Generating Surface Crack Patterns

Hayley Nicole Iben

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2007-142
http://www.eecs.berkeley.edu/Pubs/TechRpts/2007/EECS-2007-142.html

December 6, 2007

Generating Surface Crack Patterns

by

Hayley Nicole Iben

B.S. (Duquesne University) 2001
M.S. (University of California, Berkeley) 2005

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Computer Science

in the

GRADUATE DIVISION
of the
UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:
Professor James F. O'Brien, Chair
Professor Carlo H. Séquin
Professor Panayiotis Papadopoulos

Fall 2007

Generating Surface Crack Patterns

Hayley Nicole Iben

Abstract

Generating Surface Crack Patterns

by

Hayley Nicole Iben

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor James F. O'Brien, Chair

I present a method for graphically modeling a wide range of cracking phenomena produced naturally in mud, rock and wood, and also in man-made materials such as ceramic glaze and glass. In addition to creating visually plausible crack patterns, my approach affords the user control to tailor the appearance of the cracks. To satisfy these goals, I have developed a novel algorithm that obtains the realism of a physically correct simulation but maintains the controllability of a heuristic based method. With this combination, I can create a variety of crack patterns, ranging from regular patterns formed in rock to irregular patterns created in mud or ceramic glaze, by using the same system.

To model these phenomena, I build upon existing physically based methods but incorporate heuristics for controllability. My algorithm generates cracks from a stress field defined heuristically over a triangle discretization of the surface. The simulation then produces cracks by evolving this field over time. The user can control the characteristics and

appearance of the cracks through a set of simple parameters. By changing these parameters, I have generated examples similar to a variety of crack patterns found in the real world. I assess the realism of several results by comparison with photographs of real-world examples. Because a physically based approach is temporally coherent, I am also able to generate animations of surface cracking similar to time-lapse photography. To further demonstrate the flexibility of my approach, I have created various artistically driven examples.

Professor James F. O'Brien
Dissertation Committee Chair

To my family and close friends,

for your support during my years as a student.

Contents

List of Figures

List of Tables

Acknowledgments

I would like to thank my parents, sister, and grandmother for their unwavering love and support while I have been a graduate student far from home. Their constant encouragement and belief in my abilities has inspired me to always aim high and persevere when I encounter obstacles. I would also like to thank the many graduate students at Berkeley whom I befriended over the years. Their willingness to listen, give advice, and help me when I needed it most has meant much to me. I look forward to continuing these friendships past my years at Berkeley.

I am grateful for the guidance from my advisor, James O'Brien, who taught me much about computer graphics, writing papers, and giving presentations through both my masters and doctoral projects. Particularly, I am thankful for the hours of help when I started as his student, for his confidence in me and my work, and for his understanding and support during a personally difficult time in the middle of my graduate career.

I would also like thank my thesis committee and the members of the Berkeley Graphics Group, particularly Carlo Séquin and Jonathan Shewchuk, for their helpful criticism and comments. Their questions and insights have broadened the scope of this thesis. The graphics graduate students deserve a big thanks, especially those who gave me countless practice prelim exams. Without their help, I would not have gained the confidence in myself that I needed to undertake this degree.

Last but not least, I thank Pushkar Joshi who has provided much encouragement, support, and understanding while I have completed my degree at Berkeley and started my career.

Chapter 1

Introduction

In the real world, we are surrounded by materials with a variety of imperfections often caused by aging and weathering. For example, a piece of metal may have dents, cracks, stains and rust. Paint on a wall may crack and peel. Because of the numerous flaws in the objects we see everyday, the human eye is sensitive to their presence. We immediately identify images that appear "too perfect" as synthetic. In computer graphics, we strive to create visually plausible images for various applications including computer games and feature films. The methods used to generate these images are not necessarily physically correct but instead focus on making the images believable.

It is easy to create perfect synthetic objects with a computer, but it is difficult to trick the human eye into believing that a computer generated image is real. The image creators often do not want to paint all of the flaws onto a synthetic object by hand because of the time required to do so. Consequently, much research has emerged on modeling various imperfections, enabling us to more quickly create believable imperfect virtual worlds for

Figure 1.1: A crackle glass dragon simulated by my surface cracking algorithm. I generated the cracks by using uniform shrinkage of the surface.

computer games and feature films.

The number of imperfections is too broad to exhaustively model in one thesis. Instead, I focus on modeling one imperfection that is found in a variety of materials: cracks. Specifically, I strive to create crack patterns that occur in a thin layer on the surface of objects. I refer to these type of crack formation as *surface crack patterns*. My primary goal is to automatically generate visually plausible crack patterns without substantial user input about the desired result. Additionally, my method enables the user to optionally maintain some level of control over the results. With my approach, a user can create many kinds of crack patterns by controlling the characteristics and appearance of the cracks with a set of simple parameters.

Surface crack patterns occur on a variety of materials, including glass, mud, rock,

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. In this thesis, I describe a novel approach to model the crack patterns created by these types of processes and others, as demonstrated in Figure 1.1. I give more detailed examples of the types of surface cracks I intend to create in Section 1.2.

In computer graphics, methods for creating surface crack patterns can be grouped into two primary categories: non-physical approaches, such as mapping crack patterns to a surface, and physically based approaches. My method primarily fits 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, I instead use heuristic methods to define a stress field directly. The resulting approach generates realistic crack patterns while still affording the user a substantial amount of control.

Several physically based animation papers present methods to fracture objects. Terzopoulos and colleagues introduced simulating elastic [61] and inelastic deformation of objects, including fracture [60], to computer graphics. Subsequently, a variety of methods have been used to simulation fracture, including a mass-spring system [43], finite element approaches [46, 45, 40, 41], a virtual node algorithm [38] combined with finite elements [5], a meshless framework [50], and a hybrid approach combining finite elements and meshless methods [71].

Physically based methods have also been used to create surface cracks by applying many of the methods that break or tear apart objects. The methods used to generate cracks include mass-spring systems [58, 16, 26, 27], cellular automata [22], and wedge-shaped finite elements [17, 18]. Other methods create peeling effects in addition to cracks by using cellular automata [23] and a two-layered model on a 2D grid [48]. Several of these methods that simulate cracks represent the surface of the material with some thickness, such as a layer of tetrahedra, wedges, or a mass-spring network. However, my method differs in that I generate cracks on the surface of objects instead of cracking a volume. I use a triangle mesh discretization of the model rather than 3D elements, reducing the dimension of the calculations in my simulation.

Many of the fracture methods mentioned above (e.g. [46, 27, 18, 38]), use a second-order dynamic simulation, moving the nodes according to calculated forces. My method does not simulate dynamic forces or elastic waves, so the nodes do not move or have velocity. Instead, I use a first-order quasi-static system, a method that is inherently more stable than a second-order system. My choice also avoids certain problems caused by moving nodes, such as collisions or mesh tangling issues.

Some approaches use a full finite element simulation to calculate the stress values [46, 45, 18]. Instead, the method described here initializes the stress field directly with heuristics, such as uniform shrinkage of the surface area. This difference not only reduces the stress computation costs, but also gives the user control over the generated crack pattern.

A variety of non-physical approaches also create crack patterns on surfaces. These methods include mapping some form of a crack pattern to an object's surface [29, 36, 15],

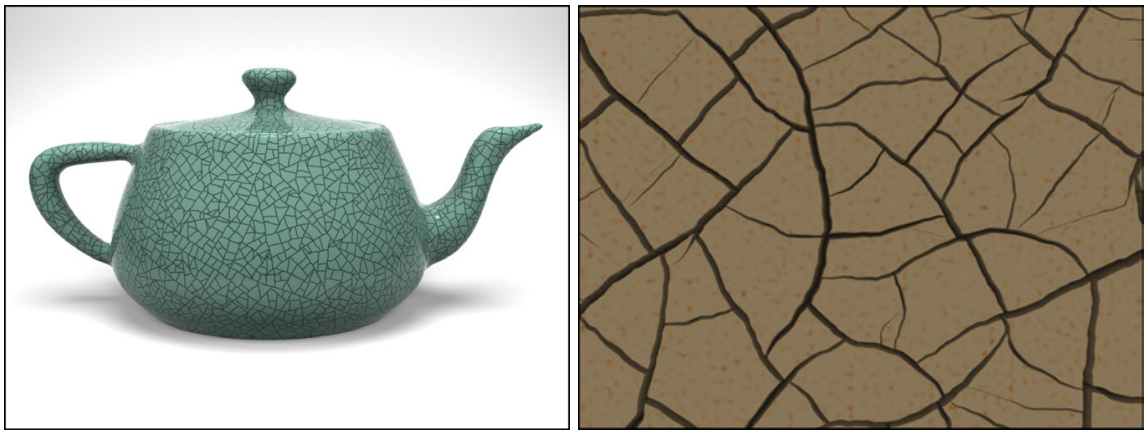


Figure 1.2: This figures shows a crackle glaze teapot on the *left* and desiccated mud on the *right*, both generated by my algorithm.

using geometric properties to create patterns [35, 70], or use techniques similar to applying textures [39, 34]. These methods are reliant on the user input, using an input pattern or image to create surface cracks. In contrast, my method does not require input about the resulting crack patterns; instead, I generate cracks from a stress field defined heuristically on the object's surface. The heuristics and the evolution of the stress field from the physical simulation are what drives the resulting crack patterns.

1.1 My Approach

To model the surface cracking phenomena mentioned above, I use existing physically based simulation methods as a basis. However, instead of calculating the stress on the surface of a model by using a numerical method (e.g. finite element method), I directly define my stress field using heuristics over a triangle discretization of the surface. I chose to use heuristics over a more physically accurate method for several reasons, including lower stress computation costs, controllability over the simulation results, and the ability

to generate a variety of crack patterns.

To generate the crack patterns, I use a triangle discretization of the input surface. This triangulation can be arbitrary, although my 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. I 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.

The algorithm I use to determine where cracks occur and how they propagate is derived from the method originally presented in [46]. 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.

My method is a novel combination, obtaining the realism of a physically correct simulation but maintaining the controllability of a heuristic based method. I 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. See Figure 1.2 for examples of crackle glaze and desiccated mud generated with my simulation.

1.2 Relevant Phenomena

I examine surface crack patterns that are produced naturally in mud and rock, but also found in man-made products such as the cracks found in ceramic glaze and glass. These crack patterns are often created to release stress within the material. The stress can occur from tensile forces generated by drying, cooling, or thermal shock as well as other mechanisms. Through understanding the process creating these types of cracks, I have identified the underlying themes tying them together. Using this information, I have developed a method enabling me to generate a wide variety of patterns as demonstrated by my results.

1.2.1 Ceramic Glaze

In ceramics, interesting crack patterns often result from failures occurring in the ceramic glaze. When a ceramic body and its glaze are the same dimensions at the firing temperature, the glaze begins solidifying as both components cool until they are rigidly joined. However, a glaze with a different coefficient of thermal expansion than that of the underlying pottery accumulates stress during the cooling process because the glaze and pottery contract at different rates. When the magnitude of the stress is large, the glaze on the ware either pulls apart, called "crazing," or cracks and then lifts, called "peeling" [59].

The glaze defect that occurs depends on the type of stress within the glaze. When the glaze contracts at a higher rate than the body, tensile stress builds in the glaze. Even a small increase in the glaze's contraction rate in this case causes failure in the glaze because glass withstands tensile forces less well than compressive forces. The glaze failure

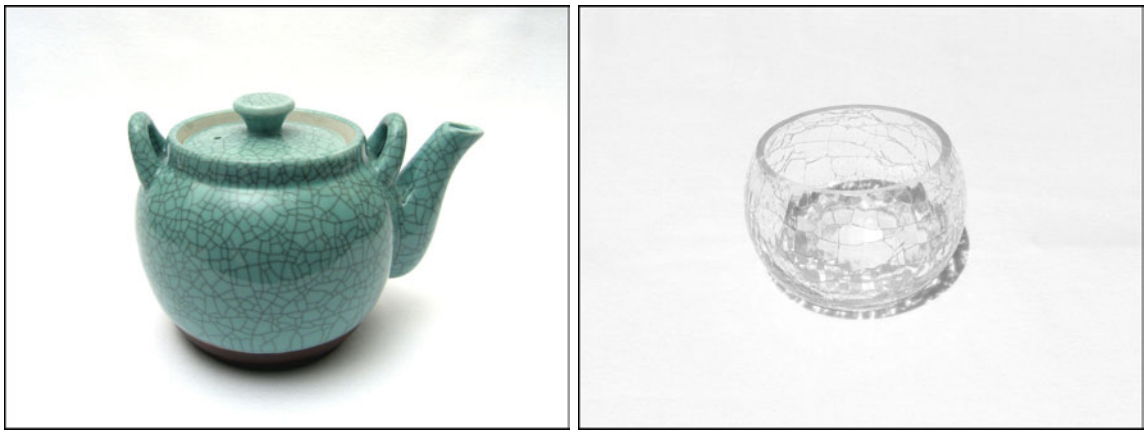


Figure 1.3: A Japanese teapot with crackle glaze is shown in the photograph on the *left*. The photograph on the *right* gives an example of crackle glass.

materializes as "crazing," a network of fine cracks forming within the glaze [59].

Crazing primarily occurs while ceramics cool in the kiln, but moisture expansion in the pottery can increase the tensile force in the glaze and cause secondary crazing to form. The resulting concentration of the crack pattern in the glaze occurs for different reasons, including cooling rates in the kiln (quicker cooling produces closer cracks) and glaze thickness. As an artistic effect, some glazes are designed to deliberately craze. These glazes are called "crackle glaze," shown in the photograph of a teapot in Figure 1.3, *left* [59]. Ceramicists make these small-scale cracks more prominent by rubbing ink into the surface; otherwise, the cracks would be difficult to discern [53].

In the opposite situation, the underlying pottery contracts at a higher rate than the glaze causing compression stress to build within the glaze. When the stress is greater than the glaze's adhesive strength, the glaze cracks and peels to alleviate it. The glaze flakes away from the interface or places that the crack edges overlap nearby glaze. However, peeling is not as prevalent as crazing in glaze defects because the glaze can withstand

higher compression forces [59]. For my research, I focus on producing ceramic crazing results because it is the more common defect. I give an example of crackle glaze produced by my method in Figure 1.2, *left*. (See Chapter 4 for an example of crackle glaze with a photographic comparison.)

1.2.2 Glass

Crack patterns also occur in glass for various reasons. For example, glass makers create glasswork with tiny networks of cracks in the surface called "crackle glass." I give an example photograph of crackle glass in Figure 1.3 (*right*). Other names for the glass treatment include craquelé (or craquelle) and ice glass [30].

To create crackle glass, glass blowers rapidly cool the outer layer of molten glass by submerge the piece in water, causing it to solidify without a chance for annealing. The resulting thermal shock causes the outer layer to finely crack. Because this layer is adhered to the inner core of molten glass, the object retains its overall shape and strength. The glass can then be reheated and reworked to achieve different crackle effects. For example, the glassblower can either seal the cracks or enlarge the gaps between cracks while maintaining a smooth interior [68]. I have developed a method to model the crack patterns created by this type of process, as demonstrated by Figure 1.1, an example of crackle glass generated from my system.

1.2.3 Mud

The structure of porous soil changes due to the amount of water it contains. When there is more water, the soil tends to swell. As the water evaporates, the soil shrinks and

Figure 1.4: The photograph on the *left* (copyright Timo Arnall) shows mud from Valle de la Luna, Chile that has cracked due to desiccation. On the *right* I give a joint pattern sometimes referred to as "tessellated pavement" in the photograph from Bouddi National Park, Australia (copyright Peter Adderley). This particular pattern developed in sandstone in the Terrigal Formation.

can lead to cracking. For example, mud in a river bed dries faster on the surface than in the underlying soil, causing tensile stress to build due to the contraction of the surface. Vertical tension cracks develop in the mud to relieve this stress, often forming polygonal-shaped crack patterns where the cracks terminate orthogonally. A photograph of a desiccation crack pattern is shown in Figure 1.4, *left*.

