Fracture Animation Based on High-Dimensional Voronoi Diagrams

Sara C. Schvartzman
Stanford University

Miguel A. Otaduy
URJC Madrid

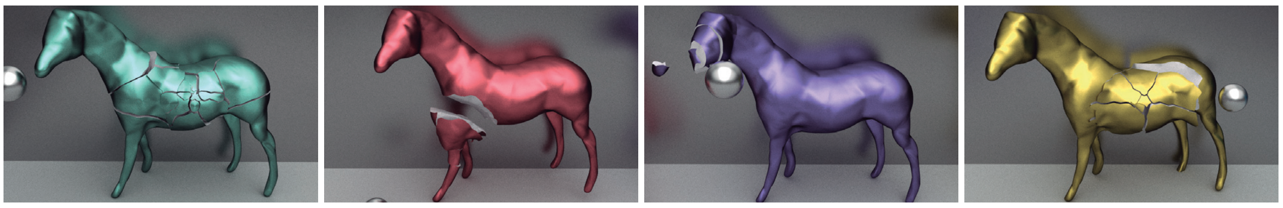

Figure 1: *Horses fractured by metal balls. These fractures emphasize the richness and versatility provided by our method: object concavities are correctly resolved, cracks may be curved, and fracture patterns adapt to the impact and object properties.*

Abstract

We propose a novel algorithm to simulate brittle fracture. It augments previous methods based on Voronoi diagrams, improving their versatility and their ability to adapt fracture patterns automatically to diverse collision scenarios and object properties. We cast brittle fracture as the computation of a high-dimensional *Centroidal Voronoi Diagram* (CVD), where the distribution of fracture fragments is guided by the deformation field of the fractured object. By formulating the problem in high dimensions, we support robustly object and crack concavities, as well as intuitive artist control. We further accelerate the fracture animation process with example-based learning of the fracture degree, and a highly parallel tessellation algorithm. As a result, we obtain fast animations of detailed and rich fractures, with fracture patterns that adapt to each particular collision scenario.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

Keywords: fracture animation, Voronoi diagram

1 Introduction

Fracture animation creates spectacular effects in motion pictures, and it is lately making its way into video games and virtual reality applications [Parker and O'Brien 2009; Muller et al. 2013]. Physically-based simulation of crack generation and propagation produces highly realistic results [O'Brien and Hodgins 1999], but it is also a computationally expensive process. It turns out excessively expensive, indeed, for interactive applications when high resolution fractures are needed. In this work, we seek a faster but plausible solution for brittle, stiff objects.

Voronoi fracture methods offer a fast alternative to physically based simulation methods. They compute the locations of fragment centers, and then the fracture fragments are defined as the Voronoi cells of those centers [Raghavachary 2002; Bao et al. 2007; Muller et al. 2013]. The success of Voronoi methods stands on the similarity in shape and distribution between the fragments in a real brittle fracture and the cells of a Voronoi diagram. Although Voronoi methods are fast, they pose other challenges, mainly the versatile distribution of fragments according to arbitrary external forces, and the correct handling of object and crack concavities. Most previous methods are oblivious of the forces acting on the fractured object. Some calculate the fracture as a preprocess [Raghavachary 2002; Hellrung et al. 2009], others place the fragment centers in a random manner [Bao et al. 2007; Zheng and James 2010], and others take the number of fragments or the fracture pattern as a predefined input [Su et al. 2009; Liu et al. 2011; Muller et al. 2013].

We introduce a new Voronoi fracture method that distributes fracture fragments based on the deformation field of the fractured object, properly handles object and crack concavities, allows for intuitive artist control, and is guided by examples to accelerate computations. Our method is sustained on four major contributions:

- As described in Section 3, we formulate fracture as an optimization problem where the distribution of fragments is a result of the deformation of the fractured object. This optimization problem can be casted as a Centroidal Voronoi Diagram (CVD).
- Unlike standard Voronoi methods based on the 3D Euclidean distance metric, in Section 3 we also show that, by lifting the object to higher dimensions, we can handle object and crack concavities. Intuitive artist control can also be easily accommodated by warping the reference space for Voronoi computations.
- In Section 4 we demonstrate that the runtime fracture computation can be drastically reduced by leveraging precomputed fracture examples. As shown in the pipeline in Fig. 2, we compute fracture examples as a preprocess, running a full optimization of fragment distribution. At runtime, we infer directly the fracture degree from the deformation field and the example database.
- To efficiently tessellate the fragments of a CVD with curved faces, we propose a highly parallel tessellation algorithm built on top of a tetrahedral lattice, as described in Section 5.

Our method produces fast animations where objects may be fractured in arbitrary non-scripted ways, showing rich and diverse fracture patterns even at close views, such as those in Fig. 1.

2 Related Work

Physically based simulation of fracture involves solving challenging mechanical problems such as the computation of deformations,

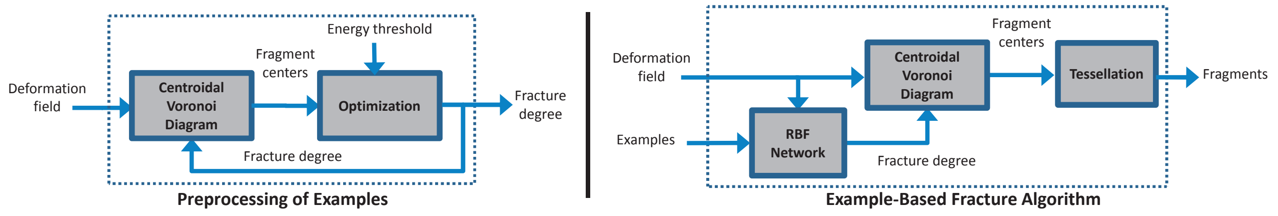

Figure 2: *As a pre-process (left), we compute multiple fracture examples of the same object, allowing us to learn a relation between the deformation field and the fracture degree. At runtime (right), given the deformation field of the fracturing object, we compute the fracture degree from the examples, solve a deformation-aware Centroidal Voronoi Diagram, and tessellate the resulting fragments.*

crack generation, and crack propagation. O'Brien et al. introduced to computer graphics methods to simulate fracture based on the propagation of crack surfaces as the result of local deformations, first for brittle fracture [O'Brien and Hodgins 1999], and then for ductile fracture [O'Brien et al. 2002]. Brittle materials recover their undeformed state upon fracture, and cracks propagate rapidly. Ductile materials, on the other hand, consume energy in the form of plastic deformation upon fracture, and often exhibit progressive crack propagation. In this work, we focus on stiff brittle materials, and we assume that the deformation of objects is visually imperceptible and fracture propagates instantaneously.

