intuitive breaking patterns which are present in our first (non-trivial) fracture mode. Even in less didactic examples, Voronoi-based prefracture methods result in convex, unrealistic and easily recognizable pieces (see Figs. 18 and 2), while our fracture modes are realistic and can produce a much wider set of shapes.

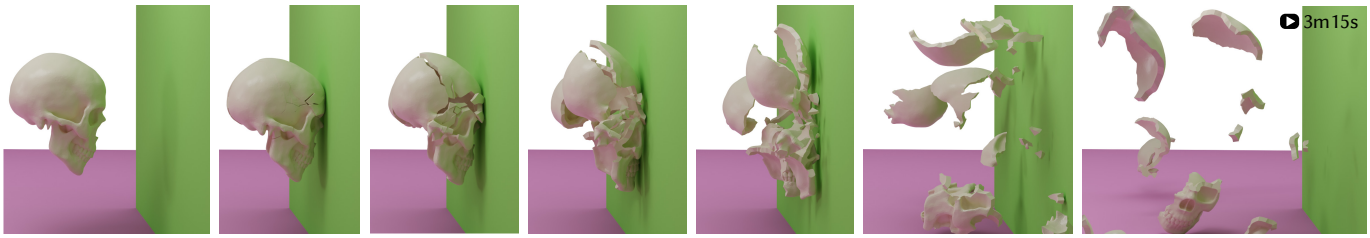


Fig. 14. Ouch, my head hurts!

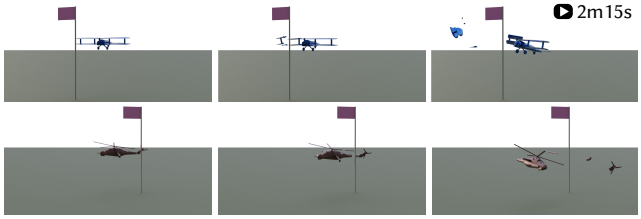


Fig. 15. Similar impacts result in different fracture patterns once we have computed our fracture modes for different objects.

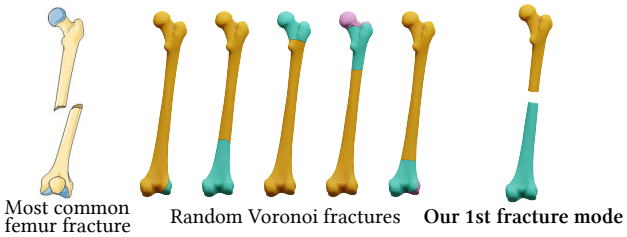


Fig. 16. A healthy femur will usually break at the mid-shaft, as our first fracture mode correctly identifies, unlike existing Voronoi-based algorithms.

"Realism" in a fracture simulation is a hard quantity to evaluate; however, there exist works on structural analysis like [Zhou et al. 2013] that idenfity the weakest regions of a given object. In Fig. 6, we show how our fracture modes produce breaking patterns that align both with their analysis as well as with their real-life experiments.

5.1 Fracture simulations

Our proposed method is ideal for its use in interactive applications. In Fig. 4, we show screenshots of our 2D realtime fracture interactive app. The user can cause different impacts on the object and see the fracture patterns that result from them by projecting onto our fracture modes.

The interactive Computer Graphics application *par excellence* is video games. In Fig. 7, we show a prototype where our precomputed fracture modes for a Space Wizard Vehicle can be stored so that the player sees different fracture behaviours depending on the received impact. In Fig. 15, we precompute the fracture modes for two different vehicles and show how they break under a similar impact.

Our algorithm can be used for any realtime fracture application, from simple objects breaking into solid pieces in the foreground of an animation (see Fig. 12) to thin shells shattering upon impact (see Figs. 17 and 13). In Fig. 14, we use our fracture modes to give a metaphorical representation of the headache produced by previous works' lack of realism.

6 LIMITATIONS & FUTURE WORK

Our fracture modes method is intended for brittle fracture. We conjecture that ductile fracture simulation could also benefit from our sparse-norm formulation. In future work, we would like to improve the performance of our precomputation optimization. We experimented with Manopt [Boumal et al. 2014], but so far observed significantly slower performance than our proposed method. For very large meshes, the projection step could exceed CPU usage allowances for realtime applications. It may be possible to conduct this entirely on the GPU. Our fracture modes are global in nature, meaning they create relations between regions of the object that will not typically fracture together. To prevent a fracture in one location from also causing an undesired fracture elsewhere, it may be possible to add a localizing term to our energy like that of Brandt and Hildebrandt [2017]. Our fracture modes exhibit aliasing due to the computation mesh which we successfully remove with *post facto* smoothing. Many materials like wood, brick, and styrofoam expose interesting high frequency geometric details during fracture. A possible future improvement would be to add fracture surface detail following, e.g., Chen et al. [2014].