The crack patterns resulting from drying depends upon the material's composition, bed thickness, drying rate [10] and the interaction with neighboring cracks [52], among other factors. These influences control the geometric characteristics of the cracks, including continuity, length, width between cracks, and jaggedness. Through my algorithm's flexibility, I can reproduce a variety of crack patterns found in dried mud (see Figure 1.2 for an example). In Chapter 5, I give two distinctly different mud crack patterns generated by my system along with photographic comparisons.

Figure 1.5: The photograph (copyright 2006 Ian Parker) shows the Checkerboard Mesa found in Zion National Park, Utah. The land formation is comprised of sandstone, a sedimentary rock created from layers of sand dunes that are compressed and cemented together over time. The colored horizontal bands, called crossbedding, represent the sand layers within the mesa.

1.2.4 Rock

Crack patterns, such as those found in Figure 1.4 (*right*) and Figure 1.5, also form in rock. They are created by high stresses from a variety of sources, including tectonics, lithostatic pressure, thermal effects such as cooling, fluid pressure, impact from extraterrestrial objects, and other geological activities including folding and volcanos [13]. Regardless of the stress source, rock fractures occur from brittle failure [10]. In this thesis, I focus on modeling fractures with no measurable shear displacement, called *joints* by geologists. Such fractures are primarily caused by tensile stress [6].

In rock, joints often form in a regular pattern that have prominent geometric

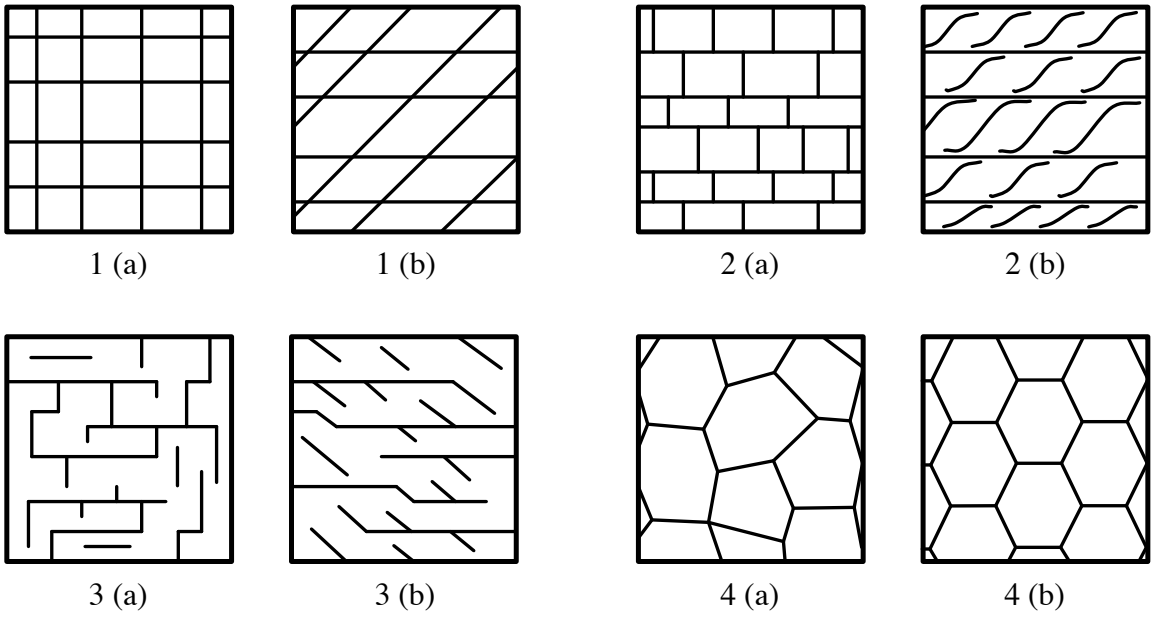


Figure 1.6: Geologists use the intersection geometry of multiple joint sets based on orthogonality and continuity as one method for classifying joint patterns. *1:* Both continuous, (a) orthogonal or (b) nonorthogonal. *2:* One continuous and the other discontinuous, (a) orthogonal or (b) nonorthogonal. *3:* Both discontinuous, (a) orthogonal and (b) nonorthogonal. *4:* All discontinuous, triple intersection with (a) various angles or (b) 120° [52].

features. By idealizing the geometric intersection characteristics, as shown in Figure 1.6, geologists are able to classify joint patterns [52]. For example, Figure 1.4 (*right*) shows a crack pattern that falls into the classification depicted in Figure 1.6, 1 (a). These parallel crack patterns probably formed after the covering rock strata was removed, releasing pressure on the sandstone. Without the heavy load, the sandstone can expand upward, creating stress and cracking the surface. Further weathering and erosion separated the cracks, making them more prominent. I provide more details on these types of rock crack patterns and their formation in Chapter 7.

Another example of surface crack patterns in rock are found in the Entrada Sandstone area of Arches National Park, Utah. The crack patterns probably formed due to stress

from the strata arching, causing brittle fracture in the sandstone. Geologists believe that the the bending in the strata occurred during different time periods [33], creating interacting crack patterns similar to Figure 1.6, 2 (a). I am also able to produce crack patterns similar to these examples with my algorithm, as described in Section 4.2.2.

Chapter 2

Background

Many scientists have studied the material failures leading to crack production. For my purposes, I focus on the literature presented in the computer graphics community in Section 2.1, where my research results primarily fit. However, for completeness I briefly discuss the work in fields that computer graphics researchers reference, including fracture mechanics, physics and geology.

2.1 Computer Graphics

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 primarily falls 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,

I 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.

2.1.1 Non-physical Methods

There have been several non-physical approaches for generating cracks on an object's surface. These methods often involve mapping an input crack representation to a given surface. Hsieh and colleagues mapped a crack pattern image by projecting a planar graph representation of the pattern to a given surface [29]. Martinet and colleagues also map a crack pattern, provided as a graph, to a surface and either carve out the pattern from the volume or fracture the object [36]. A similar approach is presented in [15], where a fracture pattern consisting of curves is mapped to the object and the crack pattern is carved out of the surface.

Other non-physical methods use information about crack geometry to create crack patterns on a surface. To model uniform dried mud and earthenware, MacVeigh [35] used the observation that crack intersections occur at right angles and cracks form such as to minimize their length. Base on this criteria, his method creates a graph with lines and circular arcs to give the crack pattern. Wyvill and colleagues [70] examine crack properties found in Batik paintings and use the Distance Transform algorithm [25] to generate cracking results on a 2D image.

Crack patterns have also been applied by using texture mapping techniques. Mould [39] modeled surface cracks from an image by applying the region boundaries of a Voronoi diagram to the object, as in texture mapping. Lu and colleagues [34] captured the shape and texture from weathering and aging effects on objects, including cracks, and

determined the context parameters for the pattern development. Using this information and a transform algorithm, they applied the textures to a synthetic object.

Most of these methods use an existing pattern or image to generate the surface cracks, relying on the input for the crack configurations. However, my method generates crack patterns from a stress field defined heuristically on the surface. This stress field enables us to create cracks based on the surface features, giving more realistic results.

2.1.2 Physically Based Methods

Several researchers developed methods for modeling fracture in solid materials, primarily using physically based simulation. Terzopoulos and colleagues [61] introduced simulating the elastic deformation of objects through a finite difference method to computer graphics. This work was extended to perform inelastic deformations, including fracture[60]. Norton and colleagues [43] used a mass-spring system to simulate fracturing objects by breaking a spring when its strain exceeded a threshold.

O'Brien and colleagues used a finite element method on a tetrahedral mesh to simulate brittle [46] and ductile fracture [45]. Müller and Gross [40] used a finite element method with a multi-resolution strategy to interactively simulate elasticity, plasticity and fracture. Another finite element paper simulated fracture and deformation of voxelized surface meshes [41]. Pauly and colleagues generated fractures on elastic and plastic materials with a meshless framework [50]. A hybrid approach that combines finite elements using tetrahedron with meshless methods to generate debris along with fracture was presented in [71]. Other fracture algorithms include the virtual node algorithm [38]. This algorithm was applied in a method combining rigid body simulation with finite element analysis for

animating fracture [5].

Several papers in computer graphics have also modeled thin shell fracture using triangle meshes, such as an inelastic deformation model [21] and a finite element model using subdivision [11]. Meshless fracture approaches include a method using a conformal parameterization [24] and a point sample method using fibers [69].

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 [58]. Hirota and colleagues [26] used a two-layered mass-spring system to describe stress and strain on an object's surface. They later used a tetrahedral configuration to propagate cracks inside a 3D volume [27]. Aoki and colleagues also use a tetrahedral mass-spring configuration combined with a moisture model to simulate drying clay [3].

Federl and Prusinkiewicz [16] used a mass-spring system to model cracks in tree bark. They later used a two-layer wedged-shaped model to generate cracks in both tree bark and drying mud [17]. Cracks are formed by removing a wedge element when its stress exceeds a threshold. They extended this work in [18] where they split the elements with the stress fracture plane, as introduced in [46], instead of removing elements. After altering the mesh, they perform a local relaxation step to reach equilibrium.

Gobron and Chiba [22] introduced a method to crack multi-layer surfaces based on cellular automata and a stress spectrum. They extended this work to simulate the peeling of materials on surfaces [23]. Paquette and colleagues [48] simulated paint cracking and peeling using a two-layered model on a 2D grid with stress. They used relaxation to reduce

the tensile stress around the crack area.

Valette and colleagues [64] present a method with both physical and non-physical aspects. They first precompute a 2D crack network using either a stochastic model, a Voronoi tessellation, or the watershed transform. Using layers of cubic cells to model the volumetric decrease in the material, they compute the resulting crack widths for the network. Lastly, they they parameterize a 3D surface and map the 2D cracks to it, giving the final result.

Our algorithm is based on the work presented in [46], using a finite element method often found in physically based approaches. Because I am interested in modeling the surface of the object, I reduce the dimension of the problem by using a triangle discretization instead of tetrahedra for the simulation. I also use heuristic methods to define the stress field directly and elastic relaxation to evolve it, rather than actually simulating the elastic deformation of an object. The resulting approach generates realistic crack patterns while still allowing the user a substantial amount of control.

2.2 Engineering and Physics

Fracture mechanics has been extensively researched in mechanical engineering, providing a wide range of methods for analyzing material failure. Researchers use finite element methods, differencing methods, and boundary integral equations to simulate the failure of real materials. An overview of methods used in this field can be found in [2] and [42]. Other methods include the extended finite element method (e.g. [4]) and meshless methods to model thin shell fracture (e.g. [55]). Physics researchers have studied spe-

cific crack pattern problems, such as mud [31], both mud and ceramics [7, 8], or drying alumina/water slurry [57].

The goals of modeling cracks and fracture differ between engineering and computer graphics. In engineering and physics, real-world applications require accurate results from computer simulations. However, in computer graphics, our primary goal is to produce an image that is visually realistic, not necessarily physically correct. With this in mind, I chose to implement a physical simulation with simplifying assumptions to generate a variety of crack patterns instead of focusing on one particular application.

2.3 Geology

Geologists have studied the causes of fracture in rock and soil for over a century. Pollard and Aydin present the culmination of this research on the mechanics and geometric characteristics of joints in [52]. They conclude that joints are primarily opening mode fractures, joints initiate at material inhomogeneities, and the mechanical interaction between joints influence the final crack pattern. In addition to general joint characteristics, geologists have also studied specific geological structures, such as crack formation in soil and mud due to drying [65, 67], joints in sandstone at Arches National Park, Utah [14, 33] and the scaling relationships between limestone fractures found in The Burren, County Clare, Ireland [20].

Several papers in the geology field use computer simulations to create crack patterns, either by using a geometric approach or a physically based approach. The geometric approaches examine the crack geometry and use a set of rules to replicate the pattern char-

acteristics. To model drying soil, Horgan and Young [28] use a random walk along with three crack stages to generate cracks pattern on a 2D plane. Perrier and colleagues [51] model drying soil by using a fractal structure with multiple levels of fragmentation to generate the crack pattern.

The physically based approaches incorporate many of the methods discussed above with models from geology. Olson and Pollard [47] discuss a method that uses the shape of overlapping cracks to find the magnitude of the differential stress. Using this information, they numerically generate fracture patterns. Thomas [62] used the boundary element method to model faults, fractures and cavities in a program called Poly3D. Bourne and colleagues [9] use Poly3D with geomechanical models to calculate the stress field around faults so that they can then grow a fracture network for fluid simulation. To model drying clay, Vogel and colleagues [66] use a triangular mass-spring network where springs break when the strain between two nodes exceeds the material's threshold.

These methods focus on geological problems with emphasis on cracking in soil and rock. My method instead proposes a more general approach, generating cracks in soil and rock as these papers do, but also in ceramic glaze, glass, and wood. Additionally, my approach gives the user artistic control over the simulation through the use of heuristics to define the stress field and the various simulation parameters. This diversity enables the user to generate many crack patterns using one system.

Chapter 3

Stress, Forces and the Separation Tensor

As in previous approaches [46, 45, 18], I use a finite element discretization with some form of local remeshing after crack formation. However, my method differs in several aspects. First, my algorithm generates cracks on the surface of objects instead of fracturing their volume. Because my interest is only simulating the surface features, I use a triangle mesh discretization of the model where other methods use 3D elements. This reduced dimension of my elements simplifies the simulation computations. Second, I initialize my stress field directly with heuristics instead of using a full finite element simulation. This simplification gives the user optional control over the resulting crack patterns and decreases the initial stress computation costs. Lastly, instead of moving the nodes when fracture occurs, such as in [46, 27, 18, 38], my method keeps the nodes stationary and updates the stress field with a first-order quasi-static system.

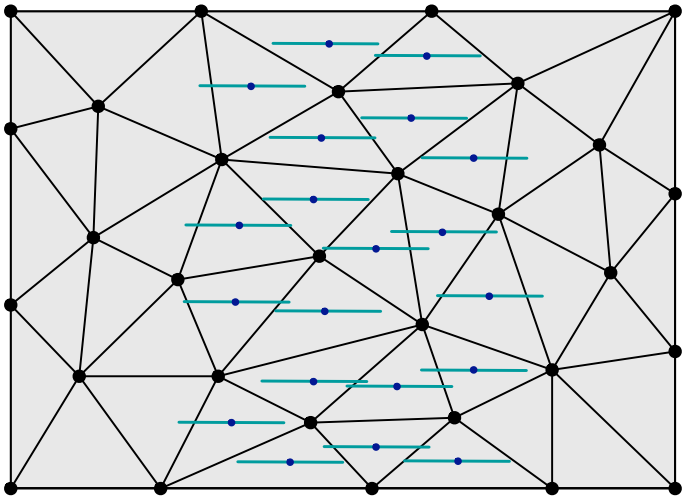


Figure 3.1: This figure shows a simple stress field created by horizontal uniform tension in a central strip of the mesh. The stress field is represented by plotting the two stress tensor's eigenvectors times their corresponding eigenvalues at the triangle centers. Because this stress is unidirectional, one eigenvalue is zero, giving only one eigenvector to display. I will use this example and method for displaying the stress field in later figures.

3.1 The Algorithm

To generate cracks, I define a stress field over the surface of the object using heuristics to model various stress patterns. I then evolve this field over time using crack generation processes. By analyzing the stress field in terms of forces, the algorithm forms cracks in areas of high stress on the mesh. These cracks are introduced as free boundaries that affect the relaxation process and stress field. After cracks form, I update the stress field and repeat the process until either additional cracks cannot be added or the user terminates the simulation.

