Eurographics/ ACM SIGGRAPH Symposium on Computer Animation (2006)
M.-P. Cani, J. O'Brien (Editors)

# Generating Surface Crack Patterns

Hayley N. Iben James F. O'Brien

University of California, Berkeley

**Abstract**

*We present a method for generating surface crack patterns that appear in materials such as mud, ceramic glaze, and glass. To model these phenomena, we build upon existing physically based methods. Our algorithm generates cracks from a stress field defined heuristically over a triangle discretization of the surface. The simulation produces cracks by evolving this field over time. The user can control the characteristics and appearance of the cracks using a set of simple parameters. By changing these parameters, we have generated examples similar to a variety of crack patterns found in the real world. We assess the realism of our results by comparison with photographs of real-world examples. Using a physically based approach also enables us to generate animations similar to time-lapse photography.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based modeling

## 1. Introduction

Surface crack patterns occur on a variety of materials, including glass, mud, and ceramic glaze. These cracks are often the result of shrinkage of the object's surface area. For example, mud in a river bed dries faster on the surface than in the underlying soil, causing stress to build. The mud develops cracks to relieve this stress. In ceramics, glaze with a different coefficient of thermal expansion than that of the pottery will accumulate stress during the cooling process. When this stress is too high, the glaze cracks. We have developed a method to model the crack patterns created by these types of processes, as demonstrated by the example of "crackle glass" shown in Figure 1.

Several physically based simulation papers present methods to fracture objects, such as a finite element approach [OH99] and the virtual node algorithm [MBF04]. Such fracturing methods use a second-order dynamic simulation, moving the nodes according to calculated forces. Our method does not simulate dynamic forces or elastic waves, so the nodes do not move or have velocity. Instead, we use a first-order quasi-static system, a method that is inherently more stable than a second-order system. Our choice also avoids certain problems caused by moving nodes, such as collisions or mesh tangling issues.

We generate crack patterns on a triangle discretization of the input surface. This triangulation can be arbitrary, although our results are better if the surface is 2-manifold and the size of the triangles does not vary too drastically over the mesh. We define a stress field over this triangle mesh and evolve it iteratively due to processes such as elastic relaxation or shrinkage. Areas of high tensile stress form cracks which in turn alleviate stress in the surrounding region.

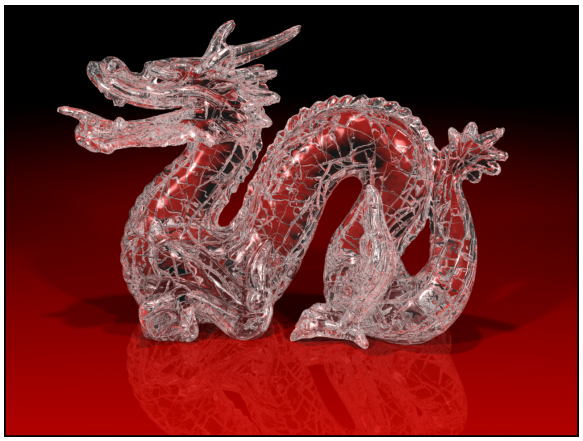

**Figure 1:** *A crackle glass dragon, generated by uniform shrinkage of the surface. This example required 2.7 hours of computation with an input mesh of 75,000 triangles.*

The algorithm we use to determine where cracks occur and how they propagate is derived from the method originally presented in [OH99]. Cracks occur at existing vertices in the triangle mesh and they propagate through a process of local remeshing. The tip of an advancing crack always

corresponds to a vertex location, allowing the formation of smooth and realistic-looking crack patterns.

Our method is a novel combination, obtaining the realism of a physically correct simulation but maintaining the controllability of a heuristic based method. We provide the user with several parameters to control the appearance and characteristics of the crack patterns. Some of these parameters are based on physical properties of the material while others are intuitive heuristics. By tuning these quantities, the user can produce a variety of results using the same system.

## 2. Background

The methods used in computer graphics for generating crack patterns can be grouped into two categories. One area of research follows a non-physical approach for creating crack patterns, such as mapping crack patterns to a surface. Other papers use physically based methods to crack or fracture objects. Our method falls primarily into the second category, using a finite element discretization common in physically based approaches. However, rather than accurately simulating the elastic deformation of an object, we instead use heuristic methods to define a stress field directly. The resulting approach generates realistic crack patterns while still allowing the user a substantial amount of control.

Several researchers developed methods for modeling fracture in solid materials, primarily using physically based simulation. Terzopoulos and colleagues introduced simulating elastic [TPBF87] and inelastic deformation of objects, including fracture [TF88]. A mass-spring system was later used to simulate fracture [NTB*91]. Finite elements have been widely used to simulate brittle fracture [OH99], ductile fracture [OBH02], elasto-plastic materials and fracture interactively [MG04], and the fracture and deformation of voxelized surface meshes [MTG04]. Other algorithms include generating fracture on elastic and plastic materials with the virtual node algorithm [MBF04], a membrane-bending model for thin shell objects [GSH*04], and a meshless framework [PKA*05].

In addition to being used for objects that are being broken or torn apart, physically based methods have also been used to create surface cracks patterns. A mass-spring system was used to reproduce crack patterns in microsphere monolayers [SM88], model tree bark [FP96], crack a surface [HTK98], and crack a volume [HTK00]. Gobron and Chiba used cellular automata to crack multi-layer surfaces [GC00] and simulate materials peeling off of surfaces [GC01]. Paint cracking and peeling was also simulated using a two-layered model on a 2D grid [PPD02]. Federl and Prusinkiewicz used wedge-shaped finite elements to model cracks formed by drying mud and tree bark [FP02] [FP04].

