Graphical Models 71 (2009) 198–208

Contents lists available at ScienceDirect

Graphical Models

journal homepage: www.elsevier.com/locate/gmod

Generating surface crack patterns

Hayley N. Iben [a,b,*], James F. O'Brien [b]

[a] Pixar Animation Studios, Emeryville, CA, United States
[b] University of California, Berkeley, CA, United States

ARTICLE INFO

Article history:
Received 25 February 2007
Received in revised form 5 December 2008
Accepted 30 December 2008
Available online 14 January 2009

Keywords:
Crack patterns
Physically based modeling
Simulation
Finite elements
Fracture

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.

1. Introduction

Surface crack patterns occur on a variety of materials, including glass, mud, and ceramic glaze. These cracks often occur due to shrinkage of the object's surface area. For example, mud in a river bed dries faster on the surface than the underlying soil, causing stress to build. The mud subsequently 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 for modeling the crack patterns created by these types of processes, as demonstrated by the example of "crackle glass" shown in Fig. 1.

Several papers on physically based animation have presented methods to model fracture in solid objects. These fracturing methods use a second-order dynamic simulation, accelerating 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 quasi-static system will not be able to capture the dynamic effects that may occur in real materials when the motion of the crack front is driven by the momentum of the separating surfaces. This limitation is acceptable for the slow-forming surface cracks that we are concerned with here.

Our approach avoids certain problems for simulations that are caused by moving nodes, such as collisions or mesh tangling issues. Instead, we optionally move nodes as a post-processing step to create crack width. Although we may encounter similar tangling issues during this step, they only occur during post-processing and thus would not affect simulation behavior or stability.

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 and aspect ratio of the triangles do 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.

* Corresponding author. Address: Pixar Animation Studios, Emeryville, CA, 94608, United States.
E-mail addresses: iben@pixar.com (H.N. Iben), job@eecs.berkeley.edu (J.F. O'Brien).

1524-0703/$ - see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.gmod.2008.12.005

Fig. 1. A crackle glass dragon, generated by uniform shrinkage of the surface. This example required 2.17 h 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 [27]. 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.

In this paper, we present an extended version of our work that was originally presented in [14,15]. 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 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 loosely grouped into two categories. One area of research follows a non-physical approach for creating crack patterns, such as mapping procedural 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 to the graphics community methods for simulating elastic [34] and inelastic deformation of objects, including fracture [33]. A mass-spring system was later used to simulate fracturing solids [24]. Finite elements have been widely used to simulate brittle fracture [27], ductile fracture [26], elasto-plastic materials and interactive fracture [21], and the fracture and deformation of voxelized surface meshes [22]. Other algorithms include generating fracture on elastic and plastic materials with the virtual node algorithm [19], a membrane-bending model for thin shell objects [9], and a meshless framework [29].

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 [32], and to model tree bark [6], cracks in surfaces [12], and in volumes [13]. Gobron and Chiba used cellular automata to crack multi-layer surfaces [10] and simulate materials peeling off of surfaces [11]. Paint cracking and peeling was also simulated using a two-layered model on a 2D grid [28]. Federl and Prusinkiewicz used wedge-shaped finite elements to model cracks formed by drying mud and tree bark [7,8].

There have been several non-physical approaches to generating surface cracks. One type of approach maps some form of procedural crack pattern to an object's surface [16] and then carves out a volume to generate crack depth [18,5]. Others form cracks on a 2D surface to replicate Batik painting cracks [35] or create cracks similar to an input image [20]. These methods use an input pattern or image to generate surface cracks. In contrast, our method creates cracks from a mechanical stress field defined over the object's surface.

Similar to our overall approach, Valette and colleagues [36] use both physical and non-physical processes to generate cracks. First they precompute a 2D crack network using either a stochastic model, a Voronoi tessellation, or a watershed transform accounting for material thickness. Using layers of cubic cells to model the volume decrease in the material, they compute the resulting crack widths for the given network and then apply it to a parameterized 3D surface. They further discuss their crack model for soil desiccation in [37]. Our method uses a different combination of physical and non-physical components to create crack patterns. We use heuristics to initialize the stress field and a relaxation process to compute the resulting crack pattern. We also run our simulation directly on the 3D triangle mesh of the model.

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 [1] and [23]. Physics researchers have studied specific crack pattern problems, such as drying mud [17], alumina/water slurry [31], or both mud and ceramics [2,3].

3. Stress, forces and the separation tensor

As in previous approaches [27,26,8], we use a finite element discretization with local remeshing. However, our method differs from previous ones in several aspects. First, our algorithm generates cracks on the surface of objects in-

stead of fracturing their volume. Because our interest concerns only simulating the surface features, we use a triangle mesh discretization of the model where other methods use 3D elements. The reduced dimension of our elements simplifies the simulation computations. Second, 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 costs of the initial stress computation. Lastly, instead of moving the nodes during the simulation, such as in [27,13,8,19], our method keeps the nodes stationary and updates the stress field with a first-order quasi-static system.

3.1. The algorithm

To generate the crack patterns, we define a stress field over the triangles and evolve this field through time using a crack generation process. 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 evolving stress field. After cracks form, we update the stress field and repeat the process until either additional cracks cannot be created or the user terminates the simulation.