The following steps describe my 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 and locally remesh as needed
   (b) Evolve the stress field:
      i. Perform relaxation by using the forces acting on each node to update the surrounding elements' stress tensors
      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 geometry
   (b) Directly render the crack edges

3.2 Stress Field

In three-dimensional continuum mechanics, the stress tensor reflects the force distribution per unit area within an element [19]. The stress tensor is completely defined at every point within the object by examining the stresses acting on three mutually orthogonal surface elements. To obtain the principal planes of stress, these three orthogonal planes are

chosen so that there are no shear stress components. The normals to the principal planes are the principal stress directions at that point. When one of the principal stress components reduces to zero, the three-dimensional stress state simplifies to two dimensions. The remaining two principal stresses lie in a plane (see Figure 3.2). For this simulation, I assume that the stress is zero in the direction normal to the surface. The resulting stress tensor is a 2D tensor lying in the plane of the triangle elements.

This stress tensor varies from point to point on the object and depends on the chosen orthogonal planes, defined by the frame for a coordinate system. These frames are not unique; I may choose a different set. However, I can convert a stress tensor defined in

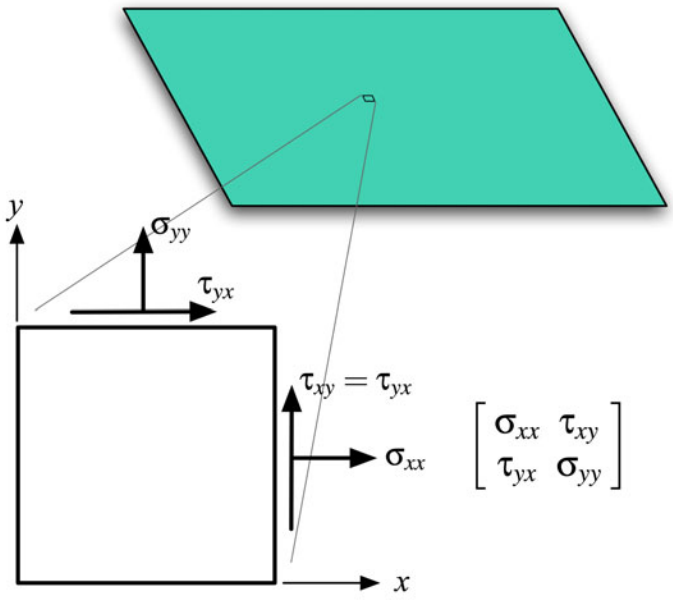


Figure 3.2: The stress tensor measures the internal force distribution per unit area resulting from external forces. In two dimensions, stress measures the internal force acting upon a small differential area on a plane as described by the normal (denoted σ) and shear (denoted τ) stresses.

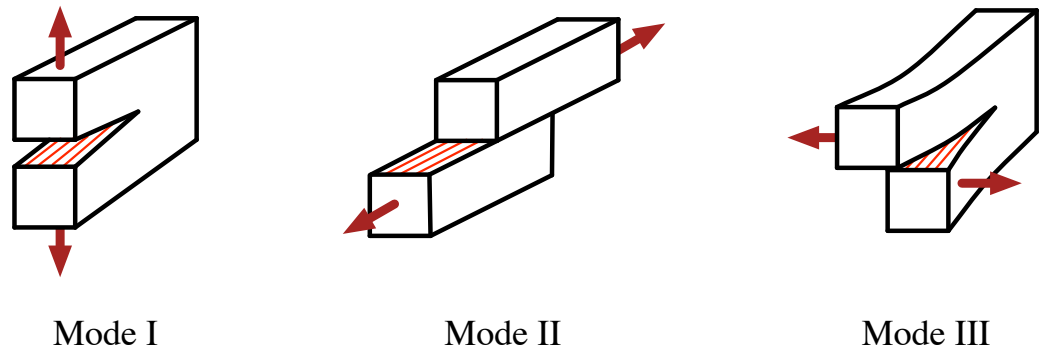


Figure 3.3: The three modes of fracture development are opening fracture (*Mode I*), sliding fracture (*Mode II*), and tearing fracture (*Mode III*).

one coordinate system to another by using the rotation between the two frames [6].

I define the stress field on the surface of an object by storing the 2D stress tensors at the triangle centers and treating the stress as a constant over the area of the triangle (see Figure 3.1). Because my elements are triangles, I store the stress tensor in a local 2D coordinate system defined by the triangle, resulting in a 2×2 tensor quantity (see Figure 3.2). I further explain the triangle coordinate system below in Section 3.3.

Fracture occurs when the stress acting within a material exceeds the cohesive strength at the atomic level. However, scientists have found that the actual strength of a material differs from the theoretical estimates by up to several orders of magnitude. The reason for this discrepancy is that flaws in the material lower its global strength by locally concentrating the stress [2]. By heuristically defining the stress field, I can also easily model the flaws within a material.

To understand the stress heuristics needed to form a surface crack, I turn to fracture mechanics and examine the three modes of fracture development (depicted in Figure 3.3). Mode I corresponds to fractures opening in the direction perpendicular to the

surface. Mode II refers to an in-plane shearing of the material through horizontal sliding. The last, Mode III, pertains to material shearing out of the plane, also referred to as tearing. A material may have one or more of these modes as it undergoes forces at the crack tip [2].

The surface crack patterns modeled in this thesis fall into the Mode I fracture category because they are all opening fractures. Mode I fractures are commonly due to tensile stress generated from processes such drying or cooling causing the material to shrink. As illustrated in Figure 3.3, the fracture forms perpendicular to the tensile forces pulling on the material.

When the stress field is initialized to zero, the surface is in equilibrium. To simulate drying and shrinkage, I initialize the stress tensor to uniform tension, indicating that all directions pull uniformly. However, materials are inhomogeneous, causing fluctuations in the stress field. For example, on the surface of mud, equal amounts of principal normal stress are originally generated when the surface contracts. However, the resulting cracks are arranged arbitrarily partly due to fluctuations in the tensile stress introduced by the material's inhomogeneities [10]. My method can model material inhomogeneities, including flaws, by modifying the stress field. For example, I can create variations in the crack patterns by introducing some amount of random noise into the stress field or the shrinkage amount used to compute the stress.

For artistic results, I can use the curvature tensor to initialize stress, indicating that the high curvature areas are more prone to cracks (e.g. Figures 4.11 and 4.13). The user could also manually specify areas of high tension on the surface (e.g. Figure 4.16) or use a pattern to define the stress (e.g. Figures 4.5 and 4.9). For more details on updating

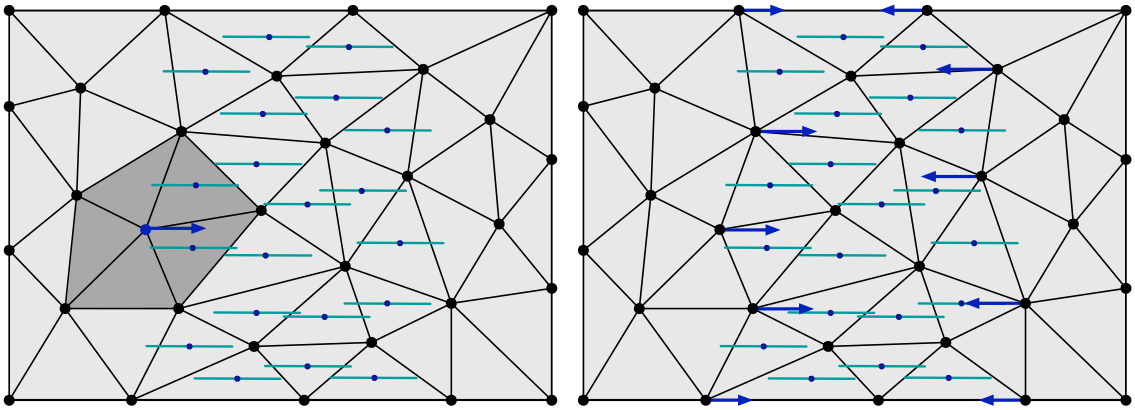


Figure 3.4: I compute the net force acting on a node by summing up the forces contributed by each element of its vertex ring (*left*). By repeating this process, I compute the net forces acting on all nodes of the mesh (*right*).

the stress field, see Section 4.2.

3.3 Forces

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

$$\boldsymbol{f}_{[i]} = -A \sum_{j=1}^{3} \boldsymbol{p}_{[j]} \sum_{k=1}^{2} \sum_{l=1}^{2} \beta_{jl} \beta_{ik} \sigma_{kl} \tag{3.1}$$

where A is the area of the element, β the barycentric basis matrix, $\boldsymbol{p}$ the node's position in world coordinates and $\boldsymbol{\sigma}$ the stress tensor. To calculate the total force for a given node, I sum the forces exerted on the node by the elements in its surrounding vertex ring. See Figure 3.4, *left*, for an example using the stress field described in Figure 3.1.

The derivation of Equation (3.1) and the barycentric basis matrix can be found in [46]. To summarize, they calculate the total force as the sum of the elastic forces and damping forces (absent in this case) acting on the node. The barycentric basis matrix

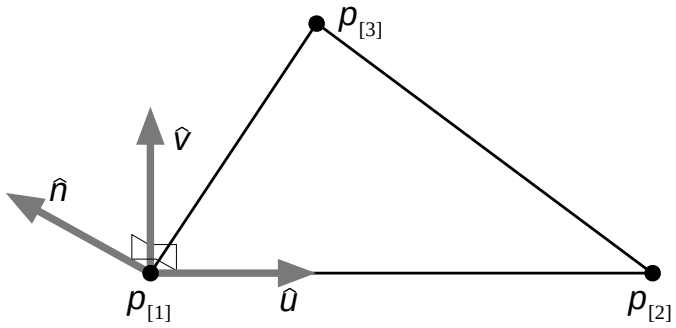


Figure 3.5: I compute a local 2D coordinate system for each triangle by fixing an orthonormal basis using the vectors $\boldsymbol{u} = \boldsymbol{p}_{[2]} - \boldsymbol{p}_{[1]}$ and $\boldsymbol{p}_{[3]} - \boldsymbol{p}_{[1]}$. Specifically, the local coordinate system is derived from $\boldsymbol{n} = \hat{\boldsymbol{u}} \times \big(\boldsymbol{p}_{[3]} - \boldsymbol{p}_{[1]}\big)$, the triangle plane's normal. (Note that $\hat{\boldsymbol{a}}$ denotes the normalization of $\boldsymbol{a}$.) The v-axis is then found by $\boldsymbol{v} = \hat{\boldsymbol{n}} \times \hat{\boldsymbol{u}}$. The resulting vectors $\{\hat{\boldsymbol{u}}, \hat{\boldsymbol{v}}\}$ define the 2D coordinate system.

converts a point from local triangle coordinates to its barycentric coordinates. For a triangle, the basis matrix is defined by

$$\beta = \begin{bmatrix} \mathbf{m}_{[1]} & \mathbf{m}_{[2]} & \mathbf{m}_{[3]} \\ 1 & 1 & 1 \end{bmatrix}^{-1} \tag{3.2}$$

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 as depicted in Figure 3.5, I use the vectors $\boldsymbol{p}_{[2]} - \boldsymbol{p}_{[1]}$ and $\boldsymbol{p}_{[3]} - \boldsymbol{p}_{[1]}$ to fix an orthonormal basis. Specifically, I arbitrarily set the u-axis to be $\hat{\boldsymbol{u}}$, the normalized vector of $\boldsymbol{u} = \boldsymbol{p}_{[2]} - \boldsymbol{p}_{[1]}$. The normal vector for the plane of the triangle is $\hat{\boldsymbol{n}}$ where $\boldsymbol{n} = \hat{\boldsymbol{u}} \times \big(\boldsymbol{p}_{[3]} - \boldsymbol{p}_{[1]}\big)$. I find the v-axis of my coordinate system by $\boldsymbol{v} = \hat{\boldsymbol{n}} \times \hat{\boldsymbol{u}}$. The two vectors $\{\hat{\boldsymbol{u}}, \hat{\boldsymbol{v}}\}$ define my local 2D triangle coordinate system.

The primary difference between my equations and those presented in [46] is due to my discretization choice. I use triangles instead of tetrahedra, so the stress tensor is a two dimensional quantity in the plane of its triangle and the β 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, the expression in Equation (3.1) produces a force vector in the common 3D coordinate system where the mesh is embedded. I calculate the force in 3D because each triangle in the node's surrounding vertex ring has its own local coordinate system. Defining a common coordinate system in 2D would involve a prohibitively large amount of calculation because of the varying dihedral angles between neighboring triangles. Instead, to combine the stress quantities of a node's vertex ring, they are converted to their 3D equivalent using the barycentric basis matrix, giving a common coordinate system for the force calculation.

I can also classify forces as tensile or compressive based on the distortion caused by the force. Compressive forces refer to those forces pushing particles of an object closer together, while tensile forces pull particles of an object farther apart [37]. The force I compute in Equation (3.1) gives the total force, the sum of tensile and compressive forces acting on an object. I will use tensile and compressive forces and describe their computation in the following section.

3.4 Separation Tensor

To determine where to generate cracks, I analyze the stress field by using the formulation of the separation tensor ς originally presented in [46]. They form the tensor by balancing the tensile and compressive forces exerted on a node. Because the forces are

calculated in 3D world coordinates, I can use them directly in the equation

$$\boldsymbol{\varsigma} = \frac{1}{2}\left(-\mathbf{m}(\boldsymbol{f}^{+}) + \sum_{\boldsymbol{f}\in\{\boldsymbol{f}^{+}\}} \mathbf{m}(\boldsymbol{f}) + \mathbf{m}(\boldsymbol{f}^{-}) - \sum_{\boldsymbol{f}\in\{\boldsymbol{f}^{-}\}} \mathbf{m}(\boldsymbol{f}) \right) \tag{3.3}$$

where $\boldsymbol{f}^{+}$ is the tensile force, $\boldsymbol{f}^{-}$ the compressive force, and $\mathbf{m}$ a function computing the outer product of a vector with itself divided by the vector's length.

To compute these forces, I decompose the stress tensor into tensile $\boldsymbol{\sigma}^{+}$ and compressive $\boldsymbol{\sigma}^{-}$ components by using principal stresses. (Note that I am using the solid mechanics notation, opposite that of soil and rock mechanics [6].) By taking the eigendecomposition of the stress tensor, I find the principal stresses (the eigenvalues of $\boldsymbol{\sigma}$), the maximum and minimum normal stresses acting upon the principal planes of stress. These planes are orthogonal planes where only normal stresses exist. Because the stress tensor is symmetric, it is always possible to find these planes and their normal stresses [19].

Let the eigenvalues of the stress tensor be $\mathsf{v}^{i}(\boldsymbol{\sigma})$ with $i \in \{1, 2, 3\}$ and let their corresponding normalized eigenvectors be $\hat{\mathbf{n}}^{i}(\boldsymbol{\sigma})$. Because my stress tensor is a 2×2 quantity, the eigendecomposition is relatively cheap. I compute the two components of the stress tensor by

$$\boldsymbol{\sigma}^{+} = \sum_{j=1}^{2} \max(0, \mathsf{v}^{i}(\boldsymbol{\sigma}))\ \mathbf{m}(\hat{\mathbf{n}}^{i}(\boldsymbol{\sigma})) \tag{3.4}$$

$$\boldsymbol{\sigma}^{-} = \sum_{j=1}^{2} \min(0, \mathsf{v}^{i}(\boldsymbol{\sigma}))\ \mathbf{m}(\hat{\mathbf{n}}^{i}(\boldsymbol{\sigma}))\ . \tag{3.5}$$

I then substitute $\boldsymbol{\sigma}^{+}$ and $\boldsymbol{\sigma}^{-}$ for $\boldsymbol{\sigma}$ in Equation (3.1) to compute $\boldsymbol{f}^{+}$ and $\boldsymbol{f}^{-}$, respectively, as described in [46].