There have been several non-physical approaches to generating surface cracks. One type of approach maps some form of a crack pattern to an object's surface [HTCY04] and then carves out a volume to generate crack depth [MGDA04] [DGA05]. Others form cracks on a 2D surface to replicate Batik painting cracks [WvOC04] or create cracks similar to an input image [Mou05]. These methods use an input pattern or image to generate surface cracks. In contrast, our method creates cracks from a stress field defined heuristically on the object's surface.

**Figure 2:** *As an artistic effect, we initialized the stress field to uniform tension with some bias to crack in the principle curvature directions. The result of using this heuristic is demonstrated by the vertical cracks of the angel's arm and the cracks following the folds of fabric. Total computation time was 4.0 hours with an input mesh of 105,772 triangles.*

Outside of computer graphics, fracture mechanics has been extensively studied in engineering and a wide range of methods to analyze the failure of materials is available. Researchers use finite element methods, finite differencing methods, or boundary integral equations to simulate the failure of real materials. An overview of methods used in this field can be found in [And04] and [Nis97]. Physics researchers have studied specific crack pattern problems, such as drying mud [Kit99], alumina/water slurry [SdBGM00], or both mud and ceramics [BPC05] [BPA*05].

## 3. Stress, Forces and the Separation Tensor

As in previous approaches [OH99][OBH02][FP04], we use a finite element discretization with some form of local remeshing. However, we generate cracks on the surface of objects instead of fracturing a volume. We use a triangle mesh discretization of the model where other methods use 3D elements. This reduced dimension of our elements simplifies the simulation computations. We initialize our stress field directly with heuristics instead of using a full finite element simulation. This simplification gives the user control over the resulting crack patterns and decreases the initial stress computation costs. Instead of moving the nodes when fracture occurs, such as in [OH99][HTK00][FP04][MBF04], our method keeps the node stationary and updates the stress field with a first-order quasi-static system.

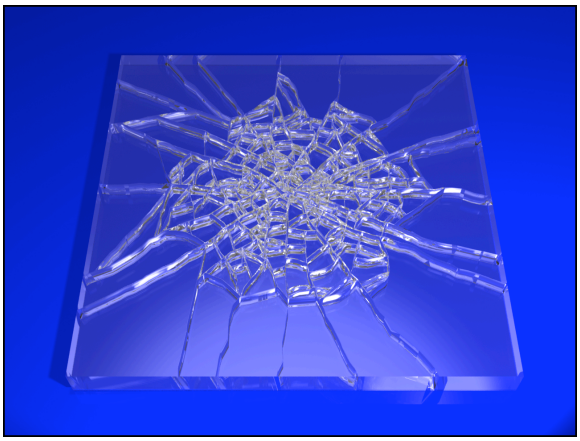

**Figure 3:** *Cracks generated from initializing the stress field with a pattern modeling impacts in flat glass. The field is evolved by the relaxation process, causing the cracks to propagate. Total computation time was 20 seconds with an input mesh of 4,804 triangles.*

### 3.1. The Algorithm

To generate the crack patterns, we define a stress field over the triangles and evolve this field over time using crack generation processes. We then analyze the stress field to determine where cracks form in the mesh. These cracks are introduced as free boundaries that affect the relaxation process and stress field. After cracks form, we update the stress field and repeat the process until either cracks cannot be added or the user is satisfied with the result.

The following pseudo-code describes our algorithm:

1. Initialize the stress field according to heuristics and optionally evolve it with relaxation.
2. Compute the failure criteria for each node and store this criteria in a priority queue.
3. While failure can occur and we want to continue:
   a. Crack the mesh at the node associated with the top of the priority queue.
   b. Evolve the stress field:
      i. Perform relaxation
      ii. Optional: Add shrinkage tension and/or curvature biasing.
   c. Update the mesh information and priority queue.
4. For display: either (a) post-process the mesh by moving the vertices to give the cracks width and filling in the gaps with side-walls for the cracks or (b) directly render crack edges.

### 3.2. Stress Field

We define the stress field by storing tensors at the triangle centers and treating the stress as a constant over the area of the triangle. This value reflects the force distribution per unit area within the element. When the stress field is initialized to zero, the surface is in equilibrium. To simulate drying and shrinkage, the stress tensor can be initialized to uniform tension, indicating all directions pull uniformly. Alternatively, we can use the curvature tensor to initialize stress, indicating that the high curvature areas are more prone to cracks. The user could also manually specify areas of high tension on the surface. We can also introduce some amount of random noise into the stress field to model material inhomogeneities.

### 3.3. Forces

The stress tensor of an element encapsulates the amount of force a small piece of material exerts on a node. The force acting on node $i$ by an element is

$$\boldsymbol{f}_{[i]} = -A \sum_{j=1}^{3} \boldsymbol{p}_{[j]} \sum_{k=1}^{2} \sum_{l=1}^{2} \beta_{jl} \beta_{ik} \sigma_{kl} \qquad (1)$$

where $A$ is the area of the element, $\beta$ the barycentric basis matrix, $\boldsymbol{p}$ the node's position in world coordinates and $\sigma$ the stress tensor. To calculate the total force for a given node, we sum the forces exerted by the surrounding elements.

The derivation of this equation and the barycentric basis matrix can be found in [OH99]. The primary difference between our equations and theirs is due to our discretization choice. We use triangles instead of tetrahedra, so the stress tensor is a two dimensional quantity in the plane of its triangle and the $\beta$ matrix is $3 \times 3$ instead of $4 \times 4$. This reduction in dimension lowers the computation cost at each step of the simulation. Note that while the stress tensor is defined in a local 2D coordinate system for each triangle, the above expression produces a force vector in the common 3D coordinate system where the mesh is embedded.