Physically based computation of fracture is attractive to artists because the shape and distribution of fracture fragments naturally respond to the collisions and forces in the simulation. However, it requires computationally demanding operations such as remeshing and fine time stepping. Bielser et al. [2003], showed how to handle all possible remeshing operations rigorously thanks to a finite state machine. The virtual node algorithm alleviates remeshing by limiting the resolution of fractures of a finite element mesh [Molino et al. 2004]. Sifakis et al. [2007] proposed defining fractures implicitly and retriangulating only for rendering purposes, to optimize the robustness of remeshing.

The first step in the animation of fracture is the computation of elastic deformations. We adopt a common approach for stiff objects, simulating them as rigid bodies until an impact is detected, and then computing a quasi-static solution to the elastic deformation [Muller et al. 2001; Bao et al. 2007; Liu et al. 2011]. Recently, Glondu et al. [2012a] have presented an approach in which they compute a reduced deformation based on modal analysis. This approach turns out beneficial for fast crack propagation, because they can evaluate local deformations and energies in a highly parallel manner.

In contrast to physically based methods that compute fracture through local crack propagation, Voronoi methods compute fracture through the global distribution of fragments. Our method falls into this category, but, unlike previous methods, it optimizes the distribution of fragments according to impact-induced deformations. Hence, each fracture produces a distinct rich pattern. Previous methods account for the object's deformation in a limited manner. Some use preprocessed Voronoi diagrams or fracture patterns [Raghavachary 2002; Hellrung et al. 2009; Su et al. 2009; Muller et al. 2013], others distribute fragments in a random manner [Bao et al. 2007; Zheng and James 2010], and the method of Liu et al. [2011] takes the number of fragments as a user input. The recent approach of Müller et al. [2013] intersects predefined fracture patterns against a convex decomposition of the fractured object. This method enables very fast runtime fracture, but the resulting fragments are cut with planar boundaries and the fracture patterns do not adapt to the collision scenarios or object properties. Our approach, instead, solves an optimization problem in order to distribute Voronoi sites according to the deformation of the fractured object, and hence we sacrifice some speed for richer and more versatile fracture. Our site placement approach is formulated as a CVD. Du et al. [1999] discuss algorithms to compute the CVD and multiple applications of this geometric structure.

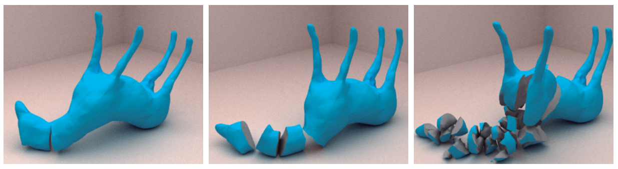

Figure 3: *Three horses falling on their heads with different fracture energy thresholds γ (highest γ on the left to lowest on the right).*

3 Deformation-Based Voronoi Fracture

In this section, we present a formulation of fracture as a CVD. Our approach is aware of the collision scenarios suffered by the fractured object, and determines the size and distribution of fragments based on the distribution of deformation energy, as shown in Fig. 3. In addition, we adopt an interior distance metric to handle robustly object and crack concavities. We demonstrate that a CVD under such interior distance metric can be computed efficiently using the well-known Lloyd's method by lifting the object representation to higher dimensions. We conclude the section describing ways to introduce intuitive artist control into our fracture method.

3.1 Fracture as a Centroidal Voronoi Diagram

In brittle materials fracture propagates quickly, and the resulting fragments release their deformation energy and recover their initial shape. Based on this observation, we propose the following fracture criterion: an object will fracture if its deformation energy is larger than a certain threshold γ.

We define the *distance-weighted deformation energy* of an object with volume Ω_i, center p_i, and strain energy density $W(x)$, $x \in \Omega_i$, as

$$E_{D,i} = \int_{\Omega_i} \text{dist}(x, p_i)^2 \, W(x) \, dx. \tag{1}$$

In essence, $E_{D,i}$ adds up strain energy, but penalizes the distance to the center of the object based on some suitable distance metric $\text{dist}(x, p_i)$. In this way, an object with a large deformation concentrated in one place is more prone to fracture than an object with a moderate homogeneous deformation.

If an object is fractured, we place the centers of the new fragments such that the post-fracture deformation energy is minimized, i.e., the energy consumed by fracture is maximized. This procedure can

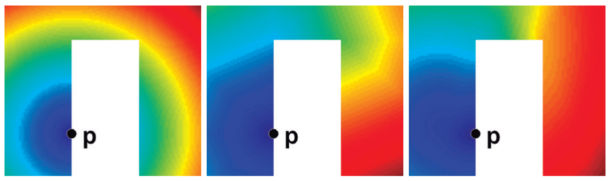

Figure 4: *Distance to a point p in a non-convex object, computed using (left) the Euclidean metric, (center) grid-based distance, and (right) the interior distance by Rustamov et al. [2009].*

be applied recursively while the maximum sustainable deformation energy γ is exceeded. With $P^* = \{p_i^*\}$ the locations of fragment centers and $N = |P^*|$ the fracture degree, we formulate fracture as the computation of the minimum number of fragments such that the distance-weighted deformation energy does not exceed the fracture threshold:

$$N = \min |P^*| \quad \text{such that} \quad E_D(P^*) < \gamma. \tag{2}$$

$$P^* = \arg\min_P E_D(P), \tag{3}$$

$$\text{with } E_D = \sum_i \int_{\Omega_i} \text{dist}(x, p_i)^2 \, W(x)\, dx.$$

It turns out that the solution to the optimization problem in (3) is well known, and it corresponds to the CVD of object Ω [Du et al. 1999]. Then, the solution to the fracture problem in (2)-(3) is given by the CVD with the least amount of sites which satisfies $E_D < \gamma$.