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 the nodes in a priority queue based on this value.
(3) While failure can occur and the user wishes 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.
  (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. 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 a uniform pull in all directions. For example, see Figs. 1 and 5. Alternatively, we can use the curvature tensor to initialize stress, indicating that the high curvature areas are more prone to cracks, as illustrated in Figs. 2 and 4. The user could also manually specify areas of high tension on the surface or use a pattern, such as in Fig. 3. We can also introduce some amount of random noise into the stress field to model material inhomogeneities. See Section 4.2 for more details.

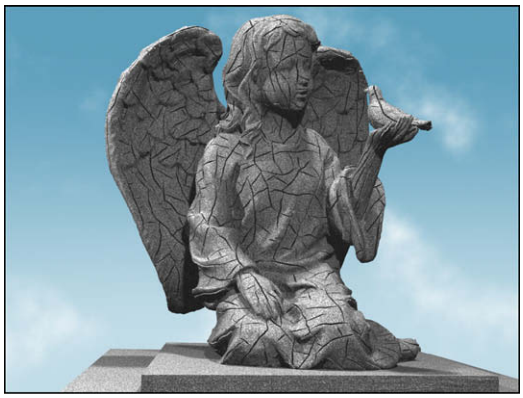

Fig. 2. As an artistic effect, we initialized the stress field to uniform tension with some bias to crack in the principal curvature directions. The result of using this heuristic is demonstrated by the vertical cracks of the angel's arm and the cracks following the folds of fabric. Total computation time was 3.13 h with an input mesh of 105,772 triangles.

3.3. Forces

The stress tensor of an element encapsulates the amount of force a small piece of material exerts on a node. Let $v_{[i]}$ denote v at node i and v_i denote element i of v. The force acting on node i by an element is

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

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

The derivation of the force equation and the barycentric basis matrix can be found in [27]. To summarize, the barycentric basis matrix will convert a point from local triangle coordinates to its barycentric coordinates. For a triangle, this basis matrix is defined by

$$\boldsymbol{\beta} = \begin{bmatrix} \mathbf{m}_{[1]} & \mathbf{m}_{[2]} & \mathbf{m}_{[3]} \\ 1 & 1 & 1 \end{bmatrix}^{-1}, \tag{2}$$

where $\mathbf{m}_{[i]}$ for $i \in \{1,2,3\}$ are the node positions in the triangle's local 2D coordinate system. To compute the triangle coordinate system, we use the vectors $\boldsymbol{p}_{[2]} - \boldsymbol{p}_{[1]}$ and $\boldsymbol{p}_{[3]} - \boldsymbol{p}_{[1]}$ to fix an orthonormal basis.

The primary difference between our equations and those presented in [27] 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 $\boldsymbol{\beta}$ matrix is 3×3 instead of 4×4. 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, Eq. (1) produces a force vector in the common 3D world coordinate system where the mesh is embedded.

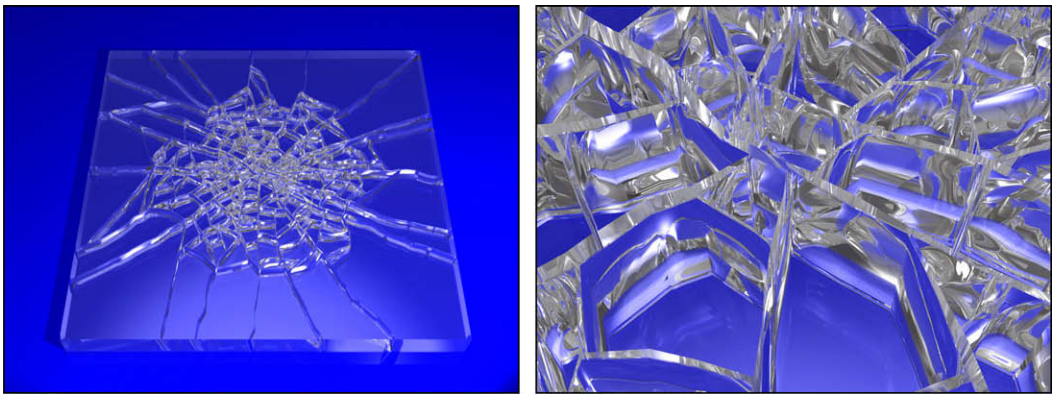

Fig. 3. Cracks generated from initializing the stress field with a pattern modeling impacts in flat glass. The field is evolved by a relaxation process, causing the cracks to propagate. The *right* image shows a close-up of the center of the impact region, demonstrating the gaps created by our post-processing step to widen the cracks and add depth. Total computation time was 15 s with an input mesh of 4804 triangles.