### 3.4. The Separation Tensor

We analyze the stress field to determine where to generate cracks by using the formulation of the separation tensor $\varsigma$ presented in [OH99]. They form the tensor by balancing the tensile and compressive forces exerted on a node. Because the forces are calculated in 3D, we can use them directly in the equation

$$\varsigma = \frac{1}{2} \left( -\mathbf{m}(\boldsymbol{f}^+) + \sum_{\boldsymbol{f} \in \{\boldsymbol{f}^+\}} \mathbf{m}(\boldsymbol{f}) + \mathbf{m}(\boldsymbol{f}^-) - \sum_{\boldsymbol{f} \in \{\boldsymbol{f}^-\}} \mathbf{m}(\boldsymbol{f}) \right) \qquad (2)$$

where $\boldsymbol{f}^+$ is the tensile force, $\boldsymbol{f}^-$ the compressive force, and $\mathbf{m}$ a function computing the outer product of a vector. To compute these forces, we decompose the stress tensor into tensile $\sigma^+$ and compressive $\sigma^-$ components and use Equation (1).

We use this tensor to determine whether a crack occurs at a node and the resulting direction. We take the eigendecomposition of $\varsigma$ to find the largest positive eigenvalue, $\nu^+$.

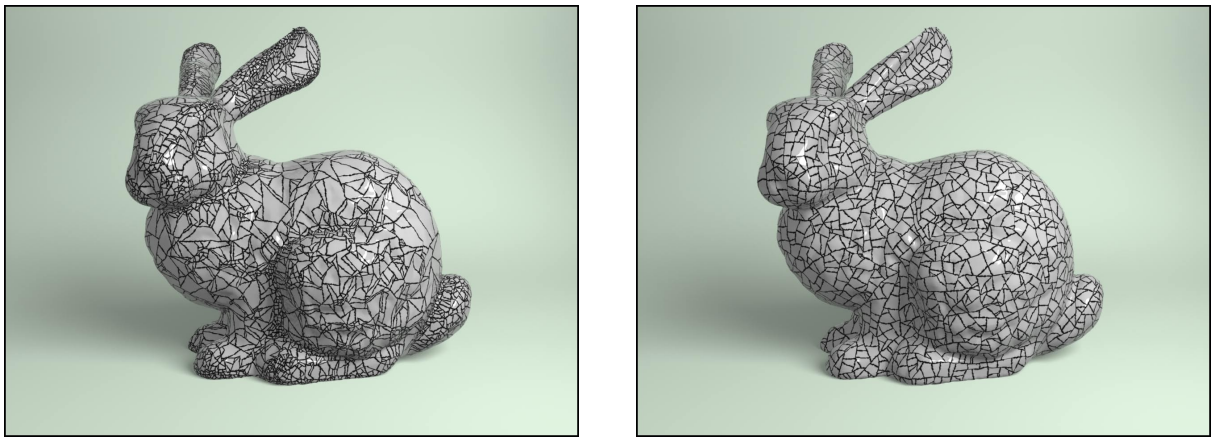

**Figure 4:** *Curvature can be used to control crack formation, as demonstrated by the image on the left requiring 3.2 hours computation. Notice the high concentration of cracks around areas of high curvature, such as the ears, neck and leg. The right image was simulated using only uniform shrinkage of the surface, resulting in more evenly spaced cracks and requiring 2.5 hours. The input mesh size was 51,382 triangles for both examples.*

The material fails at a node when $v^+$ is greater than the material toughness, $\tau$. A crack occurs at this node in the crack plane defined by $\hat{\mathbf{n}}^+$, the eigenvector corresponding to $v^+$. We discuss this process further in Section 4.2.

## 4. Mesh Updates

After calculating an initial stress field, our method performs an iterative procedure to generate crack patterns. This process interleaves the evolution of the stress field and propagation of cracks. The stress field changes primarily according to elastic relaxation. The user can further control its evolution by imposing various conditions, such as shrinkage tension, impact stress-patterns, or bias to crack on high curvature regions. Cracks occur when large stresses produce a large separation eigenvalue in the mesh. The cracks create open boundaries in the mesh that alleviate perpendicular components of the nearby stress field. The physically based interaction between the stress field evolution and crack generation algorithm is what leads to the creation of interesting crack patterns.

### 4.1. Relaxation

Rather than using a second-order dynamic system to model the behavior of the mesh by integrating the motion of its vertices, we instead treat stress directly as an independent variable that evolves according to a first-order relaxation process. If stress were a scalar quantity, this process would simply be mesh-based diffusion. For tensor quantities that each live in distinct local 2D coordinate systems, the derivation is somewhat more complex. Fundamentally, our stress relaxation is also a diffusion process, as described below.

Other cracking methods have used relaxation to smooth the stress field after initialization [GC00], to reduce the tensile stress around cracks based on distance to the nearest crack [PPD02], or to recalculate the equilibrium state adaptively during crack formation [FP04]. We use relaxation to redistribute stress from areas of high stress to areas of low stress. To perform the relaxation, we use the virtual displacement of the nodes to calculate the change in the stress field. We compute the forces exerted on nodes with Equation (1), encapsulating the effect of the stress field on the nodes. We assume that if we were modeling the displacement of the nodes, they would be moved by these forces. However, by not actually moving the nodes, we avoid the time-step and stability issues that would otherwise result.

Let $\boldsymbol{F}_{[n]}$ be the sum of all forces $\boldsymbol{f}_{[n]}$ on node $n$ exerted by the surrounding elements. We calculate the virtual displacement based on the total forces acting on node $n$ by

$$\Delta \boldsymbol{p}_{[n]} = \Delta t \, \boldsymbol{F}_{[n]} \tag{3}$$

where $\Delta t$ is the time step controlling the rate of relaxation.