3.2 CVD with Interior Distance Metric

For non-convex objects, the computation of the CVD using the Euclidean distance metric may produce small fragments topologically far away from the impact location. Instead, we correctly handle object concavities computing the CVD using an interior distance metric. We adopt the interior distance definition by Rustamov et al. [2009], which is based on a high-dimensional embedding that approximately preserves surface distances. Specifically, given two surface vertices v_i and v_j with interior distance $\text{dist}(v_i, v_j) = d_{ij}$, the Euclidean distance between their corresponding high-dimensional coordinates $\bar{v}_i$ and $\bar{v}_j$ is also $\|\bar{v}_i - \bar{v}_j\| = d_{ij}$. The high-dimensional coordinates of surface vertices are computed using a diffusion map [Coifman et al. 2005], which can be extended to meshes using the algorithm by de Goes et al. [2008], and the high-dimensional coordinates of interior points are computed through barycentric interpolation using mean-value coordinates [Floater 2003]. Given a matrix $\bar{V}$ whose columns represent the high-dimensional coordinates of surface vertices, and an interior point p with mean-value coordinates $b(p)$, its high-dimensional coordinates can be computed as $\bar{p} = \bar{V}\, b(p)$.

Following Rustamov et al., the interior distance between two arbitrary interior points p and q can be approximated as

$$\text{dist}(p, q)^2 = \|\bar{p} - \bar{q}\|^2 = (b(p) - b(q))^T \bar{V}^T \bar{V} (b(p) - b(q)). \tag{4}$$

We have considered other approaches for interior distance computation, such as a grid-based distance. However, this metric suffers from grid artifacts that can strongly hurt the smoothness of the boundaries of the CVD. Fig. 4 compares Euclidean distance, grid-based distance, and the distance of Rustamov et al.

Once the interior distance metric is defined, we present an efficient algorithm to compute the CVD using such metric. Lloyd's method [Lloyd 1982] is a popular approach to compute the energy-weighted CVD in (3) for the Euclidean distance metric. It iterates two steps until convergence: (i) computation of the Voronoi diagram for a given set of sites, and (ii) moving the sites to the centroids of their Voronoi cells. The use of an interior distance metric makes the CVD problem highly nonlinear. For instance, the centroid of a Voronoi cell is defined more generally as the Fréchet mean [Fréchet 1948] of the points in the cell, and its computation requires the solution to a complex nonlinear optimization problem. However, we demonstrate that for the interior distance of Rustamov et al., this nonlinear optimization problem admits a practical and efficient solution through the computation of a Euclidean CVD in the high-dimensional embedding space.

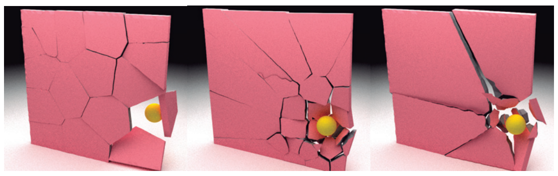

Figure 5: *Artist control of fracture granularity through simple modification of the exponent of strain energy, W^α. From left to right: $\alpha = 0$ (the deformation is ignored), $\alpha = 0.5$, $\alpha = 1$.*

Theorem 1 *Given an object Ω and its high-dimensional image $\bar{\Omega}$, the weighted centroid $\bar{p}^*$ of $\bar{\Omega}$ is the high-dimensional image of a Fréchet mean p^* of Ω.*

Formally, given $f(p) = \int_\Omega \text{dist}(x, p)^2 \, W(x)\, dx$ and $\bar{f}(\bar{p}) = \int_\Omega \|\bar{x} - \bar{p}\|^2 \, W(x)\, dx$, then:

$$\bar{p}^* = \arg\min_{\bar{p}} \bar{f}(\bar{p}) \;\Rightarrow\; p^* = \arg\min_p f(p). \tag{5}$$

The proof is given in the Appendix.

Corollary 1 *The interior CVD of an object Ω can be obtained by first computing the Euclidean CVD in the high-dimensional embedding space, and then transforming the resulting Voronoi sites to the regular low dimensional space.*

Formally:

$$\{\bar{p}_i^*\} = \arg\min_{\{\bar{p}_i\}} \sum_i \int_{\Omega_i} \|\bar{x} - \bar{p}_i\|^2 \, W(x)\, dx \Rightarrow \tag{6}$$

$$\{p_i^*\} = \arg\min_{\{p_i\}} \sum_i \int_{\Omega_i} \text{dist}(x, p_i)^2 \, W(x)\, dx.$$

Note that the high-dimensional Euclidean CVD can be computed using Lloyd's method.

3.3 Preprocessing and Runtime Algorithms

Our full fracture algorithm proceeds as follows. First, given external forces on an object Ω, we compute the strain energy density $W(x)$ using a quasi-static finite element formulation. Then, the strain energy field is used as input for the computation of the fracture fragments. Finally, the surfaces of the resulting fragments are tessellated. We use a tetrahedral mesh to discretize the computations in all three steps.