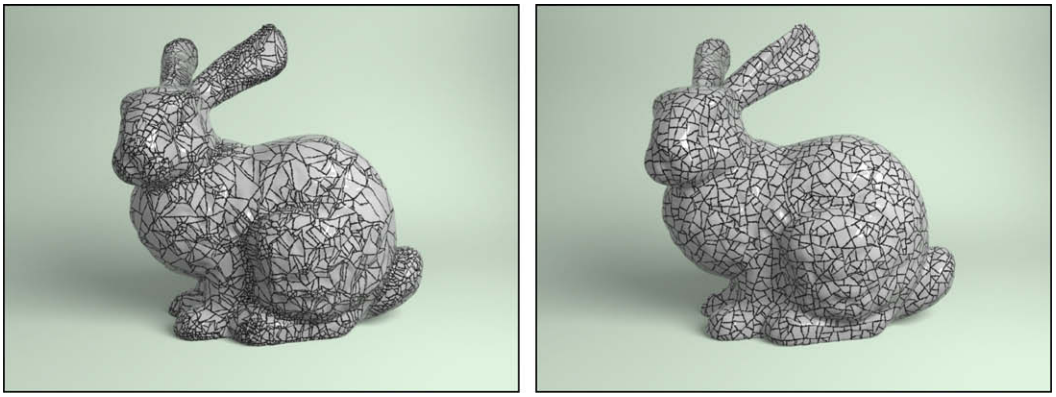

Fig. 4. Curvature can be used to control crack formation, as demonstrated by the image on the *left* requiring 2.55 h computation. Notice the high concentration of cracks around areas of high curvature, such as the ears, neck and leg. The *right* image was simulated using only uniform shrinkage of the surface, resulting in more evenly spaced cracks and requiring 2.0 h. The input mesh size was 51,382 triangles for both examples.

3.4. The separation tensor

We analyze the stress field to determine where to generate cracks by using the formulation of the separation tensor ς that was originally presented in [27]. 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

$$\varsigma = \frac{1}{2}\left(-\mathbf{m}(\boldsymbol{f}^{+}) + \sum_{\boldsymbol{f}\in\{\boldsymbol{f}^{+}\}} \mathbf{m}(\boldsymbol{f}) + \mathbf{m}(\boldsymbol{f}^{-}) - \sum_{\boldsymbol{f}\in\{\boldsymbol{f}^{-}\}} \mathbf{m}(\boldsymbol{f})\right), \tag{3}$$

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

We use this tensor to determine whether a crack occurs at a node and to determine the resulting orientation for the crack surface. We take the eigendecomposition of ς to find the largest positive eigenvalue, ν^{+}. The material fails at a node when ν^{+} is greater than the material toughness, τ. A crack occurs at this node in the crack plane defined by $\hat{\mathbf{n}}^{+}$, the eigenvector corresponding to ν^{+}. We discuss this process further in Section 4.3.

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 toward 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 evolu-

tion 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 [10], to reduce the tensile stress around cracks based on distance to the nearest crack [28], or to recalculate the equilibrium state adaptively during crack formation [8]. 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 Eq. (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 $\mathbf{g}_{[n]}$ 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

$$\Delta \boldsymbol{p}_{[n]} = \Delta t \mathbf{g}_{[n]}, \tag{4}$$

where Δt is the time step controlling the rate of relaxation. This quasi-static update corresponds to Aristotelian dynamics where units for force, displacement, and time do not balance as they would for a Newtonian formulation.

Rather than actually moving the node by $\Delta \boldsymbol{p}$, we will instead directly compute how this virtual displacement would have altered stress in the surrounding elements. We assume that stress in an element is isotropic and linearly proportional to its strain with proportionality constant 1. We use Green's strain tensor, ϵ, to relate displacement to strain within an element. This tensor is represented by a 3×3 matrix defined by

$$\epsilon_{ij} = \frac{1}{2}\left(\frac{\partial \boldsymbol{x}}{\partial u_i} \cdot \frac{\partial \boldsymbol{x}}{\partial u_j} - \delta_{ij}\right), \tag{5}$$

where δ_{ij} is the Kroneker delta:

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

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

$$\frac{\partial x_i}{\partial u_j} = \sum_{k=1}^{3} p_{[k]i} \beta_{kj}. \tag{7}$$

We need the change of the strain tensor within the element, so we compute its derivative with respect to the node positions by

$$\frac{\partial \epsilon_{ij}}{\partial p_{[n]r}} = \frac{1}{2}\left(\sum_{m=1}^{3} p_{[m]r}\beta_{ni}\beta_{mj} + \sum_{m=1}^{3} p_{[m]r}\beta_{mi}\beta_{nj}\right). \tag{8}$$

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

$$\Delta \boldsymbol{\sigma} = \frac{\partial \boldsymbol{\epsilon}}{\partial \boldsymbol{p}_{[n]}} \Delta \boldsymbol{p}_{[n]}. \tag{9}$$

At every time step, we use the force on the node to determine its virtual displacement from Eq. (4). We update the element's stress tensor by accumulating $\Delta \boldsymbol{\sigma}$ 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 4.3, 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 free open boundaries so that relaxation will relieve stress components perpendicular to the crack edges. An example of a simulation with a fixed boundary is Fig. 5 where the glaze meets the base of the cup. Notice that the majority of the cracks along this boundary are perpendicular to it, as they are in the photograph of a real teacup.

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.

4.2. Other stress field updates

In addition to relaxation, there are other ways to update the stress field. We model uniform shrinkage of the object by adding $c\boldsymbol{\delta}$, where c is a positive constant factor and $\boldsymbol{\delta}$ the identity matrix, to the stress tensor of each element. This amounts to generating uniform tension in the mesh, as illustrated by Figs. 5 and 6.

We also model anisotropic materials and impact patterns by specifying directionally varying stress patterns (Sections 4.2.1 and 4.2.2, respectively). As artistic effects, we use curvature (Section 4.2.3) or an input pattern (Section 4.2.4) to update the stress field. Additionally, we want to avoid stress patterns that are identically uniform as they can 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 field explicitly.

Fig. 5. 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 31 min with a 39,611 triangle input mesh.

Fig. 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 4.27 h computation with an input mesh of 60,000 triangles.