We hope our introduction of fracture eigenmodes inspires the realtime simulation community further to use the well-studied tools of modal analysis to this rich problem, and the broader Computer Graphics research community to look at other open problems with this modal lens.

ACKNOWLEDGMENTS

This project is funded in part by NSERC Discovery (RGPIN2017–05235, RGPAS–2017–507938), New Frontiers of Research Fund (NFRFE–201), the Ontario Early Research Award program, the Canada Research Chairs Program, the Fields Centre for Quantitative Analysis and Modelling and gifts by Adobe Systems. The first author is supported by an NSERC Vanier Canada Scholarship, the FA&S Dean's Excellence Scholarship, the Beatrice "Trixie" Worsley Graduate Scholarship and a Connaught International Scholarship. The second, third, fourth and fifth authors were supported by the 2020 Fields Undergraduate Summer Research Program.

We acknowledge the authors of the 3D models used throughout this paper and thank them for making them available for academic use: MakerBot (Fig. 1, CC BY 4.0), HQ3DMOD (Figs. 5 and 17, TurboSquid 3D Standard Model License), Freme Minskib (Fig. 7, CC BY-NC 4.0), 3Demon (Fig. 9, CC BY-NC-SA 4.0), Reality_3D (Fig. 10, CC BY 4.0), Alex (Fig. 15, CC BY-NC-SA 4.0), Falha Tecnologica (Fig. 13, TurboSquid 3D Standard Model License), LeFabShop (Fig. 14, CC

Fig. 17. A glass cup shatters, resulting in many non-convex pieces that would be impossible to obtain with Voronoi-based prefracture methods.

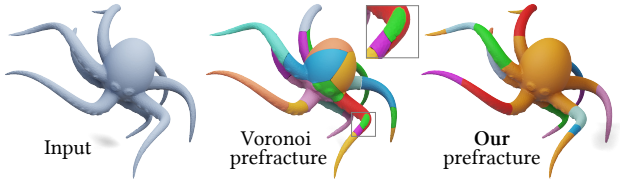


Fig. 18. The pieces generated by Voronoi-based methods can be extremely unrealistic (center). By combining all the possible fractures in all our modes into a single *prefractured* mesh, we can provide a zero-extra-realtime-cost alternative to procedural algorithms.

BY-NC 4.0), The Database Center for Life Science (Fig. 16, CC BY-SA 2.1), Gijs (inset in Section 3.5, CC BY-NC 4.0).

We would like to thank Chris Wojtan, David Hahn and Klint Qinami for early experiments and discussions of sparse-norm fracture models; Eitan Grinspun, David I.W. Levin and Oded Stein for insightful conversations; Rinat Abdrashitov for providing an implementation of his algorithm mentioned in Section 3.6; Qingnan Zhou for providing the 3D models used in Fig. 6; Xuan Dam, John Hancock and all the University of Toronto Department of Computer Science research, administrative and maintenance staff that literally kept our lab running during a very hard year.

REFERENCES

Rinat Abdrashitov, Seungbae Bang, David IW Levin, Karan Singh, and Alec Jacobson. 2021. Interactive Modelling of Volumetric Musculoskeletal Anatomy. *arXiv preprint arXiv:2106.05161* (2021).

MOSEK ApS. 2019. *The MOSEK optimization toolbox for MATLAB manual. Version 9.0.* http://docs.mosek.com/9.0/toolbox/index.html

Mirela Ben-Chen, Ofir Weber, and Craig Gotsman. 2009. Spatial deformation transfer. In *Proc. SCA*, Dieter W. Fellner and Stephen N. Spencer (Eds.).

N. Boumal, B. Mishra, P.-A. Absil, and R. Sepulchre. 2014. Manopt, a Matlab Toolbox for Optimization on Manifolds. *Journal of Machine Learning Research* 15, 42 (2014), 1455–1459. https://www.manopt.org

Stephen P. Boyd. 2010. *Distributed Optimization and Statistical Learning Via the Alternating Direction Method of Multipliers.*

Christopher Brandt and Klaus Hildebrandt. 2017. Compressed vibration modes of elastic bodies. *Computer Aided Geometric Design* 52 (2017), 297–312. http://graphics.tudelft.nl/Publications-new/2017/BH17

Emmanuel J. Candes and Michael B. Wakin. 2008. *An Introduction To Compressive Sampling.*

Zhili Chen, Miaojun Yao, Renguo Feng, and Huamin Wang. 2014. Physics-inspired adaptive fracture refinement. *ACM Transactions on Graphics* 33, 4 (2014).