As shown in Fig. 2, as a preprocess we compute a set of example fractures for each object, and these examples are used at runtime to accelerate the computation of the fracture degree N (See Section 4 for full details). During preprocessing, we compute the fracture degree following a simple optimization process. We initialize the fracture degree $N = 1$, and we double it until the distance-weighted

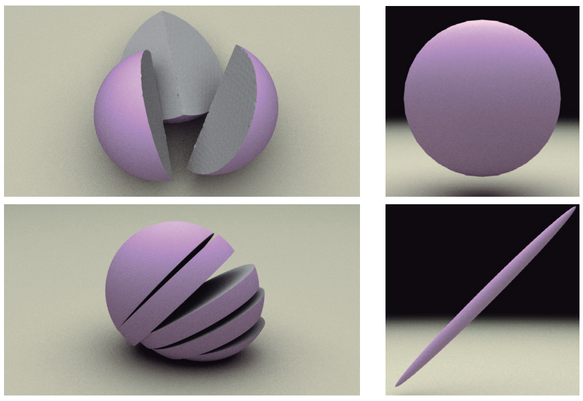
Figure 6: *Artist control based on the modification of the rest shape. Top: Regular fracture of a sphere when its rest-shape is not modified. Bottom: Fracture with a preferred direction obtained by applying anisotropic scaling to the rest shape (on the right).*

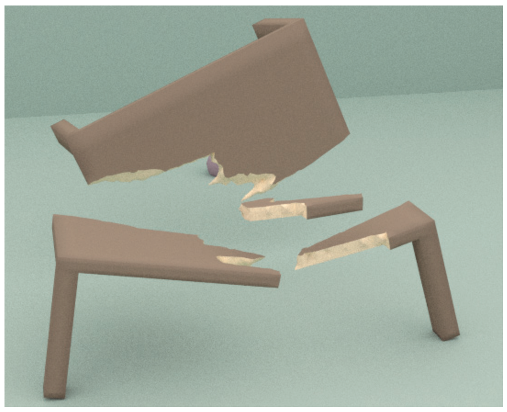
Figure 7: *Fracture of a wooden table enabled by artist control. A wavy transformation is applied to the rest-shape of the table, and thus it produces concave wood-like fragments upon fracture.*

deformation energy E_D is smaller than the fracture threshold γ. Then, we perform a bisection search on the fracture degree until we reach the smallest value for which the distance-weighted deformation energy is smaller than the fracture threshold. During this optimization, whenever we introduce new Voronoi sites we seed them randomly using as probability function the strain energy field.

To compute the location of fragment centers, both for preprocessing examples or during runtime fracture, we solve a discrete version of the high-dimensional CVD on the nodes of the tetrahedral mesh using Lloyd's method. In the context of this discretization, the strain energy $W(x)$, used as distance weight in (3), needs to be evaluated at mesh nodes. Once the deformation is computed, we integrate strain energy on tetrahedra, and we set node weights $W(x)$ by summing one fourth of the strain energy over their incident tetrahedra. The execution of Lloyd's method in the discrete setting requires small changes too. First, in the computation of high-dimensional centroids, the integral over Voronoi cells is trivially substituted by a summation. Second, the update of Voronoi cells reduces to finding the closest site for each node. We speed up this computation by flooding Voronoi cells from the sites, exploiting the graph defined by the tetrahedral mesh. In addition, to accelerate the evaluation of interior distances, as a preprocess we compute the high-dimensional coordinates for all nodes. It turns out that our algorithm does not require the 3D positions of the Voronoi sites at any time, and it is sufficient to store their high-dimensional positions and the cell-classification of the nodes.

3.4 Artist Control

A major feature of our proposed fracture algorithm is that it can accommodate many types of artist control in very simple ways. Other than the trivial energy threshold γ that guides the overall toughness of the object, we have considered artist-driven fracture granularity, inhomogeneous material toughness, anisotropy, and smoothness, but other properties may also be controllable.

Fracture granularity and material inhomogeneity may be easily controlled by tweaking the strain energy W. Using an exponential factor of the strain energy, W^α, affects the influence of the deformation on the fragment distribution. Using a spatially varying multiplicative factor, βW, allows for controllable material inhomogeneity. The artist may 'paint' fragile regions with $\beta > 1$, and tough regions with $\beta < 1$.

Fracture anisotropy and smoothness may be easily controlled by tweaking the distance metric. A simple way to do this is to apply a non-uniform transformation to the undeformed reference object where distances are computed. Fig. 6 shows an example of anisotropic material failure obtained by applying non-uniform scaling to the reference object. Fig. 7 shows wood-like fracture of a table, obtained by applying a wavy transformation to its rest shape.

4 Learning Fracture from Examples

Our fracture model defines both the fracture degree N and the fragment centers P^* as a function of the strain energy W. As described in Section 3.3, computing the fracture degree requires solving a costly iterative optimization problem. In this section, we introduce a learning method that, based on a set of precomputed fracture examples, allows us to efficiently estimate the fracture degree at runtime as a function of the deformation field. We describe our specific learning method based on a radial basis function (RBF) network, and we discuss its approximation accuracy.

4.1 RBF Network

Let $\mathbf{W}$ be regarded as a vector that concatenates the strain energies of all nodes of the mesh. Our fracture model can be represented as a process that calculates the fracture degree N and the fragment centers P^* as a process

$$(N, P^*) = f(\mathbf{W}). \tag{7}$$

Conceptually, the fracture model can be divided into two functions: (i) determining the fracture degree:

$$N = f_N(\mathbf{W}), \tag{8}$$

and (ii) computing the fragment centers as the CVD:

$$P^* = \text{CVD}(N, \mathbf{W}). \tag{9}$$

These two conceptual functions correspond to the optimization problems in (2) and (3) respectively.