4.2.1. Modeling anisotropic materials

We can also model anisotropic materials, such as wood, by heuristically specifying the stress field. In terms of internal forces, an anisotropic material may have forces acting in one orientation only. To create directional crack patterns, we add directional forces to the stress field by projecting the desired direction to the local 2D coordinate system of each element, giving direction $\mathbf{v}_d$. We update the stress tensor by

$$\boldsymbol{\sigma}' = \boldsymbol{\sigma} + \mathbf{v}_d^T \mathbf{v}_d. \tag{10}$$

For a real-world example of cracks forming in one direction, we examined wood, a simple anisotropic material that is stronger along the grain than across it. We model wood cracking along the grain by using a directional stress field that is perpendicular to the grain. Fig. 7 shows an example of the resulting crack pattern along with a photograph for comparison.

We can also model the cracks forming in the cross-section of a sawn tree trunk. These types of wood cracks form perpendicular to the tree rings. We initialize the stress field to follow the rings, forming a circular stress field around the center of the tree. To calculate the direction to add to the stress field, we use Eq. (13) derived in the next section. We show an example of the cracks created from a circular stress field and a photograph for comparison in Fig. 8.

4.2.2. Modeling impact patterns

We can generate impact patterns on a surface by initializing the stress field appropriately. However, the stress field generated by an impact varies based on the material. For flat glass, we model a low-velocity impact fracture, the type of impact that produces surface crack patterns instead of pulverizing the glass. When the pane of glass is held in place, radial cracks initially form outward from the impact point, followed by concentric cracks [30].

We model this crack interaction by initializing the stress field with low stress radiating outward from the impact point and high stress in the concentric direction, as demonstrated by Fig. 3. After picking an impact point, $\boldsymbol{p}$, we determine the area where the stress pattern is defined from a user-specified radius, r. We use a falloff function to weight the stress amount at each triangle center $\mathbf{c}_{[i]}$ so that the stress is initially concentrated near the impact point and decreases to zero at the edge of the impact radius. After experimentation, we found the following weighting function to give us the desired falloff:

$$w_i = 2.0 \cos\left(\frac{d_i}{r}\right), \tag{11}$$

where d_i is the Euclidean distance from the i-th triangle center to the impact point. We determine the radial direction by first computing the vector

$$\mathbf{v}_{rad} = w_i \left(\frac{\mathbf{c}_{[i]} - \boldsymbol{p}}{|\, \mathbf{c}_{[i]} - \boldsymbol{p} \,|} \right) \tag{12}$$

and converting it to the triangle's coordinate system so that it can be added directly to the stress tensor, giving $\mathbf{v}'_{rad}$. The concentric direction is the perpendicular vector given by

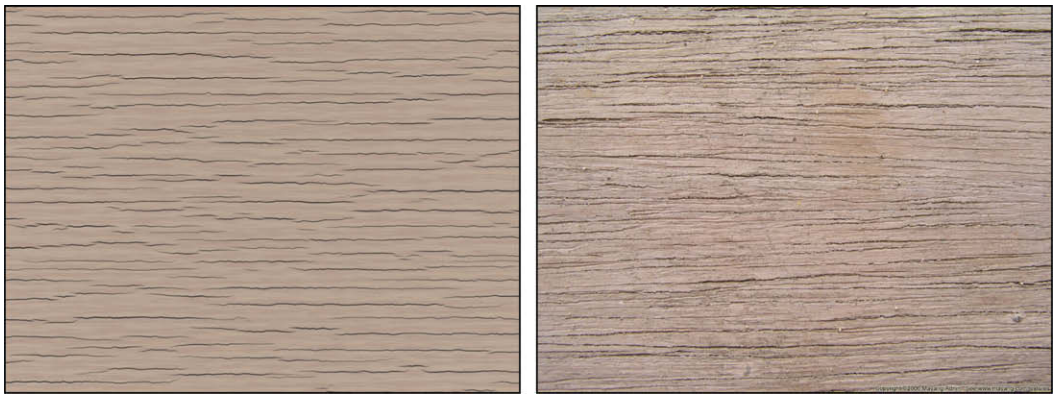

Fig. 7. An example of cracked wood along the grain (*left*) compared to a photograph (*right*, copyright 2006 Mayang Adnin). We used a vertical stress field to model the anisotropic material. Total computation time was 19.2 min on a 79,202 element mesh.

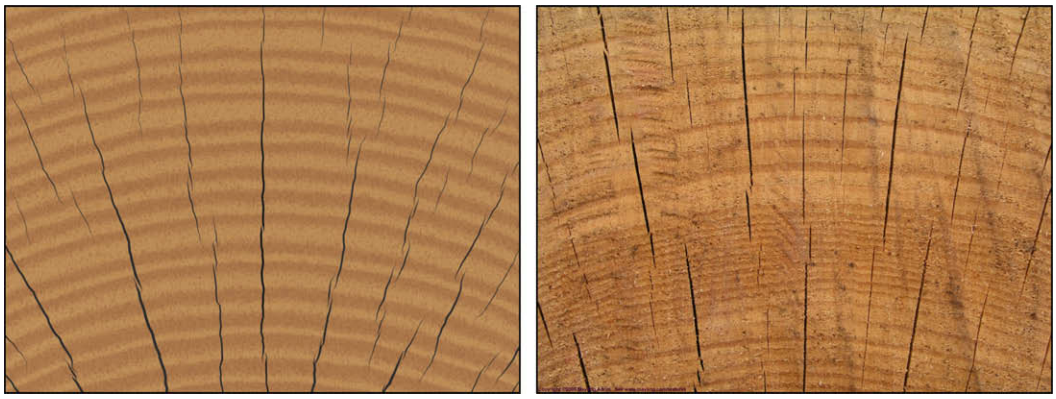

Fig. 8. We show an example of wood cracking perpendicular to the grain at the sawn end of a tree trunk (*left*) compared to a photograph (*right*, copyright 2005 Mayang Adnin). We used a concentric stress field, radiating outward from the center of the tree point, to model the anisotropic stress. Total computation time was 10.1 min on a 79,202 element mesh.