I use the separation tensor to determine whether a crack occurs at a node and the resulting crack direction. I take the eigendecomposition of $\boldsymbol{\varsigma}$ to find the largest positive

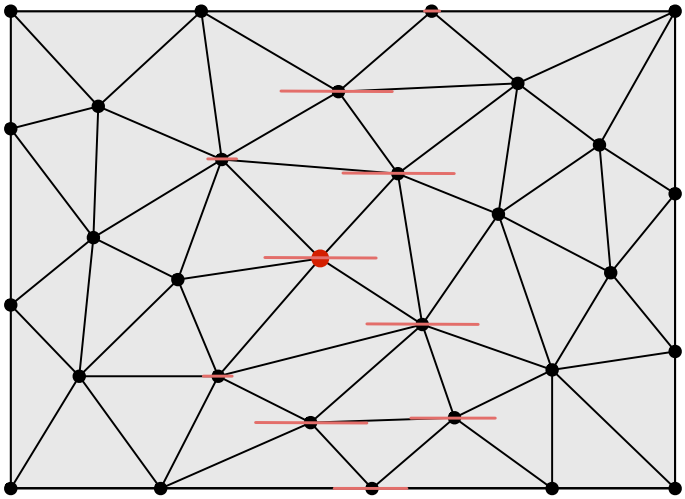


Figure 3.6: Using the forces depicted in Figure 3.4, this figure illustrates the corresponding separation tensor by plotting it's maximum eigenvector times the corresponding eigenvalue at each node. The node with the largest maximum eigenvalue is the larger node in the center and is the node tested for failure.

eigenvalue, v^+. The material fails at a node when v^+ is greater than the material toughness threshold, τ. A crack occurs at this node in the crack plane defined by $\hat{\mathbf{n}}^+$, the eigenvector corresponding to v^+ (see Figure 3.6). However, more than one eigenvalue may be greater than τ. In this situation, I use the largest eigenvalue to crack the mesh and continue the simulation. I discuss this process further in Chapter 5.

Chapter 4

Mesh Updates

After calculating an initial stress field, my method performs an iterative procedure to generate crack patterns. This process interleaves the evolution of the stress field and the 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, I instead treat stress directly as an inde-

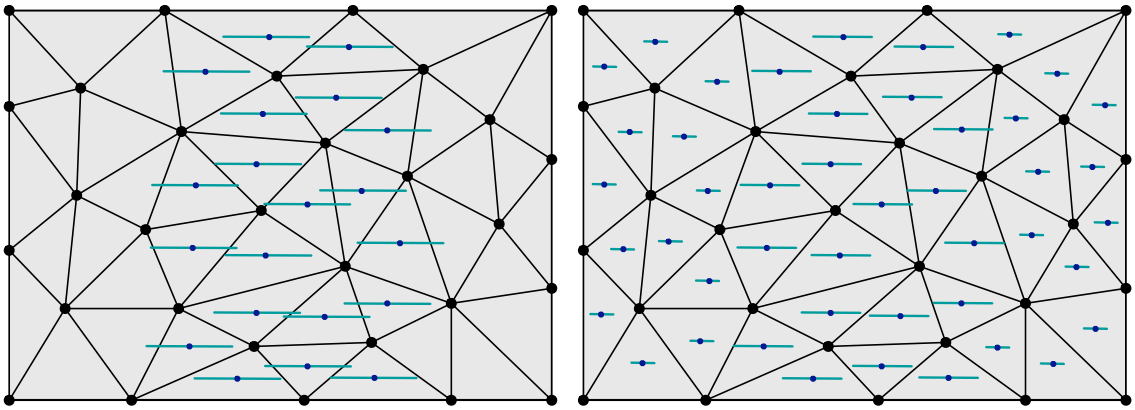


Figure 4.1: My relaxation method is a mesh-based diffusion process on tensor quantities. At each step, it redistributes stress from areas of high stress (*left*) to areas of low stress (*right*).

pendent 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, this stress relaxation is also a diffusion process, as described below.

Other cracking methods have used relaxation to smooth the stress field after initialization [22], to reduce the tensile stress around cracks based on the distance to the nearest crack [48], or to recalculate the equilibrium state adaptively during crack formation [18]. I use relaxation to redistribute stress from areas of high stress to areas of low stress (Figure 4.1). To perform the relaxation, I calculate the virtual displacement of the nodes to determine the change in the stress field. To determine this displacement, I first compute the forces exerted on nodes with Equation (3.1), encapsulating the effect of the stress field on the nodes. I then assume that if I were modeling the displacement of the nodes, they would be moved by these forces, giving the virtual displacement. However, by not actually

moving the nodes, I 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. I calculate the virtual displacement based on the total forces acting on node n by

$$\Delta \boldsymbol{p}_{[n]} = \Delta t \ \boldsymbol{F}_{[n]} \tag{4.1}$$

where Δt is the time step controlling the rate of relaxation.

I assume that stress in an element is linearly proportional to its strain with proportionality constant 1. I use Green's strain tensor, $\boldsymbol{\epsilon}$, to determine the amount of deformation of an element around the node [19]. This tensor is represented by a 3×3 symmetric matrix defined by

$$\epsilon_{ij} = \frac{1}{2}\left(\frac{\partial \boldsymbol{x}}{\partial u_i} \cdot \frac{\partial \boldsymbol{x}}{\partial u_j} - \delta_{ij}\right) \tag{4.2}$$

where δ_{ij} is the Kroneker delta:

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \quad . \end{cases} \tag{4.3}$$

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

$$\frac{\partial x_r}{\partial u_i} = \sum_{m=1}^{3} p_{[m]r}\beta_{mi} \quad . \tag{4.4}$$

Substituting this into Green's strain tensor equation gives

$$\epsilon_{ij} = \frac{1}{2}\left(\sum_{r=1}^{3}\sum_{m=1}^{3}\sum_{n=1}^{3} p_{[m]r}\beta_{mi}\, p_{[n]r}\beta_{nj} - \delta_{ij}\right) \quad . \tag{4.5}$$

I need the change of the strain tensor within the element, so I compute the derivative of Equation (4.5) with respect to the node positions

$$\frac{\partial \epsilon_{ij}}{\partial p_{[n]r}} = \frac{1}{2}\left(\sum_{m=1}^{3} p_{[m]r}\beta_{ni}\beta_{mj} + \sum_{m=1}^{3} p_{[m]r}\beta_{mi}\beta_{nj}\right) \quad . \tag{4.6}$$

Figure 4.2: A rendered example of a crackle glaze cup (*left*) compared to a photograph (*right*). My 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 minutes with a 39,611 triangle input mesh.

Because I assumed that the relationship between strain and stress is linearly proportional, I can compute the change of stress in an element with node n as

$$\Delta\boldsymbol{\sigma} = \frac{\partial\boldsymbol{\epsilon}}{\partial\boldsymbol{p}_{[n]}}\Delta\boldsymbol{p}_{[n]} \quad . \tag{4.7}$$

At every time step, I use the force on the node to determine its virtual displacement from Equation (4.1). I update the element's stress tensor by accumulating $\Delta\boldsymbol{\sigma}$ into it. After changing the stress, I 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 Chapter 5, is invoked.

I can treat boundary edges in my mesh as either free or fixed. For fixed boundaries, I 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. An example of a simulation

with a fixed boundary is Figure 4.2 where the glaze meets the base of the cup. Notice that the majority of the cracks along the boundary are perpendicular to it, as they are in the photograph of the 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, I could likewise accelerate my algorithm by using a more sophisticated integration scheme. However, I have so far found my 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. These methods including modeling uniform shrinkage of surface area (Section 4.2.1), patterns specified by a direction (Section 4.2.2), and impact patterns (Section 4.2.3). As artistic effects, I can also use curvature (Section 4.2.4) or an input pattern (Section 4.2.5) to update the stress field. Additionally, I want to avoid stress patterns that are identically uniform as they create unrealistic artifacts in the resulting crack pattern (Section 4.3). However, the discretization error in an unstructured mesh adds some inherent randomness to the stress field, avoiding this problem. I can also add randomness to the tensor field explicitly (Section 4.2.6).

Figure 4.3: 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 hours computation with an input mesh of 60,000 triangles.

4.2.1 Uniform Shrinkage of the Surface Area

I model uniform shrinkage of the object by updating the stress tensor with

$$\boldsymbol{\sigma}' = \boldsymbol{\sigma} + c\,\mathbf{I} \tag{4.8}$$

where c is a positive constant factor. This addition amounts to generating uniform tension in the mesh, as illustrated by the crackle glaze examples in Figures 4.2 and 4.3. Figure 4.18 (*right*) gives an example of a uniform stress field pointing in all directions. Other results using uniform shrinkage of the surface area include crackle glass (Figure 1.1) and mud (Figures 5.4 and 5.5).

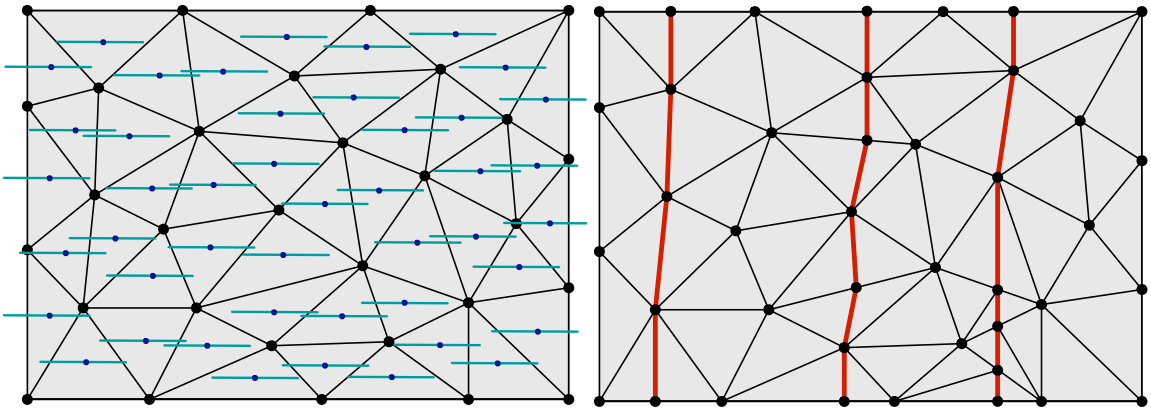


Figure 4.4: I define a stress field that models forces pulling horizontally on the *left*. The resulting cracks are vertical, as shown on the *right*.

4.2.2 Stress in a Particular Direction

By definition, anisotropic materials have properties that are directionally dependent. In terms of internal forces, an anisotropic material may have forces pulling in one direction only. When these forces exceed the material's toughness threshold, I obtain crack patterns in the direction perpendicular to the forces. To model this type of crack pattern, I add some form of the pulled direction to my stress field. However, I first need to convert the direction into a quantity that I can use.

Tensile forces represent the maximum positive eigenvalue and eigenvector pair of the stress tensor (see Equation (3.4)). By setting the eigenvector to be equal to the desired direction, I ensure that the tensile forces pull in that direction as well. However, I first convert the given direction to the local 2D coordinate system of the triangle, obtaining $\boldsymbol{d}_p$. I then create a matrix that has its eigenvector in the $\boldsymbol{d}_p$ direction by using the outer product of the vector with itself. Because $\boldsymbol{d}_p$ is in the same coordinate system as the stress tensor,

Figure 4.5: An example of cracked wood along the grain (*left*) compared to a photograph (*right*, copyright 2006 Mayang Adnin). I used a vertical stress field to model the anisotropic material. Total computation time was 19.2 minutes on a 79,202 element mesh.

I simply add the two quantities to obtain the updated stress tensor

$$\boldsymbol{\sigma}' = \boldsymbol{\sigma} + \boldsymbol{d}_p^T \boldsymbol{d}_p \quad . \tag{4.9}$$

For example, I can model forces only pulling in the horizontal direction by setting the stress field as in Figure 4.4, *left*, giving a vertical crack pattern.

For a real-world example of cracks forming in one direction, I examine wood, a simple anisotropic material that is stronger along the grain than across it. I model wood cracking along the grain by using a directional stress field that is perpendicular to the grain. Figure 4.5 shows an example of the resulting crack pattern along with a photograph for comparison.

I can also model the cracks forming in the cross-section of a sawn tree. These types of wood cracks form perpendicular to the tree rings. So, I initialize my 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, I use Equation (4.12) derived in the next section.

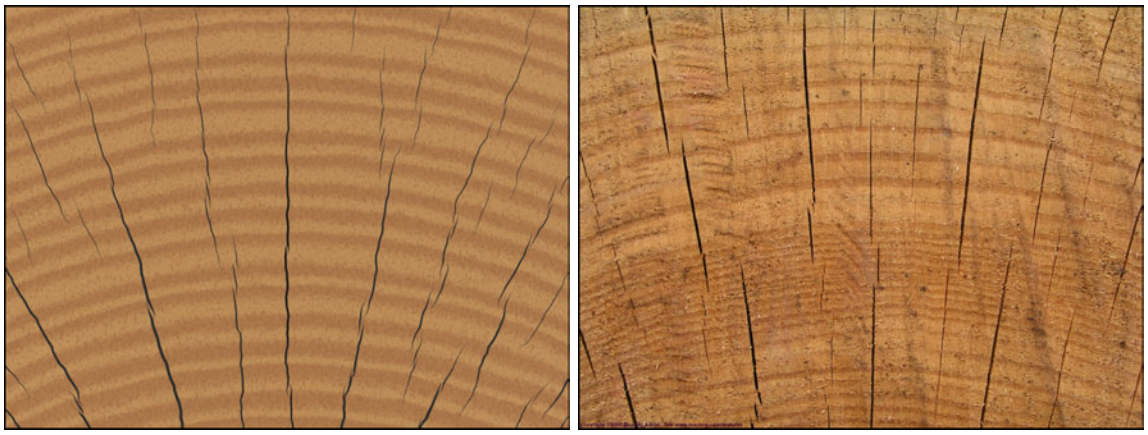


Figure 4.6: I show an example of wood cracking perpendicular to the grain at the sawn end of a tree (*left*) compared to a photograph (*right*, copyright 2005 Mayang Adnin). I used a concentric stress field, radiating outward from the center of the tree point, to model the anisotropic stress. Total computation time was 10.1 minutes on a 79,202 element mesh.

I show an example of the cracks created from a circular stress field and a photograph for comparison in Figure 4.6.

I can use my directional stress field initialization method to model more complicated patterns. If I want to set two directions in my field, as in impact patterns (see the following section), I simply add two directions to my stress tensor by using Equation (4.9) twice. I can also change the stress field over the course of a simulation to create two distinct interacting crack patterns. For example, I can run the simulation with the stress field defined by one direction, generating a set of cracks. I can then reinitialize the stress field using another direction to give a second set of cracks that interact with the first. Such crack patterns can occur in rock due to different geological events, as found in the Entrada Sandstone area of Arches National Park, Utah [33]. I give an example of a crack pattern created with two different directional stress fields in Figure 4.7.