Computing the fracture degree constitutes the bottleneck of our fracture model, because of its iterative nature. We propose an example-based approach to approximate f_N in a fast manner. As a preprocess, we generate a set of example strain energy vectors

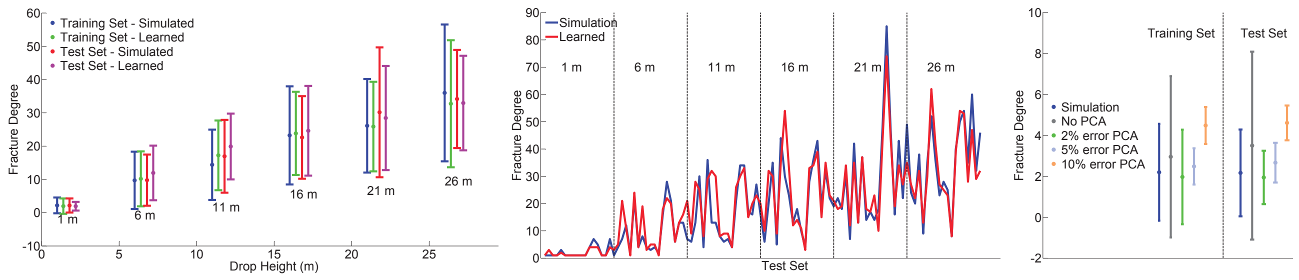

Figure 8: *We study how well our algorithm learns the fracture degree when we drop the horse in Fig. 3 from different heights with different orientations. Left: Mean and standard deviation of the fracture degree across different orientations, using the simulation and learning methods on the training and test sets separately. Middle: Exact comparison of the fracture degree for all the heights and orientations in the test set. Right: Mean and standard deviation of the fracture degree across all orientations and a height of 1 m, for various PCA error settings.*

$\{\mathbf{W}_i\}$, and compute the fracture degree N_i for each example. The example-based approximation of the fracture degree can be formalized as

$$N = \hat{f}_N(\mathbf{W}, \{\mathbf{W}_i, N_i\}) \approx f_N(\mathbf{W}). \tag{10}$$

To robustly learn the main aspects of the mapping between the deformation field and the fracture degree, without incurring into overfitting, we perform a principal component analysis (PCA) of the strain energy data $\{\mathbf{W}_i\}$, and compute a reduced deformation basis. Then, given a strain energy vector $\mathbf{W}$, the operation $\Pi\,\mathbf{W}$ projects it onto the reduced basis. The example-based approximation of the fracture degree can then be rewritten as

$$N = \hat{f}_N(\Pi\,\mathbf{W}, \{\Pi\,\mathbf{W}_i, N_i\}). \tag{11}$$

We have designed an example-based model to compute the fracture degree using an RBF network. Given a strain energy vector $\mathbf{W}$, the fracture degree is computed as

$$N = \sum_{i}^{k} w_i \phi(\|\Pi\,\mathbf{W} - c_i\|) + b. \tag{12}$$

Based on the training examples, we compute the k RBF centers $\{c_i\}$, the RBF weights $\{w_i\}$ and the bias b in Matlab using orthogonal least squares [Chen et al. 1991]. We have used the RBF $\phi(r) = r$, with global support. k is calculated by incrementing the number of RBF centers until the net does not exceed a maximum error using the training examples.

4.2 Training and Test Sets

The set of examples used for training the RBF network could depend on the types of interactions expected at runtime. For example, for the wall benchmark in Fig. 5, we generated training examples by throwing balls with different velocities to different points on the wall. However, as a general procedure for example generation, we propose to drop an object from different heights and with different orientations. We found that this simple procedure is capable of producing diverse deformation distributions. The fractures in Fig. 1 were produced by throwing balls at the horses, using examples produced by dropping horses.

To test the quality of the example-based fracture model presented above, we have compared its results to the iterative algorithm presented in Section 3.3. We have generated several training and test examples for both the horse and bunny objects shown in Fig. 3 and Fig. 9. For training, we drop each object from 6 different heights and with 64 random, uniformly distributed orientations. We repeat each experiment 3 times, to account for the randomness in the initialization of Voronoi sites. For testing, we drop the objects from the same 6 heights, but with different 18 orientations.

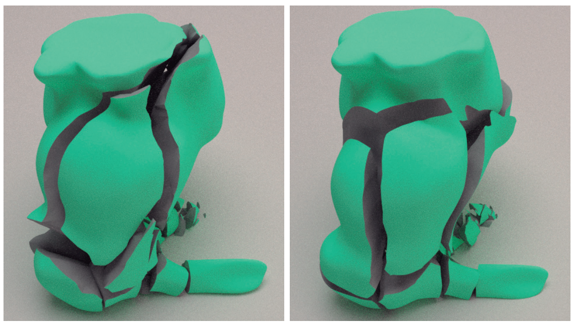

Figure 9: *Simulation of the fracture of a bunny with iterative (left), vs. example-based computation of the fracture degree (right).*

Fig. 8 shows the learning results for the horse example (The bunny example reached very similar results). The first two plots show that the example-based fracture degree is very similar to the one computed through the full iterative method, both for the training and test data sets. On the left, we plot the mean and standard deviation of the fracture degree across all initial orientations of the horse for the same drop height. In the middle, we plot the exact fracture degree for all the orientations and heights in the test set.

The third, right-most plot, compares the results for various error settings in the PCA projection of the strain energy, for a drop height of 1 m. With no PCA projection, the method suffers from overfitting, which results in larger errors in the fracture degree (indicated by a larger standard deviation). With a PCA error of 10%, the quality of the fitting degrades too. Therefore, in all the examples in the paper, we used an error tolerance of 2% in the PCA projection. For the horse example, this error tolerance reduces the size of the strain energy vector $\mathbf{W}$ from 1595 to 54 components.