$$\mathbf{v}_{circ} = 1.25\left(\mathbf{v}'_{rad}\right)^{\perp}. \tag{13}$$

Recall that cracks form perpendicular to the largest force direction acting on a node. To ensure that the radial cracks form before concentric cracks, we use the scaling factor 1.25 in the above equation, giving larger forces in the concentric stress field than the radiating stress. After computing these two directions, we add them each to the stress tensor using Eq. (10).

4.2.3. Using curvature

Another way to update the stress field is with the object's curvature. We estimate the principal curvature using the discrete operator given in [4]. 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 as illustrated by Fig. 2. 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. Directly modifying the separation tensor controls the distribution of cracks on the surface, as demonstrated in Figs. 4 and 15.

4.2.4. Defining arbitrary stress patterns

Our method also allows us to arbitrarily set the stress field. For example, we used texture mapping to define the concentration of stress. Given a gray-scale image, we use the color assigned to a triangle's center to determine the amount of stress on that element. It is arbitrary how this weight is defined, but we decided to have the weight range from $[0, 1]$ by directly using the intensity of the colors from white to black. For our example, we multiply a uniform shrinkage stress tensor by this weight, giving the highest stress where there is black in the input image and no stress where there is white. See Fig. 9 for an example.

4.3. Generating cracks

As described in Section 3.4, we use the separation tensor ς to determine where to crack the mesh. We insert the largest positive eigenvalue, v^+, of each node's ς into 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, τ, a crack occurs at the corresponding node. We generate cracks in the direc-

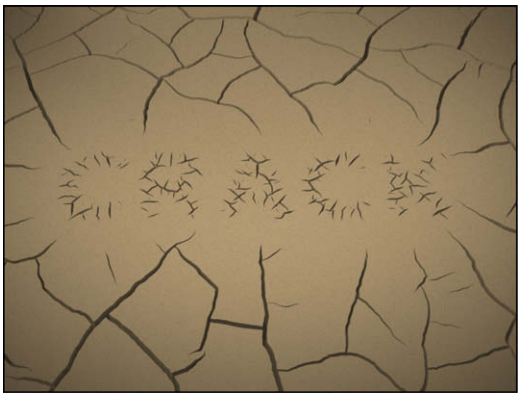

Fig. 9. We can use an input image, such as the letters "crack", to define areas with high stress and produce the small scale cracks shown in the center. We then created the large scale cracks with a second simulation by decreasing the stress in the letter region and reinitializing the surrounding area to uniform shrinkage.

tion perpendicular to 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 [27] and illustrated in Fig. 10. 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 [25] to propagate the crack further. After cracking a node, we compute a residual value $\nu^* = \nu^+ - \tau$. We then modify the separation tensor of nodes at the crack tip by

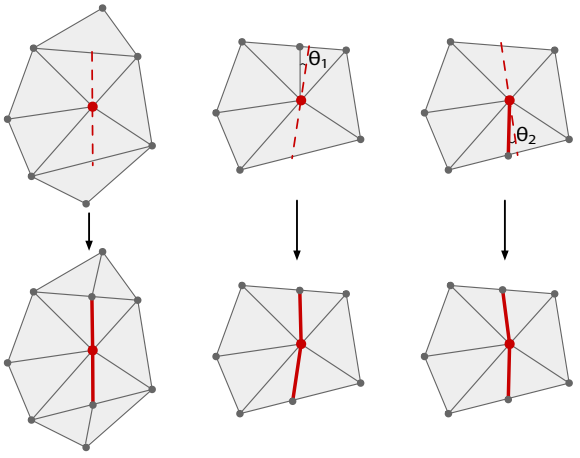

Fig. 10. The dotted line (crack plane) gives the location of a new crack. Bold edges are crack edges. *Left:* After crack edges are added, we locally remesh to ensure all elements are triangles. *Middle:* Cracks are snapped to an existing edge if θ_1 is less than a set threshold (ex: 15°). *Right:* Cracks are also snapped to an existing crack edge when θ_2 is less than a set threshold (ex: 25°), avoiding back-cracking.

$$\varsigma' = \varsigma + \alpha \mathbf{m}(\hat{\mathbf{n}}^+)\nu^*, \tag{14}$$

where α is an input parameter in the range $[0, 1]$. Small values for α result in more jagged cracks while large values cause the cracks to propagate further in the same direction. The parameter α heuristically captures material inhomogeneities, effectively controlling the jaggedness of the cracks in the simulation, as illustrated in Figs. 11 and 12. For our trials, we used α values near 0.5 to generate jagged cracks and in the range $[0.8, 0.9]$ to generate longer cracks. The value of the α parameter can not be greater than one; otherwise, the eigenvalues of the separation tensor will continually increase at each step, quickly causing the simulation to become unstable.