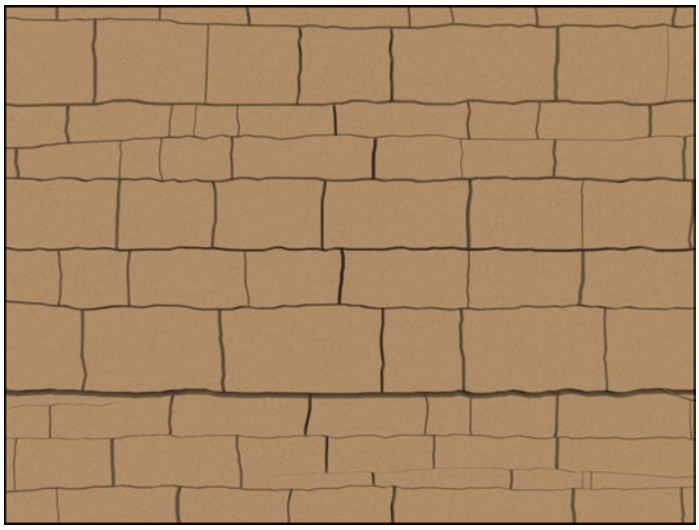


Figure 4.7: Some crack patterns in rock are created by multiple geological events, creating interacting crack sets as found in the Entrada Sandstone area of Arches National Park, Utah. I model the two crack sets by first initializing the stress field to have vertical stress, giving the horizontal crack pattern. I then reinitialize the stress field to have only horizontal stress, creating the vertical cracks. Total computation time was 3.1 minutes on a 19,602 element mesh.

4.2.3 Modeling Impact Patterns

I 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, I model a low-velocity impact fracture, the type of impact that produces surface cracks patterns instead of fracturing the glass apart. When the pane of glass is held in place, radial cracks initially form outward from the impact point, followed by concentric cracks [56].

I model this cracking interaction by initializing the stress field so that there is low stress radiating outward from the impact point and high stress in the concentric direction, as described in Figure 4.8 and demonstrated by Figure 4.9. I use the method discussed in Section 4.2.2 to set the stress tensor so that it is directed along these two directions.

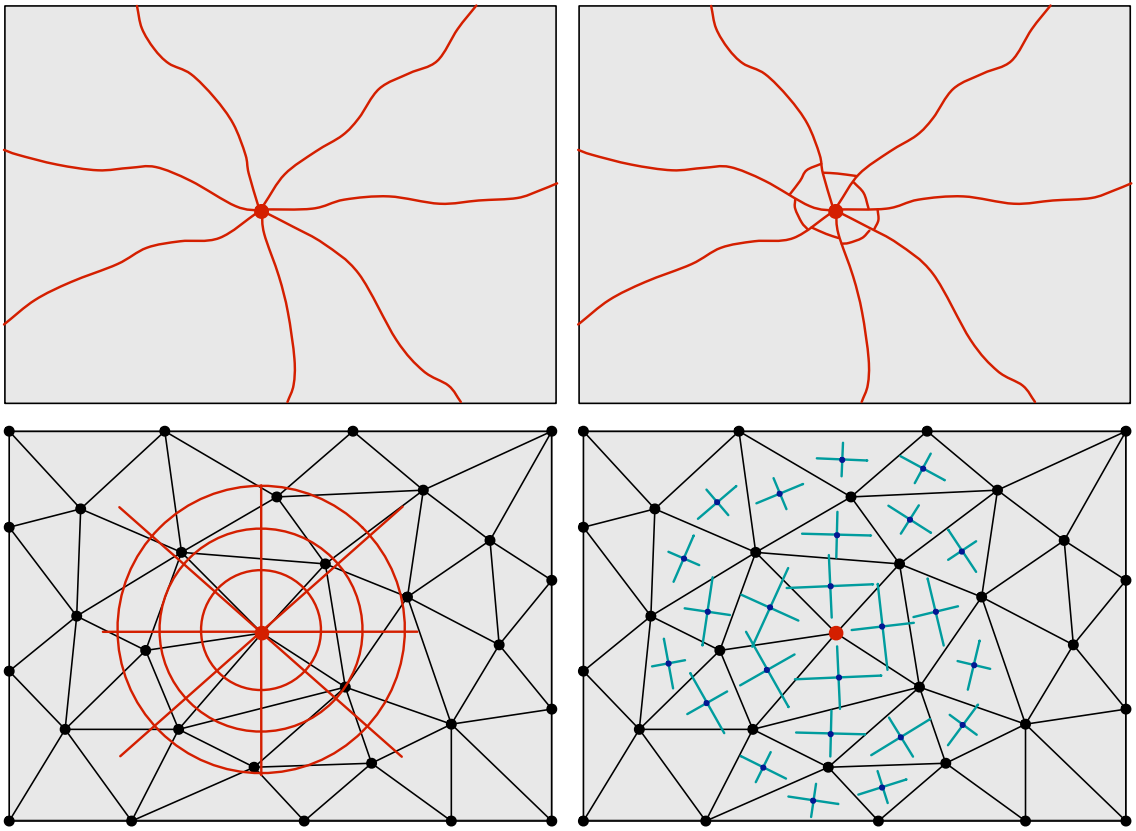


Figure 4.8: When an object impacts flat glass, cracks initially form radial from the impact point (*top left*) followed by concentric cracks around the point (*top right*). I model this type of pattern on a mesh (*bottom left*) by initializing my stress field to create cracks in the appropriate directions (*bottom right*).

After picking an impact point, $\boldsymbol{p}$, I determine the area where the stress pattern is defined from a user-specified radius, r. I define a falloff function to weight the stress amount at each triangle center $\boldsymbol{c}_i$ so that the stress is initially concentrated near the impact point and decreases to zero at the edge of the impact radius. After experimentation, I found the following weighting function to give me the desired falloff

$$w_i = 2.0 \cos\left(\frac{d_i}{r}\right) \tag{4.10}$$

where d_i is the Euclidean distance from the i^{th} triangle center to the impact point. I

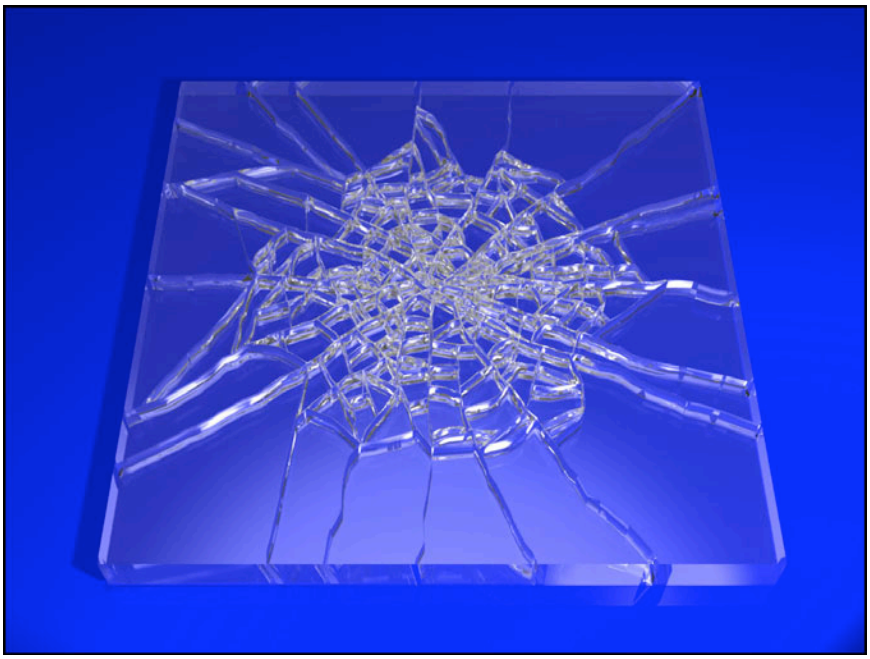


Figure 4.9: Cracks generated from initializing the stress field with a pattern modeling impacts in flat glass. The field is evolved by a relaxation process, causing the cracks to propagate. Total computation time was 15 seconds with an input mesh of 4,804 triangles.

determine the radial direction by first computing the vector

$$\boldsymbol{v}_{rad} = w_i \left(\frac{\boldsymbol{c}_i - \boldsymbol{p}}{|\boldsymbol{c}_i - \boldsymbol{p}|} \right) \tag{4.11}$$

and then converting it to the triangle's coordinate system to obtain $\boldsymbol{v}'_{rad}$. The concentric direction is the perpendicular vector given by

$$\boldsymbol{v}_{circ} = 1.25 \left(\boldsymbol{d}'_{rad} \right)^{\perp} \quad . \tag{4.12}$$

Recall that cracks form perpendicular to the largest force direction acting on a node. To ensure that the radial cracks form before concentric cracks, I use the scaling factor 1.25 in the above equation, giving larger forces in the concentric stress field than the radiating stress. I found this scale factor by experimentation.

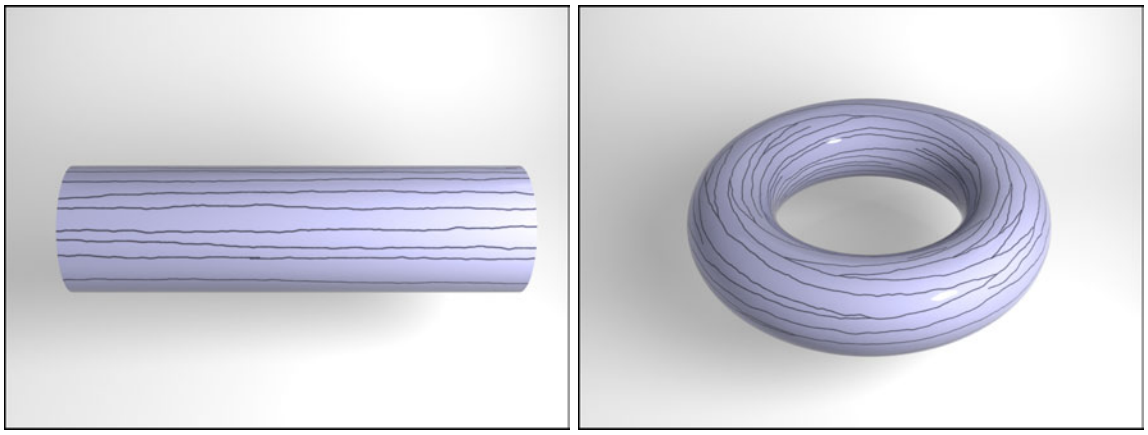


Figure 4.10: For these two examples, I cause the cracks to form perpendicular to the highest principal curvature direction by setting the stress tensor to the curvature tensor. Cracks form horizontally in the cylinder on the *left*, with some noise introduced by the discretization of the mesh. This noise is more apparent in the example of the torus on the *right*.

4.2.4 Using Curvature

Another way to update the stress field is with the object's curvature. As a pre-processing step, I estimate the principal curvature at the centers of my triangles using the discrete operator given in [12]. My implementation of this method closely follows the description presented in [1]. After computing the curvature tensor, I project the principal curvature directions to lie in the local 2D coordinate system of the triangle in which it resides. This projection enables the curvature tensor to be easily combined with the stress tensor, also stored in the triangle coordinate space.

To create interesting crack formations, I use curvature in various ways as an update. For example, I can add a specified amount of the curvature tensor to the stress tensor, causing cracks to be more prone to form perpendicular to high principal curvature. Using curvature in this manner is demonstrated in Figure 4.10. The cracks form along the length

of the cylinder (Figure 4.10, *left*), perpendicular to the direction of high principal curvature. For the torus (Figure 4.10, *right*), the cracks form in rings, although they are not perfectly circular due to the noise introduced by the mesh discretization. I give a more complex example in Figure 4.11 by combining the curvature tensor metric with uniform shrinkage of the surface area. Note that the cracks form along the cylindrical arm and in the folds of cloth.

Curvature may be used in other ways to change the crack pattern. For example, I 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

Figure 4.11: As an artistic effect, I initialized the stress field to uniform tension with some bias to crack perpendicular to 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 hours with an input mesh of 105,772 triangles.

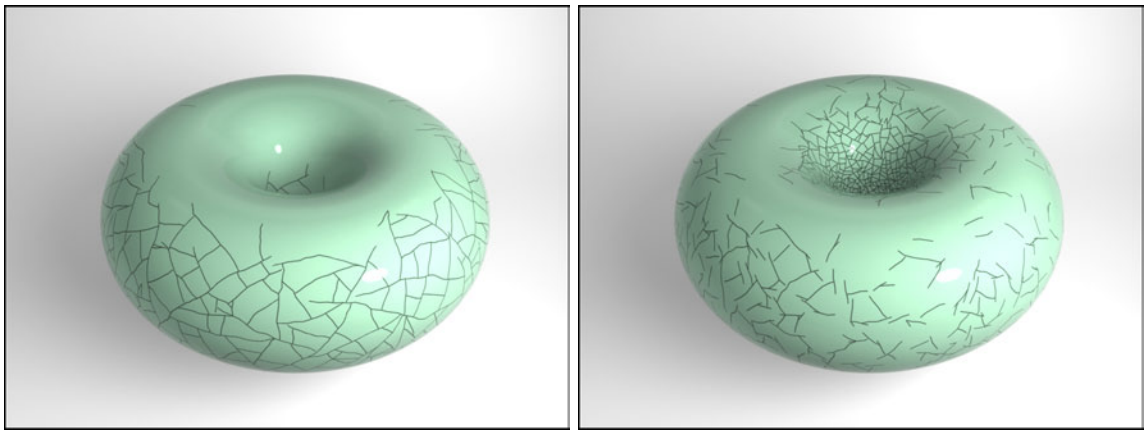


Figure 4.12: For these examples, I multiplied the separation tensor by a scale factor based on curvature to give different concentrations of cracks. The example on the *left* uses the Frobenius norm of the curvature tensor, causing cracks to form on the area of highest positive curvature, the outside of the torus. The *right* example uses the sum of squared principal curvature values, $\kappa_1^2+\kappa_2^2$. Notice that a higher concentration of cracks have formed on the inside of the torus where there is higher curvature.

tensor controls the distribution of cracks on the surface. For example, using the Frobenius norm causes areas of combined higher curvature to have a denser concentration of cracks, as demonstrated in Figure 4.12, *left* and Figure 4.13, *left*. I also give an example of using a second scale factor, the sum of squared principal curvature values, in Figure 4.12, *right* and Figure 4.14.

By combining these two methods, I can use curvature to model the stress created during the expansion of an object. Consider a cylinder that has a core expanding outward. The tensile stress forms in circular bands around the cylinder, approximately equal to the positive normal curvature of the surface. As a result, the cracks form along the length of the cylinder, perpendicular to the lines of positive curvature. See Figure 4.10, *left* for an example of cracks perpendicular to the lines of high curvature. In the case of an expanding bowl-shaped object, cracks would form on the outer convex side but not on the concave

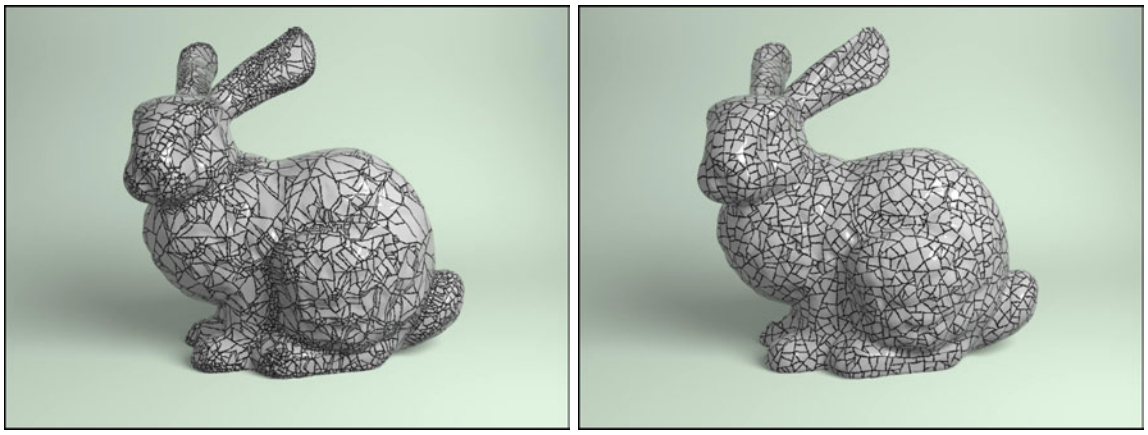


Figure 4.13: Curvature can be used to control crack formation, as demonstrated by the image on the *left* requiring 2.55 hours computation. Notice the high concentration of cracks around areas of high curvature, such as the ears, neck and leg. The *right* image was simulated using only uniform shrinkage of the surface, resulting in more evenly spaced cracks and requiring 2.0 hours. The input mesh size was 51,382 triangles for both examples.