Loeiz Glondu, Maud Marchal, and Georges Dumont. 2012. Real-time simulation of brittle fracture using modal analysis. *IEEE Transactions on Visualization and Computer Graphics* 19, 2 (2012), 201–209.

David Hahn and Chris Wojtan. 2015. High-resolution brittle fracture simulation with boundary elements. *ACM Trans. Graph.* 34, 4 (2015), 151:1–151:12. https://doi.org/10.1145/2766896

David Hahn and Chris Wojtan. 2016. Fast approximations for boundary element based brittle fracture simulation. *ACM Trans. Graph.* 35, 4 (2016), 104:1–104:11. https://doi.org/10.1145/2897824.2925902

Koichi Hirota, Yasuyuki Tanoue, and Toyohisa Kaneko. 1998. Generation of crack patterns with a physical model. *The visual computer* 3, 14 (1998), 126–137.

Yixin Hu, Qingnan Zhou, Xifeng Gao, Alec Jacobson, Denis Zorin, and Daniele Panozzo. 2018. Tetrahedral meshing in the wild. *ACM Trans. Graph.* (2018).

George R Irwin. 1957. Analysis of stresses and strains near the end of a crack traversing a plate. (1957).

Alec Jacobson et al. 2016. gptoolbox: Geometry Processing Toolbox. http://github.com/alecjacobson/gptoolbox.

Peter Kaufmann, Sebastian Martin, Mario Botsch, Eitan Grinspun, and Markus Gross. 2009. Enrichment Textures for Detailed Cutting of Shells. *ACM Trans. Graph.* (2009).

Dan Koschier, Sebastian Lipponer, and Jan Bender. 2015. Adaptive tetrahedral meshes for brittle fracture simulation. In *SCA '14.*

R. B. Lehoucq, D. C. Sorensen, and C. Yang. 1998. *ARPACK Users' Guide.* Society for Industrial and Applied Mathematics. https://doi.org/10.1137/1.9780898719628

Lien Muguercia, Carles Bosch, and Gustavo Patow. 2014. Fracture modeling in computer graphics. *Computers & graphics* 45 (2014), 86–100.

Matthias Müller, Nuttapong Chentanez, and Tae-Yong Kim. 2013. Real time dynamic fracture with volumetric approximate convex decompositions. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 1–10.

Ken Museth, Peter Cucka, Mihai Alden, and David Hill. 2021. *OpenVDB.* https://www.openvdb.org

T. Neumann, K. Varanasi, C. Theobalt, M. Magnor, and M. Wacker. 2014. Compressed Manifold Modes for Mesh Processing. *Computer Graphics Forum* 33, 5 (2014), 35–44.

Thomas Neumann, Kiran Varanasi, Stephan Wenger, Markus Wacker, Marcus A. Magnor, and Christian Theobalt. 2013. Sparse localized deformation components. *ACM Trans. Graph.* (2013).

Alan Norton, Greg Turk, Bob Bacon, John Gerth, and Paula Sweeney. 1991. Animation of fracture by physical modeling. *The visual computer* 7, 4 (1991), 210–219.

James F O'Brien and Jessica K Hodgins. 1999. Graphical modeling and animation of brittle fracture. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques.* 137–146.

Seungtaik Oh, Seunghyup Shin, and Hyeryeong Jun. 2012. Practical simulation of hierarchical brittle fracture. *Computer Animation and Virtual Worlds* 23, 3-4 (2012), 291–300.

V. Ozolins, R. Lai, R. Caflisch, and S. Osher. 2013. Compressed modes for variational problems in mathematics and physics. *Proceedings of the National Academy of Sciences* 110, 46 (Oct 2013), 18368–18373. https://doi.org/10.1073/pnas.1318679110

Eric G Parker and James F O'Brien. 2009. Real-time deformation and fracture in a game environment. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation.* 165–175.

Tobias Pfaff, Rahul Narain, Juan Miguel de Joya, and James F. O'Brien. 2014. Adaptive Tearing and Cracking of Thin Sheets. *ACM Trans. Graph.* (2014).

Saty Raghavachary. 2002. Fracture generation on polygonal meshes using Voronoi polygons. In *ACM SIGGRAPH 2002 conference abstracts and applications.* 187–187.