Even after updating the separation tensor using Eq. (14), we found that we were unable to generate certain types of long cracks. For example, cracks that initially form in ceramic glaze are often long and wrap around the object. To create this type of crack, we add a small constant factor to the residual, ν^*. We found that the value 0.25 produces the desired result, as demonstrated in our ceramic glaze examples (Figs. 5 and 6).

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 [27] and illustrated in Fig. 10. However, some slivers are large enough that edge snapping would create objectionable artifacts, yet still small enough that they would generate poorly conditioned elements.

We can trace the numerical instabilities in the simulation back to very large singular values in the $\boldsymbol{\beta}$ matrix. We initially compute a threshold based on the average singular value from the input triangles' $\boldsymbol{\beta}$ matrices. Each time an element is split and we calculate $\boldsymbol{\beta}$ matrices for the new triangles, we compute its SVD and examine the singular values. If any singular values exceed our conditioning threshold, we replace them with the threshold value and reconstruct the $\boldsymbol{\beta}$ matrix. 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 some examples, such as ceramic glaze, displaying the results does not require moving the node positions because the crack widths are small. Instead, we can render the results directly, as described below. However, for larger crack formations, we perform a post-processing step to compute crack widths. We can also add other types of effects after the simulation, such as curling or force-propagating cracks until they hit boundaries.

Because we allow the user to choose when the simulation terminates, some cracks may not have finished forming. We provide a method that propagates these crack tips until they terminate at another crack or mesh boundary. Our simulation keeps track of crack tip vertices as it progresses. However, crack tips may also form when two

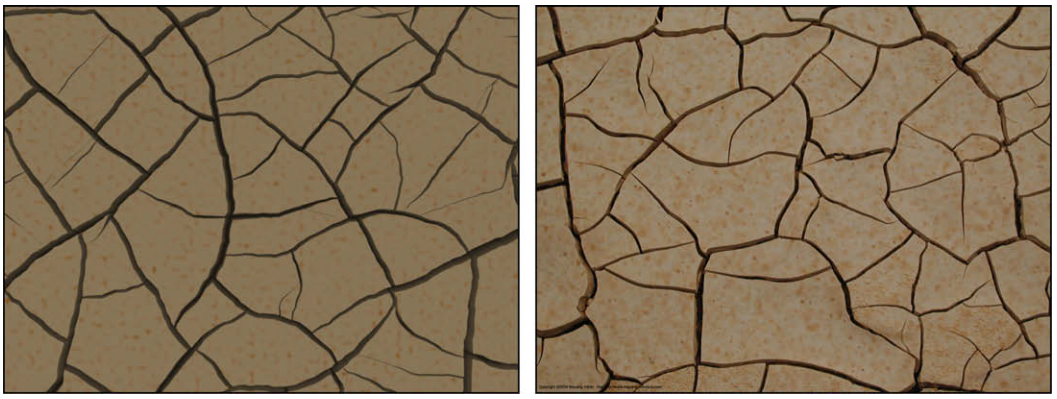

Fig. 11. A rendered example of dried mud (*left*) compared to a photograph (*right*, copyright 2004 Mayang Adnin). We used uniform shrinkage of the surface and set $\alpha = 0.85$ to propagate the cracks further, requiring 2.05 min computation time on a 19,602 triangle input mesh.

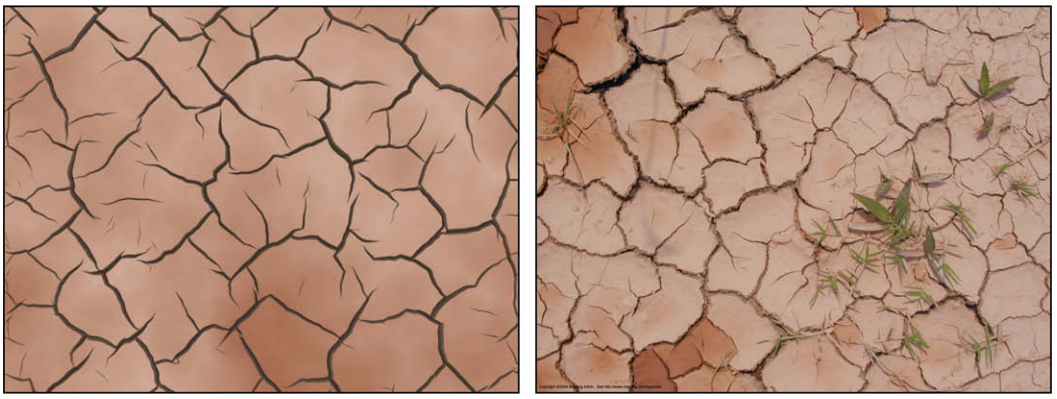

Fig. 12. A comparison between rendered dried mud (*left*) and a photograph (*right*, copyright 2004 Mayang Adnin). As in Fig. 11, we used uniform shrinkage of the surface, but changed $\alpha = 0.5$ to demonstrate controlling the crack propagation parameter. Total computation time was 1.5 min on a 19,602 triangle input mesh.

cracks intersect during growth. We detect this second case by searching for cracks with angles greater than 270° between two crack boundary edges. We treat the corresponding vertex as a crack tip.

After stopping the simulation, we finish cracks by setting the separation tensor of the crack tip's vertex to its residual tensor. This ensures that we continue cracking in the last known crack direction. We then crack the mesh as before, including the relaxation step because we need to compute the position change for generating gaps in the mesh. Finishing cracks effects the simulation results so it must be executed before other post-processing steps. This method is used in some of our examples, including Figs. 5, 11, and 15.