We assume that stress in an element is linearly proportional to its strain with proportionality constant 1. We use Green's strain tensor, $\varepsilon$, to determine the amount of deformation of an element around the node. This tensor is represented by a $3 \times 3$ matrix defined by

$$\varepsilon_{ij} = \frac{1}{2}\left(\frac{\partial \boldsymbol{x}}{\partial u_i} \cdot \frac{\partial \boldsymbol{x}}{\partial u_j} - \delta_{ij}\right) \quad . \tag{4}$$

If we assume that virtual displacements are piecewise linear over the mesh, then we can define the partial derivatives as

$$\frac{\partial x_i}{\partial u_j} = \sum_{k=1}^{3} p_{[k]i}\beta_{kj} \quad . \tag{5}$$

We need the strain tensor's change in the element, so we compute the derivative with respect to the node positions

$$\frac{\partial \varepsilon_{ij}}{\partial p_{[n]r}} = \frac{1}{2}\left(\sum_{m=1}^{3} p_{[m]r}\beta_{ni}\beta_{mj} + \sum_{m=1}^{3} p_{[m]r}\beta_{mi}\beta_{nj}\right) \quad . \tag{6}$$

Because we have assumed the relationship between strain and stress is linearly proportional, we compute the change of stress in an element with node $n$ as

$$\Delta\sigma = \frac{\partial \varepsilon}{\partial \boldsymbol{p}_{[n]}} \Delta \boldsymbol{p}_{[n]} \quad . \tag{7}$$

At every time step, we use the force on the node to determine its virtual displacement from Equation (3). We update the element's stress tensor by accumulating $\Delta\sigma$ into it. After changing the stress, we update the forces on the nodes. This process repeats iteratively. Periodically, the separation tensor at each node is updated and the crack-generation routine, described in Section 4.2, is invoked.

We can treat boundary edges in our mesh as either free or fixed. For fixed boundaries, we simply zero the net force acting on the fixed boundary vertices prior to computing stress updates. Free boundaries require no special action by the algorithm. The boundary edges introduced during crack generation are treated as open boundaries so that relaxation will relieve stress components perpendicular to the crack edges.

The above equations implement a simple forward Euler step of stress relaxation. Just as faster methods have been used for integrating diffusion in the context of mesh smoothing, we could likewise accelerate our algorithm by using a more sophisticated integration scheme. However, we have so far found our computation times to be sufficiently fast and therefore have not invested time implementing a more sophisticated, and presumably faster, method.

In addition to relaxation, there are other ways to update the stress field. We model uniform shrinkage of the object by adding $c\,\mathbf{I}$, where $c$ is a positive constant factor, to the stress tensor of each element. This amounts to generating uniform tension in the mesh, as illustrated by Figures 6 and 7. Additionally, we want to avoid stress patterns that are identically uniform as they create unrealistic artifacts in the resulting crack pattern. The discretization error in an unstructured mesh adds some inherent randomness to the stress field. We can also add randomness to the tensor explicitly.

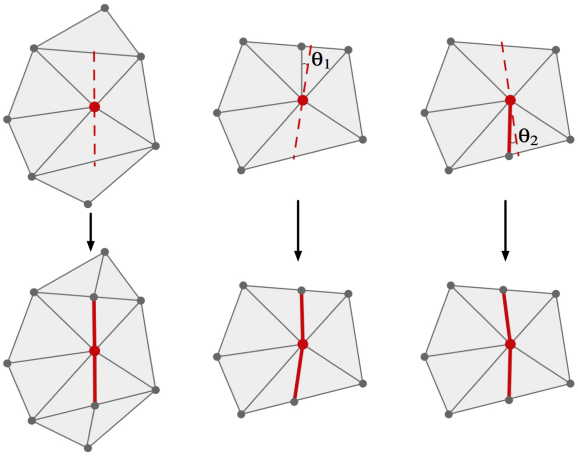


**Figure 5:** *The dotted line (crack plane) gives the location of a new crack. Bold edges are crack edges. Left: After crack edges are added, we locally remesh to ensure all elements are triangles. Middle: Cracks are snapped to an existing edge if $\theta_1$ is less than a set threshold (ex: 25°). Right: Cracks are also snapped to an existing crack edge when $\theta_2$ is less than a set threshold (ex: 15°), avoiding back-cracking.*

**Figure 6:** *Example of a crackle glaze teapot, generated by initializing the stress field to uniform shrinkage and evolving it by adding uniform tension. This example required 4.9 hours computation with an input mesh of 60,000 triangles.*

Another way to update the stress field is with the object's curvature. We estimate the principle curvature using the discrete operator given in [CSM03]. As an update, we add a specified amount of the curvature tensor to the stress tensor, causing areas with high curvature to be more prone to cracking. Alternatively, we can multiply the separation tensor by a scaling factor, such as Gaussian curvature, mean curvature or the Frobenius norm of the curvature tensor. We demonstrate the effect of using a scaling factor in Figures 4 and 10.

We can generate impact patterns on a surface by initializing the stress field appropriately. The stress field generated by an impact varies based on the material. For flat glass, radial cracks initially form outward from the impact point, followed by concentric cracks. We model this interaction by initializing the stress field to high stress radiating outward from the impact point and low stress in the concentric direction, as illustrated in Figure 3.