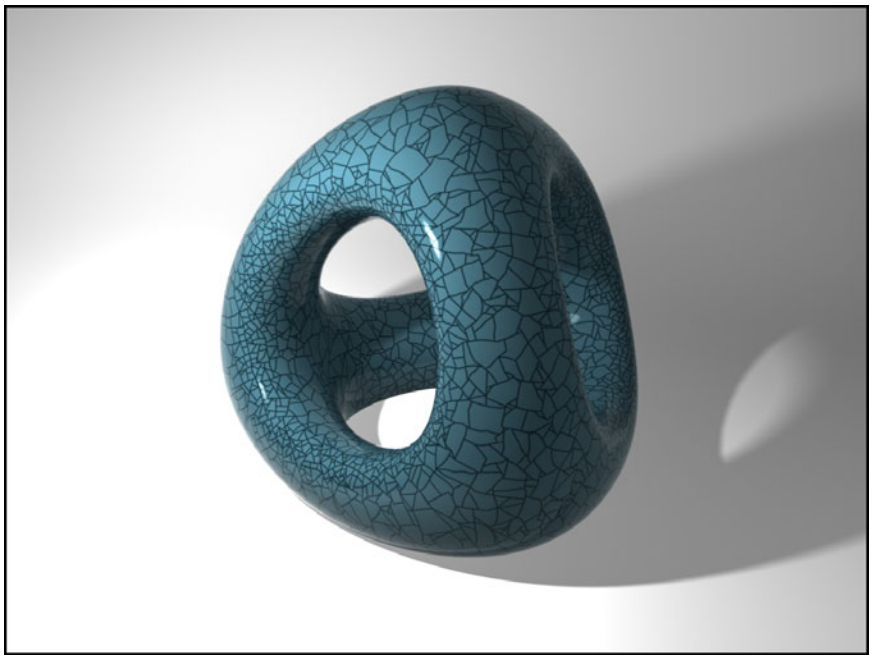


Figure 4.14: Another example of an artistic effect achieved by using curvature to bias the concentration of cracks. In this example, the separation tensor was weighted by $\kappa_1^2 + \kappa_2^2$ where κ_1 and κ_2 are the principal curvature values. Computation time was 1.23 hours on a 30,008 element mesh.

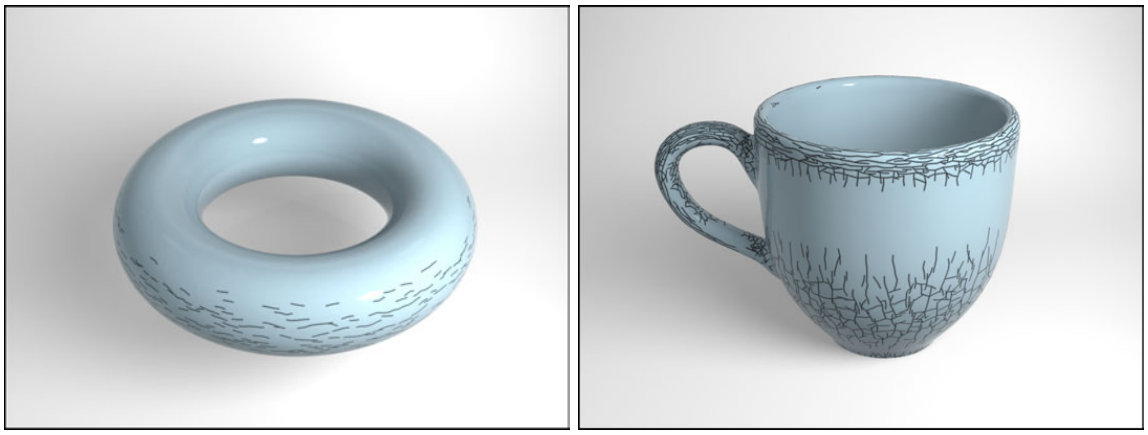


Figure 4.15: I use curvature to model the stress field and bias the concentration of cracks created by expanding materials. The image on the *left* gives a simple torus example, showing that the cracks only form on the areas of high principal curvature. On the *right*, I show a more geometrically complicated example. Higher concentrations of cracks are found on the outer part of the handle and the lip of the cup, areas of highest curvature, with less cracks on the body. No cracks are found on the inside of the cup past the lip.

side due to the compression produced by negative curvature.

Using these insights, I can set the stress tensor and scale factor for the separation tensor to generate interesting crack patterns. This metric updates the stress field by adding a multiple of the positive normal curvature to each stress tensor. I set the separation tensor scale factor to be a multiple of the largest positive principal curvature value or zero if there is no positive curvature. Such a scale factor causes a higher concentration of cracks to form where there is higher positive curvature. I give examples of a torus and a cup using this curvature metric in Figure 4.15.

4.2.5 Defining Arbitrary Stress Patterns

My method also allows the user to arbitrarily set the stress field. For example, I used texture mapping to define the concentration of stress on the surface. Given a gray-scale

Figure 4.16: I can use an input image (*left*) to define areas with high stress and produce the small scale cracks shown in the center (*right*). I 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. Total computation time was 18.2 minutes on a 72,202 element input mesh.

image, I 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 I decided to use the range from $[0, 1]$ by directly using the intensity of the colors from white to black. For my example, I multiply this weight by a stress tensor of uniform shrinkage. This method gives the highest stress where there is black in the input image and no stress where there is white.

I give a simple input image in Figure 4.16, *left*, and the resulting crack pattern on the *right*. However, I created this image in two steps. First, I generated the small scale cracks using the input image, as described above. After the simulation completed, I reduced any remaining stress on the surface to avoid further cracking near the letters. I then initialized the stress in the triangles surrounding the letters using uniform shrinkage of the surface area to produce the large scale cracks.

4.2.6 Random Noise

Although the unstructured discretization inherently introduces some randomness to the stress tensor, I give the user the option to add more variation. I add randomness in two places in the stress tensor. First, I can vary the amount of uniform shrinkage over the mesh by adding a pseudo-random number to the constant value c in Equation (4.8)

$$\boldsymbol{\sigma}' = \boldsymbol{\sigma} + (c + r_1)\,\mathbf{I} \tag{4.13}$$

where r_1 is a pseudo-random number in the range $[0, \chi]$. The user defines χ, a small constant value, as part of the input. Setting χ is dependent on the value of c and the range that the user would want it to vary. By default, it is set to zero, indicating no noise.

The second way I introduce randomness in my stress field is by varying the directions within the field. Changing a direction in the tensor amounts to adding a pseudo-random number, r_2, in the range $[0, \psi]$ to the two off-diagonal components of the stress tensor, to retain symmetry

$$\boldsymbol{\sigma}' = \begin{bmatrix} \sigma_{xx} & \tau_{xy} + r_2 \\ \tau_{yx} + r_2 & \sigma_{yy} \end{bmatrix} . \tag{4.14}$$

As before, the upper boundary for the pseudo-random number range, ψ, is user-defined.

It is essential to have a good pseudo-random number generator when using random noise to avoid introducing patterns in the stress field. Many good pseudo-random number generators exist for simulation [32]. For simplicity, I implemented the 32-bit multiplicative linear congruential generator [49] and found it to work sufficiently well for my purposes.

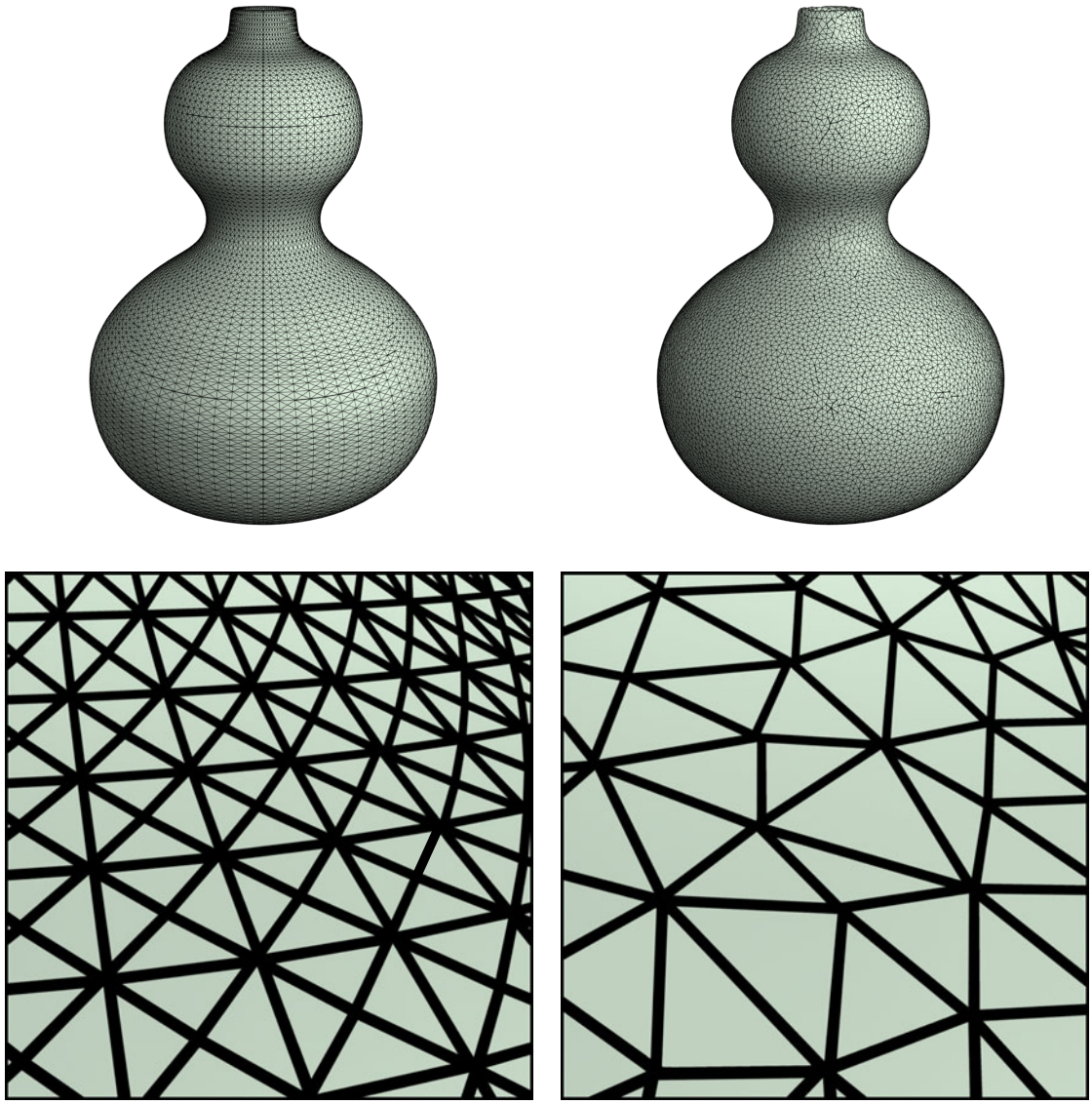


Figure 4.17: The *top row* shows two triangle meshes for the same object. The *bottom row* shows a zoomed in view of the same region on the object. The *left column* shows a structured triangle mesh (a regular 4-8 mesh). I can break the regularity by using existing algorithms to give a less structured triangle mesh, such as the mesh in the *right column*.

4.3 Artifacts from Exactly Uniform Stress Fields

Meshes that follow a regular grid pattern, such as triangulating a quadrilateral mesh by inserting diagonal edges, are undesirable for my algorithm. Such regularly struc-

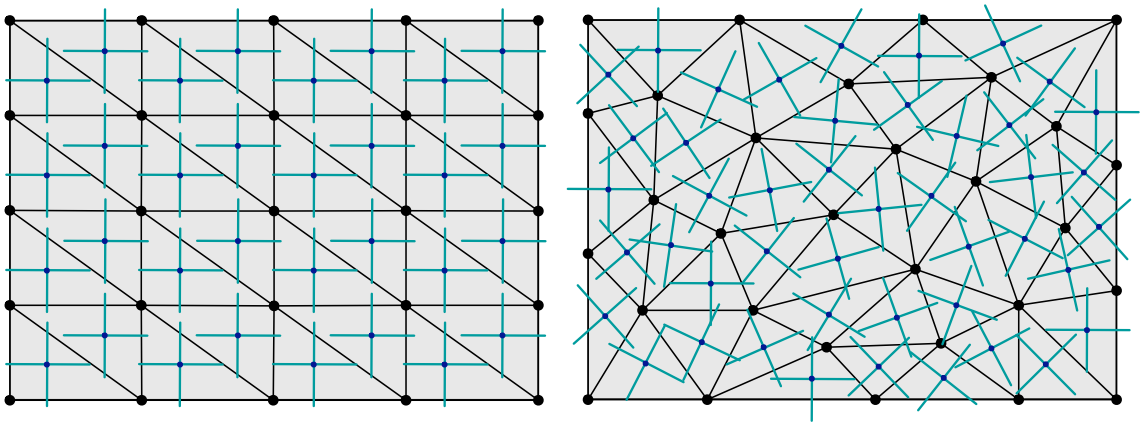


Figure 4.18: A regular mesh results in a regular stress field (*left*), while irregular meshes give a stress field pointing in all directions (*right*).

tured meshes include a 4-8 mesh that has alternating vertices of valence four and eight, as illustrated in Figure 4.17, *left*. These meshes introduce identically uniform patterns in the stress field when initialized using uniform shrinkage of the surface area.

The uniform stress patterns arise from adding uniform shrinkage using Equation (4.8). I initialize the stress field to the identity matrix times a constant. The resulting principal stress directions rely on the triangle coordinate axes because I convert the stress tensor to world coordinates using the basis matrix (Equation (3.2)). As a consequence of the mesh following a regular grid pattern, the triangle coordinate axes are most likely in the same orientation, introducing biased directions in the stress field. A uniform shrinkage stress field transformed by similar triangle axes may result in a field resembling Figure 4.18, *left*. Such a stress field yields unrealistic crack patterns (see Figure 4.19, *left*).

Many models representations, including B-Spline patches and Catmull-Clark subdivision surfaces, result in quadrilateral meshes when discretized. Applications often create triangle meshes from these models by first generating the quadrilateral mesh and then divid-

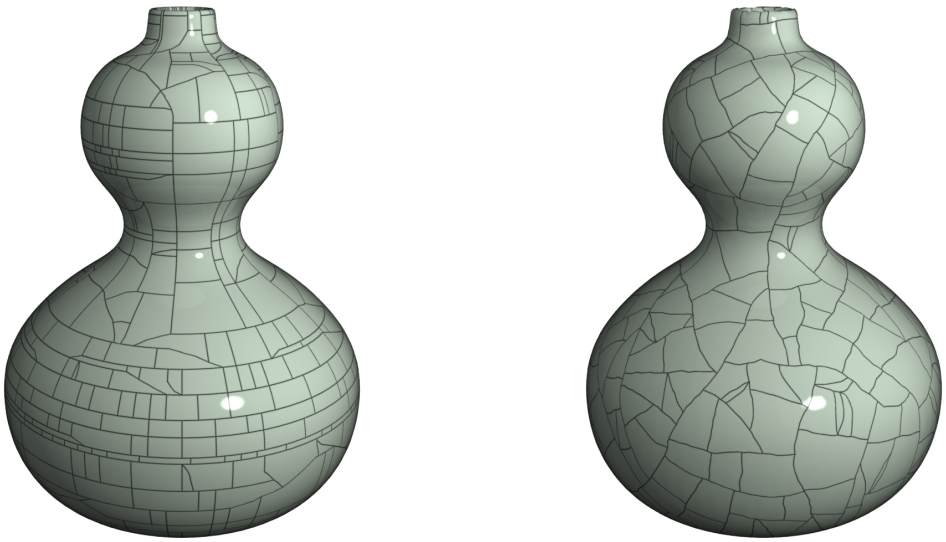


Figure 4.19: Structured triangle meshes introduce unnatural regularity in the cracks, (*left*), while irregular meshes produce more realistic results (*right*). See Figure 4.17 for the meshes used to produce these results.

ing each element. Directly using these results would give unnaturally regular crack patterns due to the bias in the stress field discussed above. Fortunately, there are many methods to convert a regularly structured mesh into one without structure. I found the re-tiling method presented in [63] to nicely break the regularity. For example, I created the mesh on the *right* of Figure 4.17 from the regularly structured 4-8 mesh on the *left* using their algorithm. The discretization error in an unstructured mesh adds inherent randomness to the stress field, as illustrated in Figure 4.18, *right*. The crack patterns resulting from an unstructured mesh are more plausible than the patterns produced using a structured mesh (see Figure 4.19).