For the ceramic examples, we implemented a shader that interpolates information about crack locations to render dark lines. This rendering technique is demonstrated in Figs. 5 and 6. To achieve this effect, we treat each triangle separately and annotate its vertices with the distance to a crack edge in the triangle, if present. We split triangles with multiple crack edges into triangles with only one crack edge to ensure that this distance metric remains unique. Otherwise, the entire element would be treated as a crack and would be shaded the crack color.

For some of our examples, such as Figs. 3 and 11, we wanted to create a visible gap at the crack locations. To do so, we sum the change in positions from Eq. (4) for each node n as we compute them in the relaxation process:

$$\Delta_T \boldsymbol{p}_{[n]} = \sum_{i=1}^{R} \gamma \boldsymbol{g}_{[n]}, \tag{15}$$

where R is the total number of relaxation steps and γ 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 perpendicular to the surface by a user specified amount. This creates nice side-walls for the cracks, as illustrated in Figs. 11 and 12. Other examples using this method include Figs. 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 difficulties at this stage.

We can also add a curling effect to the crack edges, as found in some mud and peeling paint. To model this phenomena, we use the magnitude of the result from Eq.

Fig. 13. Example of adding a curling effect to the cracked mud from Fig. 12. This example required 1.5 min computation time on a 19,602 triangle input mesh.

(15), 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 Fig. 13.

6. Results and discussion

We implemented the method described above in C++ and rendered our results using the open source renderer, Pixie. The running time of the algorithm is dependent on the mesh resolution and concentration of cracks on the surface, with the largest amount of 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 2.5 GHz G5 computer with 2.5 GB of memory. These times are given in the figure 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 Figs. 5 and 6. We also provide a comparison photograph with a Japanese teacup in Fig. 5.

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 adheres 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 Fig. 1.

Flat glass also cracks in a particular manner when struck by an object, as described in Section 4.2.2. We allow the user to specify a stress field on the object to model this impact effect and demonstrate our results in Fig. 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 Figs. 11 and 12. The simulation generating these results differed in the parameter controlling crack propagation, α. 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 Fig. 14. We also generate a curled version of Fig. 12 in Fig. 13. These examples demonstrate the ease of generating different results by varying a few parameters.

We also model anisotropic materials by using a nonuniform stress field. For example, we model cracks forming along the wood grain by initializing the stress field to lie in the direction perpendicular to the grain. We demonstrate cracks created by this method in Fig. 7. We also create wood cracked along the rings of the sawn end of a tree by initializing the stress field along the rings. The resulting cracks are perpendicular to the rings, as illustrated in Fig. 8.

Our method can use the object's geometry to influence crack patterns. For example, the simulation creating Fig. 2

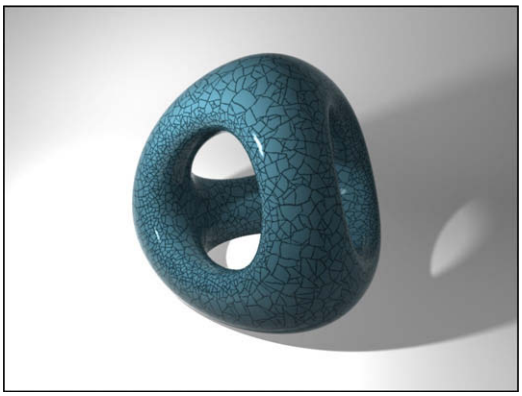

Fig. 15. Another example of an artistic effect achieved by using curvature to bias the concentration of cracks. In this example, the separation tensor was weighted by $\kappa_1^2 + \kappa_2^2$ where κ_1 and κ_2 are the principal curvature values. Computation time was 1.23 h on a 30,008 element mesh.

Fig. 14. An animation of mud drying for Fig. 11, simulated by setting the stress field to uniform shrinkage of the surface.

initialized the stress field to uniform shrinkage and then biased it in the principal 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 Fig. 4 and the sum of squares of the principal curvature values in Fig. 15, 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, GAANN, and Siebel fellowships.

Appendix A. Supplementary material

Supplementary data associated with this article can be found, in the online version, at doi:10.1016/j.gmod.2008.12.005.