### 4.2. Generating Cracks

As described in Section 3.4, we use the separation tensor $\varsigma$ to determine where to crack the mesh. We insert the largest positive eigenvalue, $v^+$, of each node's $\varsigma$ to a priority queue. We then iteratively crack based on the largest values in the queue. When the top eigenvalue is larger than the material toughness threshold, $\tau$, a crack occurs at the corresponding node. We generate cracks in the direction perpendicular to

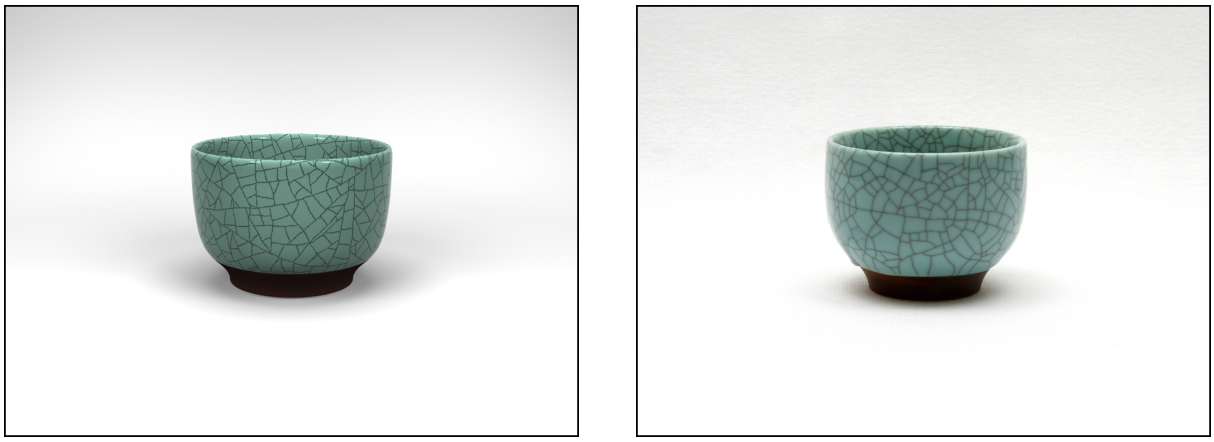

**Figure 7:** *A rendered example of a crackle glaze cup (left) compared to a photograph (right). Our algorithm generates this effect by initializing the stress field to uniform shrinkage over the surface and evolving it with uniform tension. Total computation time was 43 minutes with a 39,611 triangle input mesh.*

the corresponding eigenvector. To compute where new edges should be inserted in the mesh, we intersect the plane defined by $\hat{\mathbf{n}}^+$ with the surrounding triangles. The intersecting triangles are split, creating new free boundary edges in the mesh. We do not add a new crack edge if it is too close to a current crack edge, avoiding back-cracking as described in [OH99] and illustrated in Figure 5. This plane-triangle intersection is fast in comparison to splitting and remeshing tetrahedra, an advantage of computing cracks on a surface discretization.

The results generated by our method depend on the resolution of the discretization of the model, only generating a crack in the elements surrounding the node at each time step. We use the approach presented in [O'B00] to propagate the crack further. After cracking a node, we compute a residual value $\mathsf{v}^* = \mathsf{v}^+ - \tau$. This quantity modifies the separation tensor of a node by

$$\varsigma' = \varsigma + \alpha\, \mathbf{m}(\hat{\mathbf{n}}^+)\mathsf{v}^* \tag{8}$$

where $\alpha$ is an input parameter. Small values for $\alpha$ result in more jagged cracks while large values cause the cracks to propagate further in the same direction. The parameter $\alpha$ heuristically captures material inhomogeneities, effectively controlling the jaggedness of the cracks in the simulation, as illustrated in Figures 11 and 12. For our trials, we found $\alpha$ values between zero and one to generate nice results.

Small or poorly shaped elements can introduce numerical instabilities in the simulation. We attempt to avoid creating ill-shaped triangles by snapping crack edges to mesh edges if they are close, as described in [OH99] and illustrate in Figure 5. However, some slivers are large enough that edge snapping would create objectionable artifacts, yet still small enough that they would generate poorly conditioned elements. This problem can be traced back to very large singular values in the $\beta$ matrix. We initially compute a threshold based on the singular values of the input triangles' $\beta$ matrices. Each time we calculate the $\beta$ matrix of an element, we compute its SVD and examine the singular values. We reconstruct the $\beta$ matrix when we need to restrict its singular values to the threshold. The effect on the behavior of the simulation is that these small slivers become more compliant in the direction with large singularity. This change causes the system to remain numerically stable even with poorly shaped elements.

## 5. Post-Processing

During the simulation, we do not change the positions of the nodes. For the ceramic examples, we implemented a shader that interpolates information about crack locations to render the dark lines. This rendering technique is demonstrated in Figures 6 and 7.

For some of our examples, such as Figures 3 and 11, we wanted to create a visible gap at the crack locations. To do this, we sum the change in positions from Equation (3) for each node $n$ as we compute them in the relaxation process

$$\Delta_T \boldsymbol{p}_{[n]} = \sum_{i=1}^{R} \gamma \boldsymbol{F}_{[n]} \tag{9}$$

where $R$ is the total number of relaxation steps and $\gamma$ a parameter controlling the rate of movement. As a post-processing step, we displace the nodes by this vector. This displacement generates gaps in the mesh where the cracks occur. To fill the gaps, we build rectangles connecting the displaced crack node positions to the original positions, offset inward by a user specified amount. This creates nice sidewalls for the cracks, as illustrated in Figures 11 and 12. Other examples using this method include Figures 1 and 2. Note that this post-processing movement is performed after the simulation is finished. Poorly shaped or inverted triangles that may result do not create significant problems.