5 Tessellation of Fragments

Our fracture animation method allows concave crack surfaces through the artist control methods described in Section 3.4, but curved crack surfaces complicate the tessellation step. In this section, we propose a highly parallel algorithm to tessellate the fragments resulting from the CVD. Our algorithm exploits the tetrahedral mesh used for deformation computations, and defines a simple local tessellation procedure inside each tetrahedron indepen-

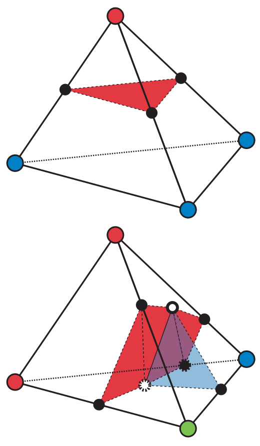

Figure 10: *Four canonical tessellation schemes of a tetrhaedron. The nodes of the tetrahedron are colored based on the fragment they belong to, and the fracture vertices are colored in the following way: black for intersections of Voronoi sheets and tetrahedral edges, white for intersections of Voronoi edges and tetrahedral faces, and yellow for Voronoi vertices.*

dently. It shares the philosophy of the marching tetrahedra algorithm [Guéziec and Hummel 1995], but it tessellates surfaces that are equidistant to Voronoi sites instead of tessellating isosurfaces.

We tessellate the CVD by approximating its intersection with a tetrahedral mesh. We start by labeling the nodes of each tetrahedron according to their closest Voronoi site (i.e., fragment center), and then the tessellation remains local to each tetrahedron. Based on the labeling of nodes, we select one out of four canonical tessellation schemes (shown in Fig. 10), which entirely define the arrangement of fracture triangles inside the tetrahedron.

The intersection of the CVD with the tetrahedral mesh may produce three types of fracture vertices:

- If the two nodes of a tetrahedral edge are labeled differently, we compute a fracture vertex as the intersection of a Voronoi sheet with the edge.
- If the three nodes of a tetrahedral face are labeled differently, we compute a fracture vertex as the intersection of a Voronoi edge with the face.
- If all four nodes of a tetrahedron are labeled differently, a Voronoi vertex defines a fracture vertex inside the tetrahedron.

After all fracture vertices are computed, we proceed with the parallel tessellation of all tetrahedra. For each tetrahedron, based on the number of nodes closer to each of the four possible different sites, the four canonical tessellation schemes can be summarized as follows (see Fig. 10):

- $(3, 1, 0, 0)$: There are 3 fracture vertices at edges, and 1 fracture triangle.
- $(2, 2, 0, 0)$: There are 4 fracture vertices at edges, and 2 fracture triangles.
- $(2, 1, 1, 0)$: There are 5 fracture vertices at edges, 2 more at faces, and 5 fracture triangles.
- $(1, 1, 1, 1)$: There are 6 fracture vertices at edges, 4 more at faces, 1 due to a Voronoi vertex, and 12 fracture triangles.

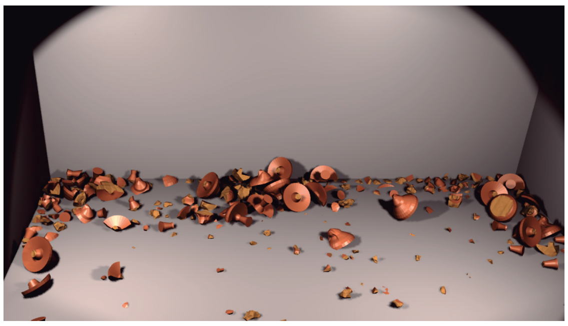

Figure 11: *Breaking Mushrooms. The user throws and drags around mushrooms interactively. The complete simulation runs at an average rate of 14.80 fps, and it consists of 250 fragments at the end.*

All three types of fracture vertices can be computed following a common definition: Given a simplex with all its M nodes closer to M different Voronoi sites, the fracture vertex is given by the barycentric combination of the nodes which is equidistant to all sites. We propose an approximate computation of the fracture vertex, based on the barycentric interpolation of distances to the Voronoi sites. We construct a matrix D of size $M \times M$, where each term d_{ij} stores the distance from node n_i to the Voronoi site p_j, which is the site closest to node n_j. Then, the barycentric coordinates b of the fracture vertex, and the distance d to the Voronoi sites are given by the solution to the system

$$\begin{pmatrix} D & -[1] \\ [1]^T & 0 \end{pmatrix} \begin{pmatrix} b \\ d \end{pmatrix} = \begin{pmatrix} [0] \\ 1 \end{pmatrix}. \qquad (13)$$

$[0]$ and $[1]$ represent column vectors of 0s and 1s of size M.

For an edge, the fracture vertex is guaranteed to lie inside, but this is not true for a face or a tetrahedron. In such cases, we propose an approximation of the fracture vertex that optimizes the smoothness of the approximate Voronoi sheets, but is constrained to lie inside the face or tetrahedron, therefore guaranteeing the locality of the tessellation algorithm. Instead of considering arbitrary positions inside a tetrahedron, we reduce the complexity of the problem by choosing the approximate fracture vertex out of the set of fracture vertices at lower dimensional simplices or their centroid. Constraining the tessellation to each tetrahedron simplifies the computational process, but also improves the smoothness of fracture surfaces.

In our examples, we use tetrahedral meshes of moderate resolution that embed the high-resolution triangle surfaces. Then, the tessellation requires a final step of clipping fracture surfaces against the original triangle-based surface, similar to the process described in [Steinemann et al. 2006]. To accelerate this clipping operation, as a preprocess we store in each tetrahedron a list of the surface triangles it intersects. Then, when a tetrahedron is fractured, we test for intersections its fracture triangles and its embedded surface triangles, and we triangulate the surface triangles using the Triangle library [Shewchuk 1996].

6 Results

We have integrated our fracture algorithm in the Bullet Physics simulation engine. The objects are simulated as rigid bodies until a collision is detected, and then we compute the resulting deformation field using a quasi-static finite element formulation. We then apply our fracture algorithm, and insert the resulting fragments as rigid bodies in the Bullet scene.