References

[1] T.L. Anderson, Fracture Mechanics: Fundamentals and Applications, third ed., CRC Press, 2004.
[2] S. Bohn, L. Pauchard, Y. Couder, Hierarchical crack patterns as formed by successive domain divisions. I. Temporal and geometrical hierarchy, Physical Review E 71 (4) (2005) 046214.
[3] S. Bohn, J. Platkiewicz, B. Andreotti, M. Adda-Bedia, Y. Couder, Hierarchical crack patterns as formed by successive domain divisions. II. From disordered to deterministic behavior, Physical Review E 71 (4) (2005) 046215.
[4] D. Cohen-Steiner, J.-M. Morvan, Restricted delaunay triangulations and normal cycle, in: Proceedings of the 19th Annual ACM Symposium on Computational Geometry, 2003.
[5] B. Desbenoit, E. Galin, S. Akkouche, Modeling cracks and fractures, The Visual Computer 21 (8–10) (2005) 717–726.
[6] P. Federl, P. Prusinkiewicz, A texture model for cracked surfaces, with an application to tree bark, in: Proceedings of the 7th Western Computer Graphics Symposium, 1996.
[7] P. Federl, P. Prusinkiewicz, Modelling fracture formation in bi-layered materials, with applications to tree bark and drying mud, in: Proceedings of the 13th Western Computer Graphics Symposium, 2002.
[8] P. Federl, P. Prusinkiewicz, Finite element model of fracture formation on growing surfaces, Lecture Notes in Computer Science 3037 (2004) 138–145.
[9] Y. Gingold, A. Secord, J.Y. Han, E. Grinspun, D. Zorin, A discrete model for inelastic deformation of thin shells, in: ACM SIGGRAPH/Eurographics Symposium on Computer Animation, 2004, poster.
[10] S. Gobron, N. Chiba, Crack pattern simulation based on 3d surface cellular automaton, in: Proceedings of the International Conference on Computer Graphics, 2000.
[11] S. Gobron, N. Chiba, Simulation of peeling using 3d-surface cellular automata, in: Proceedings of the 9th Pacific Conference on Computer Graphics and Applications, 2001.
[12] K. Hirota, Y. Tanoue, T. Kaneko, Generation of crack patterns with a physical model, The Visual Computer 14 (3) (1998) 126–137.
[13] K. Hirota, Y. Tanoue, T. Kaneko, Simulation of three-dimensional cracks, The Visual Computer 16 (2000) 371–378.
[14] H.N. Iben, J.F. O'Brien, Generating surface crack patterns, in: Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation, 2006.
[15] H.N. Iben, Generating Surface Crack Patterns, Ph.D. Thesis, University of California, Berkeley, December 2007.
[16] H.-H. Hsieh, W.-K. Tai, C.-C. Chiang, M.-T. Yang, Flexible and interactive crack-like patterns presentation on 3d objects, Lecture Notes in Computer Science 3280 (2004) 90–99.
[17] S. Kitsunezaki, Fracture patterns induced by desiccation in a thin layer, Physical Review E 60 (6) (1999) 6449–6464.
[18] A. Martinet, E. Galin, B. Desbenoit, S. Akkouche, Procedural modeling of cracks and fractures, in: International Conference on Shape Modeling and Applications, 2004.
[19] N. Molino, Z. Bao, R. Fedkiw, A virtual node algorithm for changing mesh topology during simulation, ACM Transactions on Graphics 23 (3) (2004) 385–392.
[20] D. Mould, Image-guided fracture, in: Proceedings of Graphics Interface, 2005.
[21] M. Müller, M. Gross, Interactive virtual materials, in: Proceedings of Graphics Interface, 2004.
[22] M. Müller, M. Teschner, M. Gross, Physically-based simulation of objects represented by surface meshes, in: Proceedings of the Computer Graphics International, 2004.
[23] T. Nishioka, Computational dynamic fracture mechanics, International Journal of Fracture 86 (1997) 127–159.
[24] A. Norton, G. Turk, B. Bacon, J. Gerth, P. Sweeney, Animation of fracture by physical modeling, The Visual Computer 7 (4) (1991) 210–219.
[25] J.F. O'Brien, Graphical modeling and animation of fracture, Ph.D. Thesis, Georgia Institute of Technology, August 2000.
[26] J.F. O'Brien, A. Bargteil, J. Hodgins, Graphical modeling and animation of ductile fracture, in: Proceedings of ACM SIGGRAPH 2002, 2002.
[27] J.F. O'Brien, J. Hodgins, Graphical modeling and animation of brittle fracture, in: Proceedings of ACM SIGGRAPH 1999, 1999.
[28] E. Paquette, P. Poulin, G. Drettakis, The simulation of paint cracking and peeling, in: Proceedings of Graphics Interface, 2002.
[29] M. Pauly, R. Keiser, B. Adams, P. Dutré, M. Gross, L.J. Guibas, Meshless animation of fracturing solids, in: Proceedings of the ACM SIGGRAPH 2005, 2005.
[30] Scientific Working Group for Materials Analysis, Glass fractures, Forensic Science Communications 7 (1) (2005).
[31] K. Shorlin, J. de Bruyn, M. Graham, S. Morris, Development and geometry of isotropic and directional shrinkage-crack patterns, Physical Review E 61 (6) (2000) 6950–6957.
[32] A.T. Skjeltorp, P. Meakin, Fracture in microsphere monolayers studied by experiment and computer simulation, Nature 335 (1988) 424–426.
[33] D. Terzopoulos, K. Fleischer, Modeling inelastic deformation: Viscoelasticity, plasticity, fracture, in: Computer Graphics (SIGGRAPH'88 Proceedings), vol. 22, 1988.
[34] D. Terzopoulos, J. Platt, A. Barr, K. Fleischer, Elastically deformable models, in: Computer Graphics (SIGGRAPH'87 Proceedings), vol. 21, 1987.
[35] B. Wyvill, K. van Overveld, S. Carpendale, Rendering cracks in batik, in: Proceedings of the 3rd International Symposium on Non-photorealistic Animation and Rendering, 2004.
[36] G. Valette, S. Prévost, L. Lucas, A generalized cracks simulation on 3d-meshes, in: Proceedings of the Eurographics Workshop on Natural Phenomena, 2006.
[37] G. Valette, S. Prévost, L. Lucas, J. Léonard, A dynamic model of cracks development based on a 3d discrete shrinkage volume propagation, Computer Graphics 27 (1) (2008) 47–62.