We can also add a curling effect to the crack edges, as found in mud and peeling paint. To model this phenomena,

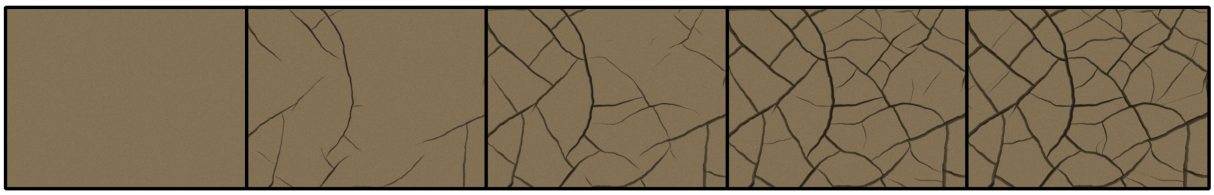

**Figure 8:** *An animation of mud drying for Figure 11, simulated by setting the stress field to uniform shrinkage of the surface.*

we use the magnitude of the result from Equation (9), giving the distance the node moved on the surface. We then move the crack nodes in the normal direction by a user-defined portion of this displacement. By iteratively propagating this change to the surrounding nodes, we produce a lifting effect. An example of curled mud is demonstrated in Figure 9.

## 6. Results and Discussion

We implemented the method described above in C++ and rendered our results using Pixie. The running time of the algorithm is dependent on the mesh resolution and concentration of cracks on the surface, with the largest time spent in the relaxation process. Because adding cracks increases the size of the mesh, the computation time for each iteration also increases. However, we found the simulation times to be reasonable on a Macintosh G5 computer with 2.5 GB of memory and are given in the captions.

Our method generates crack patterns similar to a variety of cracks found in nature. For example, ceramic glazes often contain cracks, sometimes generated intentionally by a crackle glaze process. Our method simulates this effect by uniformly shrinking the surface and evolving the stress field with uniform tension. The results of simulating this phenomenon appear in Figures 6 and 7. We also provide a comparison photograph with a Japanese teacup in Figure 7.

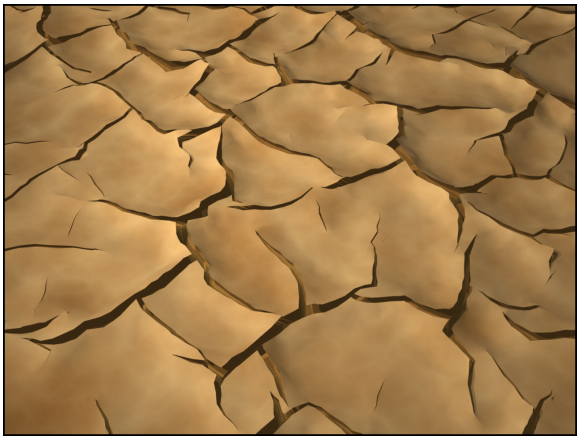

**Figure 9:** *Example of adding a curling effect to the cracked mud from Figure 12. This example required 2.0 minutes computation time on a 19,602 triangle input mesh.*

Crack patterns also occur in glass for various reasons. When the outer layer of molten glass cools rapidly, as happens when submerged in water, it solidifies without chance for annealing. The resulting thermal shock causes the solidified outer layer to crack, creating glasswork known as "crackle glass." Because this layer is adhered to the inner core of molten glass, the object retains its overall shape. We model this effect by initializing the stress field to uniform shrinkage, as illustrated in Figure 1.

Flat glass also cracks in a particular manner when struck by an object, as described in Section 4.1. We allow the user to specify a stress field on the object to model this impact effect and demonstrate our results in Figure 3. As a future direction, we could extend our system to allow users to specify arbitrary stress fields on objects.

During the mud drying process, cracks form to alleviate the stress that builds on the surface. Our method generates similar results by using uniform shrinkage, as demonstrated in the comparison between our results and photographs in Figures 11 and 12. The simulation generating these results differed in the parameter controlling crack propagation, $\alpha$. Capturing such a mud cracking process with time-lapse photography is time consuming because drying can be slow. Our method is able to generate animations of the cracking process similar to the real world by writing frames as the simulation progresses, as demonstrated in Figure 8. We also generate a curled version of Figure 12 in Figure 9. These examples demonstrate the ease of generating different results by varying a few intuitive parameters.

Our method can also use the object's geometry to influence crack patterns. For example, the simulation creating Figure 2 initialized the stress field to uniform shrinkage and then biased it in the principle curvature directions. This is illustrated by the vertical cracks in the left arm and the cracks along the folds of cloth. We can also bias the cracks to form in regions of high curvature by scaling the separation tensor. We use the Frobenius norm of the curvature tensor as a scale factor in Figure 4 and the sum of squares of the principle curvature values in Figure 10, generating a higher concentration of cracks in high curvature regions.

The method presented in this paper generates a variety of crack patterns by combining a physically based simulation with traditional appearance driven heuristics. With this approach, we obtain the realism of a physically correct simulation but keep the controllability of a heuristic based method.

We compare some of our results, such as the mud and ceramic glaze, with photographs to demonstrate the realism generated by our method. Although the images are different, the generated crack patterns have qualitatively similar characteristics to the real ones. Our approach could be incorporated into an artistic tool, enabling users to paint a stress field on an object to easily generate various cracking results using one system.

**Acknowledgments**

We thank the other members of the Berkeley Graphics Group, particularly Jonathan Shewchuk and Carlo Séquin, for their helpful criticism and comments. This work was supported in part by California MICRO 04-066 and 05-044, and by generous support from Apple Computer, Pixar Animation Studios, Autodesk, Intel Corporation, Sony Computer Entertainment America, and the Alfred P. Sloan Foundation. Iben was supported by NSF and GAANN fellowships.

**References**

[And04] ANDERSON T. L.: *Fracture Mechanics: Fundamentals and Applications*, 3 ed. CRC Press, 2004.

[BPA*05] BOHN S., PLATKIEWICZ J., ANDREOTTI B., ADDA-BEDIA M., COUDER Y.: Hierarchical crack patterns as formed by successive domain divisions. II. From disordered to deterministic behavior. *Physical Review E 75* (Apr 2005).