	#tris	#tetras	#nodes	high-dim	RBFs	Frac. deg.	CVD (iters)	CVD (ms)	Tessel. (ms)
Horse (Figs. 1 & 3)	38016	5256	1595	74	38	7 / 67	4 / 15	15 / 37	65 / 233
Wall (Fig. 5)	4800	1408	582	-	18	3 / 18	7 / 3	7 / 7	24 / 58
Bunny (Fig. 9)	163584	5301	1477	147	45	3 / 48	13 / 18	49 / 67	213 / 272
Mushrooms (Fig. 11)	768	1309	433	250	90	5 / 81	10 / 18	26 / 21	23 / 45

Table 1: *Simulation statistics for several benchmarks. First, we indicate the resolution of the models, the size of their high-dimensional embedding space for interior distance computation, and the number of RBF centers used for example-based computation of the fracture degree. Then, for two different fracture examples with each model, we indicate the fracture degree, the number of iterations needed to compute the CVD, and the time to compute the CVD and the tessellation (in ms.).*

We have executed several benchmarks in order to analyze different aspects of our algorithm. All experiments were executed on a 3.4GHz Intel i7-2600 processor with 8GB of RAM, using 8 cores for the parallel tessellation of fragments.

The various examples in the paper highlight the richness and versatility of fracture patterns with our method, as well as the influence of collision scenarios. As expected, smaller fragments are concentrated in the zone of impact. Moreover, if an object falls on a thin part (e.g., when the bunny in Fig. 9 falls on its ears), the object naturally shatters into more fragments than when an object falls on a large part (e.g., when the horse in Fig. 3 falls on its head). Please see the accompanying video. Also as expected, the fracture degree grows on average as objects are dropped from a higher height, as shown in Fig. 8. Fig. 3 demonstrates the expected behavior when the material is modified. When the horse is dropped from the same height and with the same orientation, it breaks into more and smaller fragments with a choice of weaker material (lower threshold γ).

We have already discussed the accuracy of the example-based approach for computing the fracture degree in Section 4.2. Fig. 9 shows a comparison of a bunny whose fracture degree is computed iteratively, vs. a bunny whose fracture degree is quickly computed based on examples. We observe that the distribution of fragments with the example-based approach is very similar to the iterative method, but it reduces the computation of the final CVD from more than one second to under 70 ms. Fig. 8 indicates the same results for a large battery of tests executed on the horse in Fig. 3.

In Table 1, we show some statistics and timings to compute a single fracture event in our benchmarks. For a small fracture degree, the cost is dominated by the CVD computation, but as the fracture degree grows, the cost is rapidly dominated by the tessellation, in particular by the tessellation of the original surface, as noted by the data from the bunny benchmark. Our implementation takes advantage of multi-core CPU architectures to parallelize the tessellation, but further performance improvements would be possible by using the general triangulation routines in Triangle [Shewchuk 1996] only in complex cases. It is worth noting that, for the wall model, which is originally convex, we simply used the Euclidean metric as a suitable interior distance metric.

The mushroom benchmark in Fig. 11 shows a practical application of our algorithm in an interactive scene. A user throws and drags around the mushrooms interactively, producing fractures and collisions. As indicated in Table 1, fracture events in this scene reach a cost of up to 66 ms with our algorithm. The complete simulation, including collision handling with the Bullet engine, runs at an average rate of 14.80 fps, and it contains 250 fragments. The interactive horse benchmark in Fig. 1 shows how fracture patterns adapt to the various collision scenarios. This example also demonstrates the plausibility of example-based fracture. The animation runs at an average rate of 21.82 fps, and it consists of 69 fragments at the end.

7 Discussion and Future Work

We have presented an algorithm to simulate brittle fracture that combines the flexibility and plausibility of physically based methods, with the efficiency and artist control of Voronoi-based geometric methods. The major components of our algorithm are a formulation of fracture as the computation of an interior CVD, an example-based learning method to accelerate the computations, and a highly parallel local tessellation method based on a tetrahedral lattice. In our implementation, the tetrahedral lattice is the same tetrahedral mesh used for computing elastic deformations, but our algorithm could easily be extended to other deformation algorithms. It would be sufficient to evaluate deformation energies at the nodes of the tetrahedral lattice.

The major limitation of our approach is that it relies heavily on preprocessing, at several levels. First, the computation of the high-dimensional embedding space for the evaluation of interior distances requires a precomputation step [Rustamov et al. 2009]. Second, the example-based computation of the fracture degree requires the precomputation of multiple simulation examples for each object. In our benchmarks we have computed up to 1152 examples per object (6 heights $\times$ 64 orientations $\times$ 3 instances). With an average cost of 2 seconds per example, this amounts to a preprocessing cost of 38 minutes. Note that this preprocessing can be trivially parallelized. Fortunately, our results indicate that a general training procedure, based on dropping objects from various heights and with various orientations, is sufficient even for runtime simulations with very different dynamics and collisions.

From our experiments, we can easily identify tessellation, in particular the tessellation of the fine surface, as the major bottleneck of the runtime algorithm. As mentioned in the previous section, further optimizations would be possible by identifying simple cases that do not require the general triangulation solution of Triangle [Shewchuk 1996]. An additional bottleneck is collision handling at fracture events. Recently, Glondu et al. [2012b] have developed algorithms to efficiently update collision detection data structures and reduce the cost of collision queries at fracture events.

To conclude, we believe that our work also poses interesting questions about the outreach of current geometry-based and physically based methods for simulating fracture. A combination of both methods, as done in our algorithm, produces plausible results with high flexibility for artist control. It would be interesting to extend artist control tools to handle data from real-world fractures. Another open question is whether Voronoi methods could be used for handling partial and/or ductile fracture.

Acknowledgements

This research is supported in part by the Spanish Ministry of Economy (TIN2012-35840) and by the European Research Council (ERC-2011-StG-280135 Animetrics). We would also like to thank Ming C. Lin and Alberto Sánchez for initial discussions.

References

BAO, Z., HONG, J.-M., TERAN, J., AND FEDKIW, R. 2007. Fracturing rigid materials. *IEEE Transactions on Visualization and Computer Graphics 13*, 2 (Mar.), 370–378.

BIELSER, D., GLARDON, P., TESCHNER, M., AND GROSS, M. 2003. A state machine for real-time cutting of tetrahedral meshes. In *Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, 377–.