Chapter 5

Generating Cracks

As described in Section 3.4, I use the separation tensor ς to determine where to crack the mesh. I insert the largest positive eigenvalue, v^+, of each node's ς into a priority queue. I 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

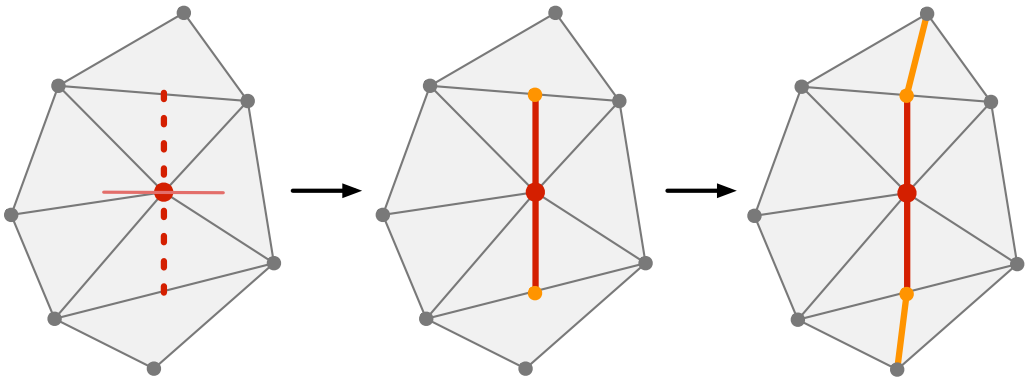


Figure 5.1: If the maximum eigenvalue of a node's separation tensor is above the material threshold, then I use the corresponding eigenvector (horizontal line) to compute the crack plane (vertical dotted line), as shown on the *left*. I intersect this plane with the surrounding triangles to insert the crack edges, shown as bold edges in the *middle*. Lastly, I do a local remeshing step to ensure that the surrounding elements are triangles (*right*).

corresponding node. I generate cracks in the direction perpendicular to the corresponding eigenvector, $\hat{\mathbf{n}}^+$.

To compute where new edges should be inserted in the mesh, I intersect the plane defined by $\hat{\mathbf{n}}^+$ with the surrounding triangles. I then split the intersected triangles, creating new free boundary edges in the mesh (see Figure 5.1). I do not add a new crack edge if it is too close to a current crack edge, avoiding back-cracking as discussed in [46] and illustrated by Figure 5.2. This plane-triangle intersection is fast in comparison to splitting and remeshing tetrahedra, an advantage of computing cracks on a surface discretization.

Because I treat stress as constant over an element, I duplicate much of the information contained at the center of a triangle when it is divided into two. However, the topology invalidates the triangle's local 2D coordinate system and the barycentric basis matrix (β),

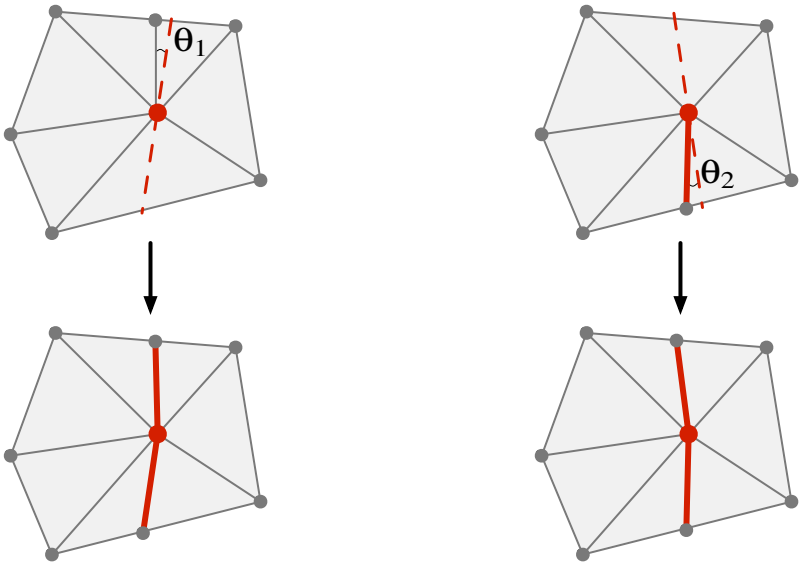


Figure 5.2: The crack plane (dotted line) gives the location of a new crack. Bold edges are crack edges. *Left:* Cracks are snapped to an existing edge if θ_1 is less than a set threshold (ex: 15°). *Right:* Cracks are also snapped to an existing crack edge when θ_2 is less than a set threshold (ex: 25°), avoiding back-cracking.

requiring that this information be recomputed. To speed up the simulation computations, I also cache any data constant over the triangle's lifetime, including the partial derivatives from Equation (4.6) and the constant pieces of the force calculation, Equation (3.1). Because these equations rely on information that has now been changed, I also update all cached data at this time.

The stress tensor, defined in the local 2D coordinates of its triangle, also requires updating when the local coordinate system changes. Because the plane of the triangle is unchanged, the local 2D coordinate system only undergoes a rotation. The stress tensor update amounts to

$$\boldsymbol{\sigma}' = \boldsymbol{R}^T \boldsymbol{\sigma} \boldsymbol{R} \tag{5.1}$$

where $\boldsymbol{R}$ is the rotation matrix transforming the old local 2D coordinate system into the new one.

5.1 Propagating Cracks

Theories from linear elastic fracture mechanics state that fractures propagate when the stress intensity factor exceeds the material toughness threshold. Using this theory, I generate my cracks as describe above. However, for Mode I fractures, the tensile stress is largest in the fracture plane for a small radius around the crack tip. This loading causes a crack to continue propagating in its own plane perpendicular to the greatest tension direction at the crack tip [13]. To follow this principle, I incorporate a method to model the distance a crack continues propagating in its current crack direction before pursuing a different crack direction.

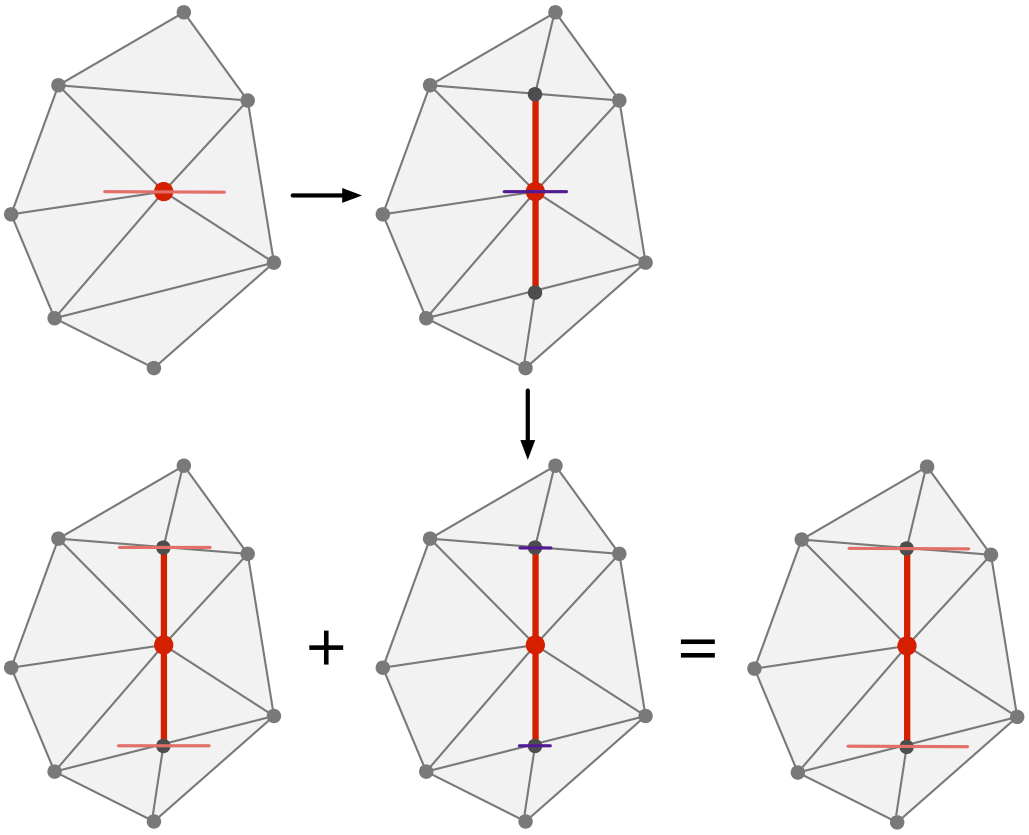


Figure 5.3: Each node has a maximum eigenvalue/eigenvector pair computed from its separation tensor (*top left*). When the maximum eigenvalue on the mesh exceeds the material toughness threshold, the algorithm inserts a crack and updates that node's separation tensor (*top right*). However, a residual amount of the maximum eigenvalue remains after the update. To account for the remaining force at the node, I multiply the residual amount by a scale factor at the crack tips (*bottom middle*) and add it to the crack tips' current separation tensor (*bottom left*) to obtain an updated tensor encouraging crack propagation in the crack's initial direction (*bottom right*). Note that I simplified this example so that the eigenvectors at the crack tips point in the same direction as the crack node; normally, the eigenvectors would vary.

The results generated by my method depend on the resolution of the model's discretization, only generating a crack in the elements surrounding the crack node at each time step. Because my algorithm takes discrete time steps, I do not crack a node exactly when the cracking criteria, v^+, reaches the material toughness threshold, τ. Instead, $\mathsf{v}^+ > \tau$,

leaving some amount of the maximum force pulling apart the crack node after cracking has occurred. To account for this force remaining at the crack node, I use the residual value remaining to further propagate a crack at its two crack tips, as presented in [44].

After cracking a node, I compute the residual value by finding the difference between the maximum eigenvalue and the material toughness threshold,

$$\mathsf{v}^* = \mathsf{v}^+ - \tau \quad . \tag{5.2}$$

I then account for the remaining force by modifying the separation tensor of the crack tip nodes using a portion of the residual value. The crack tips' separation tensor becomes

$$\boldsymbol{\varsigma}' = \boldsymbol{\varsigma} + \alpha \, \mathbf{m}(\hat{\mathbf{n}}^+)\mathsf{v}^* \tag{5.3}$$

where α is an input parameter ranging from $[0, 1]$. See Figure 5.3 for an illustration of this equation's effect on the separation tensor of the crack tips.

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 Figures 5.4 and 5.5. For my trials, I found α values between 0.5 and 1.0 to generate nice results. This parameter can not be larger than one because it would continually increase the separation tensor at each step, quickly causing the simulation to go unstable.

When the user gives a large value for the crack propagation parameter α, the two crack tip nodes will often be near the top of the priority queue due to the large amount of residual added. As a result, the simulation continues cracking at these crack tips soon

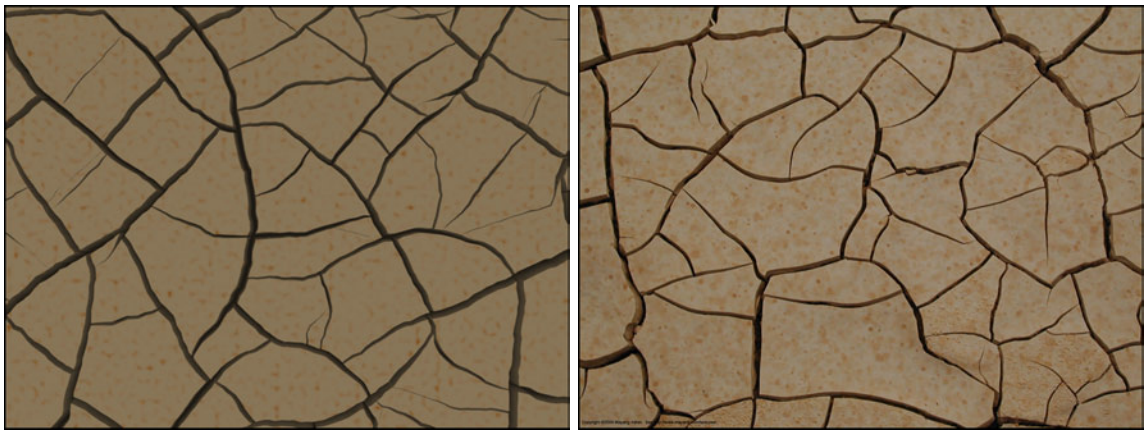


Figure 5.4: A rendered example of dried mud (*left*) compared to a photograph (*right*, copyright 2004 Mayang Adnin). I used uniform shrinkage of the surface and set $\alpha = 0.85$ to propagate the cracks further, requiring 2.05 minutes computation time on a 19,602 triangle input mesh.

after their creation by pursuing the one with the highest cracking criteria first. Relaxation dissipates the stress as cracking occurs, causing the second crack tip to eventually becomes next in the priority queue. Cracking then continues on the other side of the crack. The resulting behavior is a rippling effect where failure occurs outward from both sides of the original crack node as the crack grows.

Even after updating the separation tensor using Equation (5.2), I found that I was 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 formation, I add a small constant factor to the residual, v^*. I found that the value 0.25 produces the desired result, as demonstrated in my ceramic glaze examples (Figures 4.3 and 4.2).

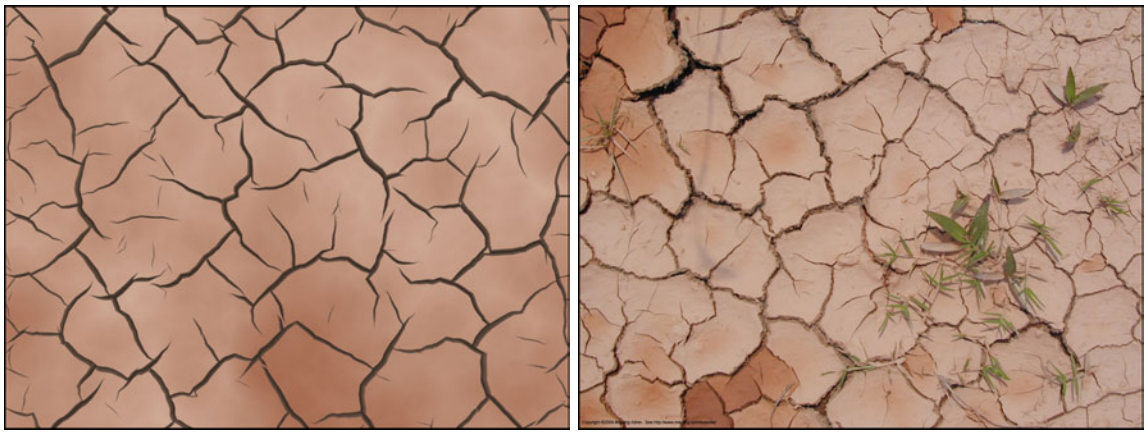


Figure 5.5: A comparison between rendered dried mud (*left*) and a photograph (*right*, copyright 2004 Mayang Adnin). As in Figure 5.4, I used uniform shrinkage of the surface, but changed $\alpha = 0.5$ to demonstrate controlling the crack propagation parameter. Total computation time was 1.5 minutes on a 19,602 triangle input mesh.