[BPC05] BOHN S., PAUCHARD L., COUDER Y.: Hierarchical crack patterns as formed by successive domain divisions. I. Temporal and geometrical hierarchy. *Physical Review E 75* (Apr 2005).

[CSM03] COHEN-STEINER D., MORVAN J.-M.: Restricted delaunay triangulations and normal cycle. In *Proceedings of the 19th Annual ACM Symposium on Computational Geometry* (2003), pp. 312–321.

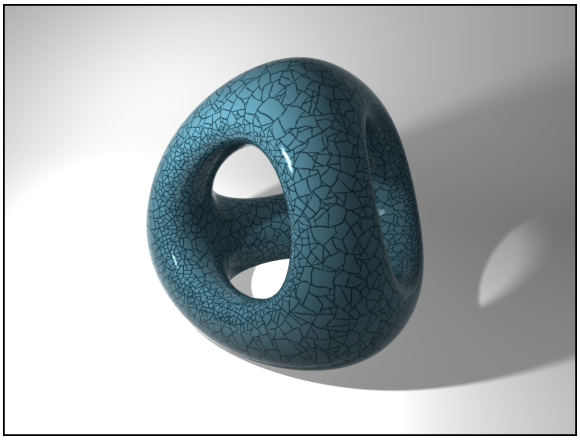

**Figure 10:** *Another example of an artistic effect achieved by using curvature to bias the concentration of cracks. In this example, the separation tensor was weighted by* $\kappa_1^2 + \kappa_2^2$ *where* $\kappa_1$ *and* $\kappa_2$ *are the principle curvature values. Computation time was 1.5 hours on a 30,008 element mesh.*

[DGA05] DESBENOIT B., GALIN E., AKKOUCHE S.: Modeling cracks and fractures. *The Visual Computer 21*, 8–10 (Sep 2005), 717–726.

[FP96] FEDERL P., PRUSINKIEWICZ P.: A texture model for cracked surfaces, with an application to tree bark. In *Proc. of the 7th Western Computer Graphics Symp.* (1996), pp. 23–29.

[FP02] FEDERL P., PRUSINKIEWICZ P.: Modelling fracture formation in bi-layered materials, with applications to tree bark and drying mud. In *Proc. of the 13th Western Computer Graphics Symp.* (2002).

[FP04] FEDERL P., PRUSINKIEWICZ P.: Finite element model of fracture formation on growing surfaces. *Lecture Notes in Computer Science 3037* (Jan 2004), 138–145.

[GC00] GOBRON S., CHIBA N.: Crack pattern simulation based on 3d surface cellular automaton. In *Proc. of the International Conference on Computer Graphics* (2000), pp. 153–162.

[GC01] GOBRON S., CHIBA N.: Simulation of peeling using 3d-surface cellular automata. In *Proceedings of the 9th Pacific Conference on Computer Graphics and Applications* (2001), pp. 338–347.

[GSH*04] GINGOLD Y., SECORD A., HAN J. Y., GRINSPUN E., ZORIN D.: A discrete model for inelastic deformation of thin shells. In *ACM SIGGRAPH / Eurographics Symposium on Computer Animation* (2004). Poster.

[HTCY04] HSIEH H.-H., TAI W.-K., CHIANG C.-C., YANG M.-T.: Flexible and interactive crack-like patterns presentation on 3d objects. *Lecture Notes in Computer Science 3280* (Jan 2004), 90–99.

[HTK98] HIROTA K., TANOUE Y., KANEKO T.: Generation of crack patterns with a physical model. *The Visual Computer 14*, 3 (1998), 126–137.

[HTK00] HIROTA K., TANOUE Y., KANEKO T.: Simulation of three-dimensional cracks. *The Visual Computer 16* (Nov 2000), 371–378.

[Kit99] KITSUNEZAKI S.: Fracture patterns induced by dessication in a thin layer. *Physical Review E 60*, 6 (Dec 1999).

[MBF04] MOLINO N., BAO Z., FEDKIW R.: A virtual node algorithm for changing mesh topology during simulation. *ACM Transactions on Graphics 23*, 3 (2004), 385–392.

[MG04] MÜLLER M., GROSS M.: Interactive virtual materials. In *Proceedings of Graphics Interface* (2004), pp. 239–246.

[MGDA04] MARTINET A., GALIN E., DESBENOIT B., AKKOUCHE S.: Procedural modeling of cracks and fractures. In *International Conference on Shape Modeling and Applications* (2004), pp. 346–349.

[Mou05] MOULD D.: Image-guided fracture. In *Proceedings of Graphics Interface* (2005), pp. 219–226.

[MTG04] MÜLLER M., TESCHNER M., GROSS M.: Physically-based simulation of objects represented by surface meshes. In *Proceedings of the Computer Graphics International* (2004), pp. 26–33.

[Nis97] NISHIOKA T.: Computational dynamic fracture mechanics. *International Journal of Fracture 86* (1997), 127–159.

[NTB*91] NORTON A., TURK G., BACON B., GERTH J., SWEENEY P.: Animation of fracture by physical modeling. *The Visual Computer 7*, 4 (1991), 210–219.