CHEN, S., COWAN, C. F. N., AND GRANT, P. M. 1991. Orthogonal least squares learning algorithm for radial basis function networks. *IEEE Transactions on Neural Networks 2*, 2, 302–309.

COIFMAN, R. R., LAFON, S., LEE, A. B., MAGGIONI, M., WARNER, F., AND ZUCKER, S. 2005. Geometric diffusions as a tool for harmonic analysis and structure definition of data: Diffusion maps. In *Proceedings of the National Academy of Sciences*, 7426–7431.

DE GOES, F., GOLDENSTEIN, S., AND VELHO, L. 2008. A hierarchical segmentation of articulated bodies. In *Proceedings of the Symposium on Geometry Processing*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, SGP '08, 1349–1356.

DU, Q., FABER, V., AND GUNZBURGER, M. 1999. Centroidal voronoi tessellations: Applications and algorithms. *SIAM Rev. 41*, 4 (dec), 637–676.

FLOATER, M. S. 2003. Mean value coordinates. *Computer Aided Geometric Design 20*, 2003.

FRÉCHET, M. 1948. *Les éléments aléatoires de nature quelconque dans un espace distancié*. Annales de l'Institut Henri Poincaré. Gauthier-Villars.

GLONDU, L., MARCHAL, M., AND DUMONT, G. 2012. Real-time simulation of brittle fracture using modal analysis. *IEEE Trans. on Visualization and Computer Graphics*.

GLONDU, L., SCHVARTZMAN, S. C., MARCHAL, M., DUMONT, G., AND OTADUY, M. A. 2012. Efficient collision detection for brittle fracture. In *Proc. of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 10.

GUÉZIEC, A., AND HUMMEL, R. 1995. Exploiting triangulated surface extraction using tetrahedral decomposition. *IEEE Transactions on Visualization and Computer Graphics 1*, 4 (Dec.), 328–342.

HELLRUNG, J., SELLE, A., SHEK, A., SIFAKIS, E., AND TERAN, J. 2009. Geometric fracture modeling in bolt. In *SIGGRAPH 2009: Talks*, ACM, New York, NY, USA, SIGGRAPH '09, 7:1–7:1.

LIU, N., HE, X., LI, S., AND WANG, G. 2011. Meshless simulation of brittle fracture. *Comput. Animat. Virtual Worlds 22*, 2-3 (Apr.), 115–124.

LLOYD, S. P. 1982. Least squares quantization in PCM. *IEEE Transactions on Information Theory 28*, 129–137.

MOLINO, N., BAO, Z., AND FEDKIW, R. 2004. A virtual node algorithm for changing mesh topology during simulation. *ACM Transactions on Graphics 23*, 3 (Aug.), 385–392.

MULLER, M., MCMILLAN, L., DORSEY, J., AND JAGNOW, R. 2001. Real-time simulation of deformation and fracture of stiff materials. In *Proceedings of the Eurographic workshop on Computer animation and simulation*, 113–124.

MULLER, M., CHENTANEZ, N., AND KIM, T. Y. 2013. Real time dynamic fracture with volumetric approximate convex decompositions. *ACM Transactions on Graphics 32*, 4.

O'BRIEN, J. F., AND HODGINS, J. K. 1999. Graphical modeling and animation of brittle fracture. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, 137–146.

O'BRIEN, J. F., BARGTEIL, A. W., AND HODGINS, J. K. 2002. Graphical modeling and animation of ductile fracture. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, 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.

RAGHAVACHARY, S. 2002. Fracture generation on polygonal meshes using voronoi polygons. In *ACM SIGGRAPH 2002 conference abstracts and applications*, 187–187.

RUSTAMOV, R., LIPMAN, Y., AND FUNKHOUSER, T. 2009. Interior distance using barycentric coordinates. *Computer Graphics Forum (Symposium on Geometry Processing) 28*, 5 (July).

SHEWCHUK, J. R. 1996. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *Applied Computational Geometry: Towards Geometric Engineering*. 203–222.

SIFAKIS, E., DER, K. G., AND FEDKIW, R. 2007. Arbitrary cutting of deformable tetrahedralized objects. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, 73–80.

STEINEMANN, D., OTADUY, M. A., AND GROSS, M. 2006. Fast arbitrary splitting of deforming objects. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, 63–72.

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

ZHENG, C., AND JAMES, D. L. 2010. Rigid-body fracture sound with precomputed soundbanks. In *ACM SIGGRAPH 2010 papers*, 69:1–69:13.

Appendix: Proof of Theorem 1

Given an object Ω and its high-dimensional image $\bar{\Omega}$, the weighted centroid $\bar{p}^*$ of $\bar{\Omega}$ according to the Euclidean metric is obtained by minimizing the cost function $\bar{f}(\bar{p}) = \int_\Omega \|\bar{x} - \bar{p}\|^2 \, W(x)\, dx$. The centroid is

$$\bar{p}^* = \frac{\int_\Omega \bar{x}\, W(x)\, dx}{\int_\Omega W(x)\, dx}. \tag{14}$$

On the other hand, the Fréchet mean p^* of Ω according to the interior distance metric is obtained by minimizing the cost function $f(p) = \int_\Omega \operatorname{dist}(x,p)^2\, W(x)\, dx$. Since $\operatorname{dist}(x,p) = \|\bar{p} - \bar{x}\|$, the gradient of $f(p)$ is:

$$\nabla_p f = 2\, \nabla_p \bar{p} \left(\left(\int_\Omega W(x)\, dx \right) \bar{p} - \int_\Omega \bar{x}\, W(x)\, dx \right). \tag{15}$$

By substituting $p = p^*$, i.e., the low-dimensional image of the high-dimensional centroid $\bar{p}^*$ defined in (14), we conclude trivially that $\nabla_p f(p^*) = 0$; therefore, p^* is a Fréchet mean of Ω.