Leonardo Sacht, Etienne Vouga, and Alec Jacobson. 2015. Nested cages. *ACM Trans. Graph.* (2015).

Sara C. Schvartzman and Miguel A. Otaduy. 2014. Fracture Animation Based on High-Dimensional Voronoi Diagrams. In *Proc. I3D.*

Hang Si. 2015. TetGen, a Delaunay-Based Quality Tetrahedral Mesh Generator. *ACM Trans. Math. Softw.* (2015).

SideFX. 2020. *Houdini.* https://www.sidefx.com

Martin Wicke, Daniel Ritchie, Bryan M Klingner, Sebastian Burke, Jonathan R Shewchuk, and James F O'Brien. 2010. Dynamic local remeshing for elastoplastic simulation. *ACM Transactions on graphics (TOG)* 29, 4 (2010), 1–11.

Joshuah Wolper, Yunuo Chen, Minchen Li, Yu Fang, Ziyin Qu, Jiecong Lu, Meggie Cheng, and Chenfanfu Jiang. 2020. AnisoMPM: Animating anisotropic damage mechanics: Supplemental document. *ACM Trans. Graph* 39, 4 (2020).

Joshuah Wolper, Yu Fang, Minchen Li, Jiecong Lu, Ming Gao, and Chenfanfu Jiang. 2019. CD-MPM: continuum damage material point methods for dynamic fracture animation. *ACM Transactions on Graphics (TOG)* 38, 4 (2019), 1–15.

Qingnan Zhou, Eitan Grinspun, Denis Zorin, and Alec Jacobson. 2016. Mesh arrangements for solid geometry. *ACM Trans. Graph.* (2016).

Qingnan Zhou, Julian Panetta, and Denis Zorin. 2013. Worst-case structural analysis. *ACM Trans. Graph.* 32, 4 (2013), 137–1.

Yufeng Zhu, Robert Bridson, and Chen Greif. 2015. Simulating Rigid Body Fracture with Surface Meshes. *ACM Trans. Graph.* (2015).

A CANONICAL CONIC PROGRAM FORM OF EQ. (15)

Let us define a sparse matrix $\mathbf{D} \in \mathbb{R}^{2p\times(d+1)m}$ that operates on a deformation map, so that $\mathbf{Du}$ contains the vertex-wise discontinuity (weighted as per the Gaussian quadrature rule). More precisely, each row of $\mathbf{D}$ corresponds to an endpoint of an interior edge and will have nonzero entries corresponding to the incident vertices.

$\mathbf{D}$ can be constructed in any arbitrary, but consistent, order. For convenience, we construct $\mathbf{D}$ such that its ith row for $i \in \{1,\dots,p\}$ acts on one endpoint and its $(i+p)$th row acts on the other endpoint of the same internal edge.

Since $\mathbf{Q}$ is positive semi-definite, we can write it as $\mathbf{Q} = \mathbf{R}^\top\mathbf{R}$ for some matrix $\mathbf{R}$. Define

$$\mathbf{Y}_e = \sqrt{l_e/2}(\mathbf{Du})_e\,, \forall e = 1,\dots,p \tag{22}$$

where l_e is the length of edge e. Next, define

$$\mathbf{r}_i = (\mathbf{Ru})_i\,, \forall i = 1,\dots,6m. \tag{23}$$

Then, Eq. (15) can be written in the canonical form

$$\underset{\mathbf{u},t,\mathbf{Y},\mathbf{z}}{\operatorname{argmin}} \begin{bmatrix}1\\ \mathbf{0}\\ \mathbf{0}\\ \mathbf{0}\\ \mathbf{1}\end{bmatrix}^\top \begin{bmatrix}t\\ \mathbf{r}\\ \mathbf{u}\\ \mathbf{Y}\\ \mathbf{z}\end{bmatrix} \tag{24}$$

subject to

$$\begin{aligned} t &\geq \sqrt{\mathbf{r}_1^2 + \ldots + \mathbf{r}_{6m}^2} \\ \mathbf{z}_e &\geq \sqrt{\|\mathbf{Y}_e\|_2^2 + \|\mathbf{Y}_{e+p}\|_2^2} \quad \forall e \\ \mathbf{Y} &= \mathbf{Du} \\ \begin{bmatrix}\mathbf{U}_1^\top\\ \vdots\\ \mathbf{U}_{i-1}^\top\\ \mathbf{c}^\top\end{bmatrix} \mathbf{Mu} &= \begin{bmatrix}0\\ \vdots\\ 0\\ 1\end{bmatrix} \end{aligned} \tag{25}$$