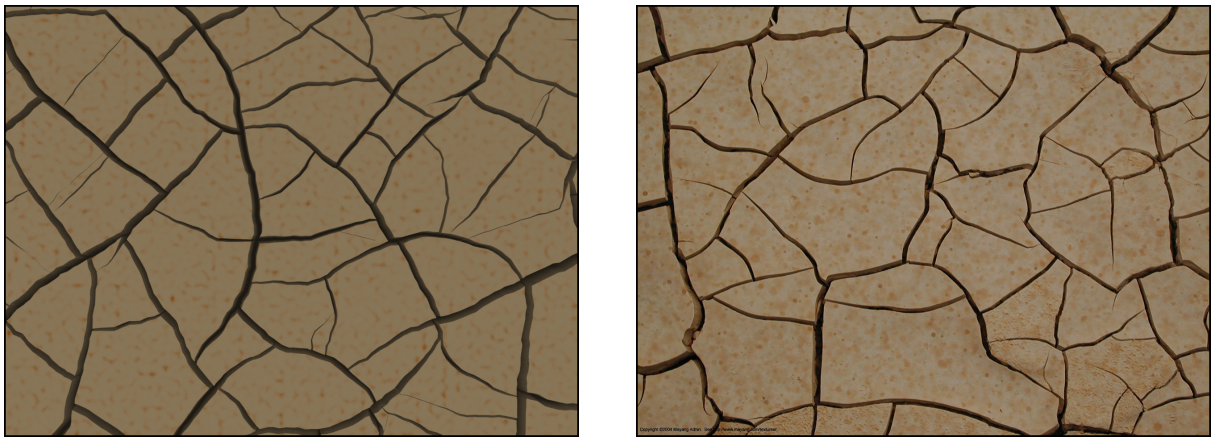

**Figure 11:** *A rendered example of dried mud (left) compared to a photograph (right, copyright 2004 Mayang Adnin). We used uniform shrinkage of the surface and set* $\alpha = 0.85$ *to propagate the cracks further, requiring 2.7 minutes computation time on a 19,602 triangle input mesh.*

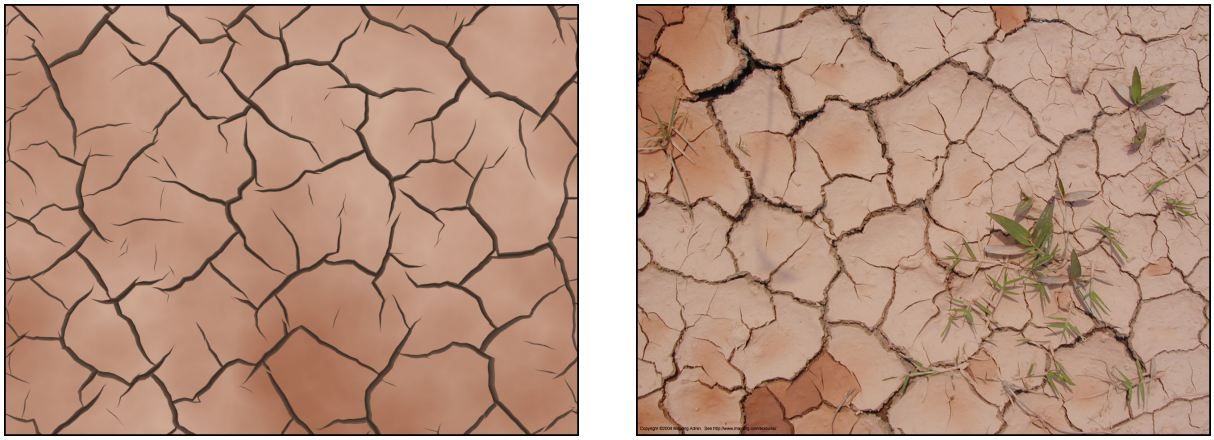

**Figure 12:** *A comparison between rendered dried mud (left) and a photograph (right, copyright 2004 Mayang Adnin). As in Figure 11, we used uniform shrinkage of the surface, but changed* $\alpha = 0.5$ *to demonstrate controlling the crack propagation parameter. Total computation time was 2.0 minutes on a 19,602 triangle input mesh.*


[O'B00] O'BRIEN J. F.: *Graphical modeling and animation of fracture*. PhD thesis, Georgia Institute of Technology, Aug 2000.

[OBH02] O'BRIEN J. F., BARGTEIL A., HODGINS J.: Graphical modeling and animation of ductile fracture. In *Proceedings of ACM SIGGRAPH 2002* (Aug 2002), pp. 291–294.

[OH99] O'BRIEN J. F., HODGINS J.: Graphical modeling and animation of brittle fracture. In *Proceedings of ACM SIGGRAPH 1999* (Aug 1999), pp. 137–146.

[PKA*05] PAULY M., KEISER R., ADAMS B., DUTRÉ P., GROSS M., GUIBAS L. J.: Meshless animation of fracturing solids. In *Proceedings of the ACM SIGGRAPH 2005* (July 2005), pp. 957–964.

[PPD02] PAQUETTE E., POULIN P., DRETTAKIS G.: The simulation of paint cracking and peeling. In *Proceedings of Graphics Interface* (May 2002), pp. 59–68.

[SdBGM00] SHORLIN K., DE BRUYN J., GRAHAM M., MORRIS S.: Development and geometry of isotropic and directional shrinkage-crack patterns. *Physical Review E 61*, 6 (June 2000).

[SM88] SKJELTORP A. T., MEAKIN P.: Fracture in microsphere monolayers studied by experiment and computer simulation. *Nature 335* (Sept. 1988), 424–426.

[TF88] TERZOPOULOS D., FLEISCHER K.: Modeling inelastic deformation: Viscoelasticity, plasticity, fracture. In *Computer Graphics (SIGGRAPH '88 Proceedings)* (Aug 1988), vol. 22, pp. 269–278.

[TPBF87] TERZOPOULOS D., PLATT J., BARR A., FLEISCHER K.: Elastically deformable models. In *Computer Graphics (SIGGRAPH '87 Proceedings)* (July 1987), vol. 21, pp. 205–214.

[WvOC04] WYVILL B., VAN OVERVELD K., CARPENDALE S.: Rendering cracks in batik. In *Proc. of the 3rd Int. Symp. on Non-photorealistic Animation and Rendering* (2004), pp. 61–69.