5.2 Avoiding Numerical Instabilities

Small or poorly shaped elements can introduce numerical instabilities in the simulation. I attempt to avoid creating ill-shaped triangles by snapping crack edges to mesh edges if they are close, as presented in [46]. For triangles, I have two edge snapping cases, illustrated by Figure 5.2. The first case occurs when I introduce a new crack at a vertex. If inserting a crack edge would create a sliver triangle, I instead convert the closest edge to a crack edge. The second case occurs when extending a crack tip. I want to avoid adding a crack close to an existing crack, or "back-cracking," because it would appear unrealistic. However, some slivers are large enough that edge snapping would create objectionable artifacts, yet still small enough that they would generate poorly conditioned elements.

I can trace the numerical instabilities in the simulation back to very large singular values in the β matrix. I assume that the input mesh contains well shaped triangles. This assumption is reasonable because I can use methods to generate meshes good for the

simulation's needs, as described in Section 4.3. I precompute a singular value threshold for the mesh by averaging the singular values of the input triangles' β matrices

$$\zeta = m \left(\frac{\sum_{i=1}^{N} \max(\sigma_i)}{N} \right) \tag{5.4}$$

where σ_i are the singular values of β, N the number of triangles, and m a constant multiplier. I chose $m = 8$ so that the threshold is not too small; I do not want to unnecessarily reconstruct matrices that are not ill-conditioned.

Each time I calculate the β matrix of an element, I compute its singular value decomposition, using the standard equation

$$\beta = U \Sigma V^* \tag{5.5}$$

where Σ is the diagonal matrix containing the singular values and U and V the matrices containing basis vectors. (See [54] for an explanation of the SVD.) I then examine the singular values in Σ and restrict them to the threshold if they exceed it. I reconstruct the β matrix using the new singular values and the basis vectors from Equation (5.5). The effect on the behavior of the simulation is that these small slivers become more compliant in the direction with large singularity. Because the slivers are often small and created near crack boundaries, the resulting crack pattern is not noticeably altered by changing the β matrix. This change enables the system to remain numerically stable even with poorly shaped elements.

Chapter 6

Post-Processing

During the simulation, I do not change the positions of the nodes but instead keep track of the forces acting upon them. For some examples, such as ceramic glaze, displaying the results does not need a gap between the cracks because this width is small. Instead, I can render these results directly, as described in Section 6.1. However, to gain the realism of larger crack formations, I perform a post-processing step to generate the crack width as described in Section 6.2. I can also add other types of effects after the simulation completes, such as curling (Section 6.3) and propagating cracks until they hit boundaries (Section 6.4).

6.1 Rendering Ceramic Cracks

For the ceramic examples, I implemented a shader that interpolates information about crack locations to render dark crack lines. I demonstrate this rendering technique in Figures 4.3 and 4.2. To achieve the glaze effect, I treat each triangle separately and annotate its vertices with the smallest distance to a crack edge, if present. I call this vertex

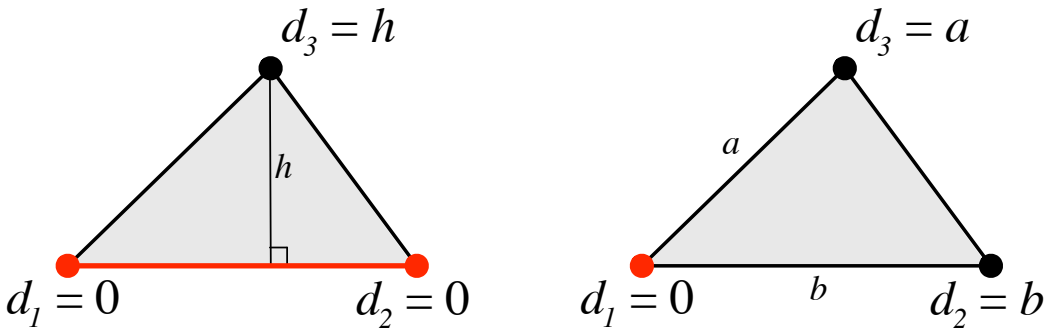


Figure 6.1: Triangles contain either zero, one or two crack vertices. Here I examine the two cases when cracks are present (one or two crack vertices). On the *left*, the bottom edge of the triangle is a crack edge. The distance for the two vertices that are part of the crack edge are $d_1 = 0$ and $d_2 = 0$. The distance for the remaining vertex is the distance to the crack edge, $d_3 = h$. The second crack case (*right*) shows a triangle with one crack vertex (bottom left vertex). In this case, the distance at the crack vertex is $d_1 = 0$, and distance for the regular vertices is the distance of the edge connecting it to the crack vertex ($d_2 = b$ and $d_3 = a$). Because the algorithm splits a triangle if it contains more than one crack edge, these two cases are the only distance measures involving cracks.

annotation value its "crack distance."

I compute the crack distance based on the number of crack vertices in the triangle. Figure 6.1 illustrates the cases for one or two crack vertices in a triangle. In both cases, I assign crack edge vertices a crack distance of zero. Otherwise, the crack distance for a regular vertex is either the distance to the crack edge (Figure 6.1, *left*) or the crack vertex (Figure 6.1, *right*). When a triangle contains no crack vertices, the distance for each vertex is the perpendicular distance to the triangle edge opposite that vertex. Consequently, all crack distance values are greater than zero in this case.

Because the crack distances are either zero, the distance to an edge, or the length of an edge, they are always non-negative. My vertex shader interpolates the distance values between vertices, so triangles without a crack edge or a crack vertex will not contain a crack edge coloration when shaded.

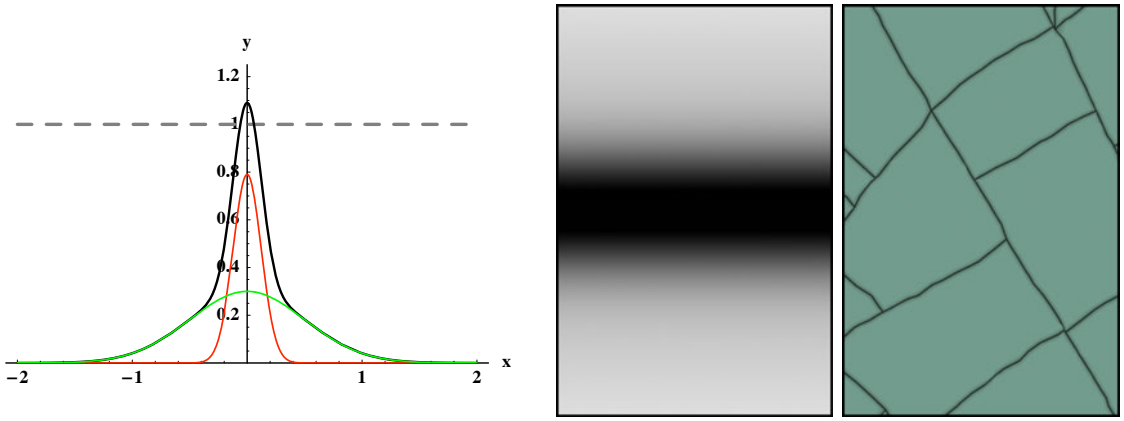


Figure 6.2: The black curve in the image on the *left* graphs the function from Equation (6.1). This function combines two Gaussian functions, the red curve ($0.79e^{-x_i^2/0.03}$) and the green curve ($0.3e^{-x_i^2/0.5}$). The horizontal dotted line illustrates where the function is capped to one. Using this function, the vertex shader darkens crack lines with a smooth falloff (*middle*). The resulting crackle glaze rendering gives the desired fuzzy crack effect (*right*).

The algorithm splits 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. Even with these precautions, some small artifacts appear in the rendering due to the triangle discretization. Because the cracks are small in relation to the overall image, these errors are not problematic.

Crackle glaze cracks often appear fuzzy because the dye used to make the cracks more prominent bleeds into the surrounding area under the glaze. I use a falloff function instead of a step function to model these soft crack edges. When the vertex shader computes values for a pixel, the renderer interpolates the crack distances to determine d_i, the distance to the nearest crack edge for that point. When this distance is within a threshold, m, determined by the model size, I compute the weight for the pixel color. Because I want the crack to have a dark line at some width surrounded by much lighter crack color, I defined

a falloff function by summing two Gaussian functions

$$\begin{aligned} x_i &= \frac{d_i}{m} \\ g_i &= 0.79e^{-x_i^2/0.03} + 0.3e^{-x_i^2/0.5} \quad . \end{aligned} \tag{6.1}$$

I determined the constant parameters for the Gaussian functions by experimentation and display the function and resulting crackle glaze effect in Figure 6.2. Notice that the function goes above one near zero, ensuring that the crack has appropriate width when rendered. To calculate the weight used for interpolating the color between the crack and ceramic colors, I limit g_i to the range $[0, 1]$, giving the dark band of crack color shown in Figure 6.2, *middle*.

6.2 Widening Cracks

For some of my examples, such as the impact pattern in Figure 4.9 and mud in Figure 5.4, I wanted to create a visible gap at the crack locations. To do so, I sum the change

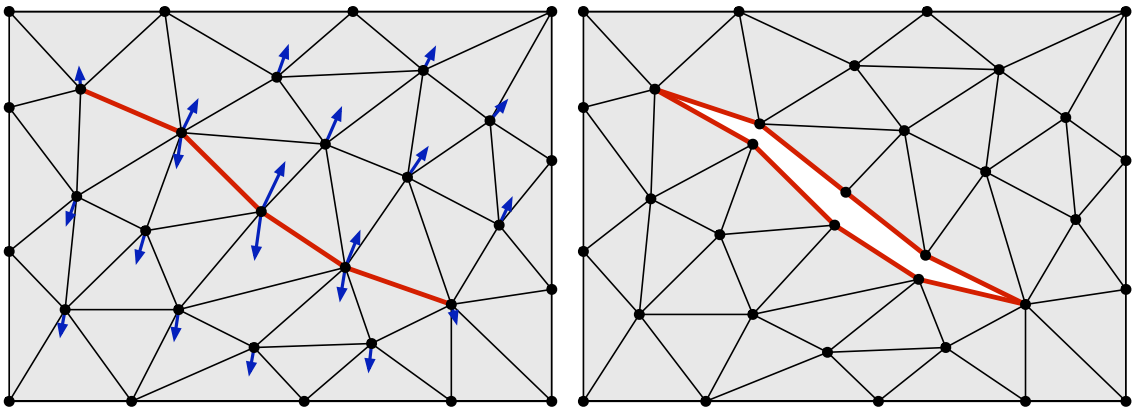


Figure 6.3: I do not move the mesh nodes during the simulation, but I do sum the change in positions computed during relaxation (vectors at the nodes in the image on the *left*). I can then use this information to move the nodes as a post-processing step, creating visual gaps (*right*).

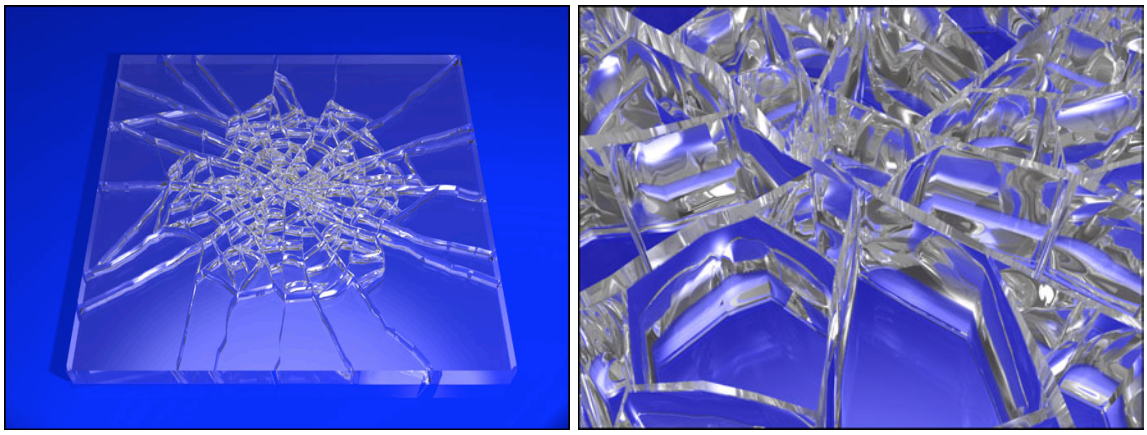


Figure 6.4: I created the image on the *left* by using the post-processing pipeline shown in Figure 6.5. The *right* image shows a close-up of the center of the impact pattern, demonstrating the gaps created by widening the cracks.

in positions from Equation (4.1) for each node n as I compute them in the relaxation process

$$\Delta_T \, \boldsymbol{p}_{[n]} = \sum_{i=1}^{R} \gamma \, \boldsymbol{F}_{[n]} \tag{6.2}$$

where R is the total number of relaxation steps and γ a parameter controlling the rate the node's of movement.

As a post-processing step, I displace the nodes by the vector computed in Equation (6.2). This displacement generates gaps in the mesh where the cracks occur, as illustrated in Figure 6.3. To fill the gaps, I build rectangles connecting the displaced crack node positions to the original positions, offset inward perpendicular to the surface by a user specified amount. This geometry creates nice side-walls for the cracks, as illustrated in Figure 6.4. Other examples using this method include the mud (Figures 5.4 and 5.5), crackle glass (Figure 1.1) and stone (Figure 4.11). Note that this post-processing movement is performed after the simulation has finished. Poorly shaped or inverted triangles that may result do not create significant problems. I give an illustrated example of this

post-processing pipeline in Figure 6.5.

6.3 Curling Cracks

I can also add a curling effect to the crack edges, as found in some mud and peeling paint. To model this phenomena, I use a portion of the result from Equation (6.2), giving the distance the node moved on the surface. I then calculate a curling factor by combining this force displacement and the normal direction at the node.

For each crack node, I calculate the vector to displace the node to add curling. Let $\boldsymbol{f}_n = \Delta_T \, \boldsymbol{p}_{[n]}$, l_f be the length of $\boldsymbol{f}_n$, and $\hat{\boldsymbol{f}}_n$ be the vector normalized. Let $\hat{\boldsymbol{n}}_n$ be the surface normal at that node. I can then calculate the initial node displacement for curling at node n by using the equation

$$\boldsymbol{c}_n = \mu \, l_f \, (\xi \, \hat{\boldsymbol{f}}_n + (1.0 - \xi) \, \hat{\boldsymbol{n}}_n) \tag{6.3}$$

where ξ and μ are scaling factors controlling the curl. For my example, I found $\xi = 0.3$ and $\mu = 5.0$ to give nice results.

After I have calculated the initial curl displacement for the crack nodes, I iteratively propagate $\boldsymbol{c}_n$ to the surrounding nodes to produce an overall lifting effect around the crack edges. I calculate the lift at a node by averaging the curl direction of its surrounding vertex ring with the displacement at the node. For non-crack nodes, there is no displacement vector, so I instead average the direction with the surface normal. I illustrate the lifting around a crack vertex in Figure 6.6.

Let $\boldsymbol{c}_{avg}$ be the average displacement vector of node n's vertex ring, l_c its length and $\hat{\boldsymbol{c}}_{avg}$ the vector normal. At each iteration, I average the displacements for the mesh

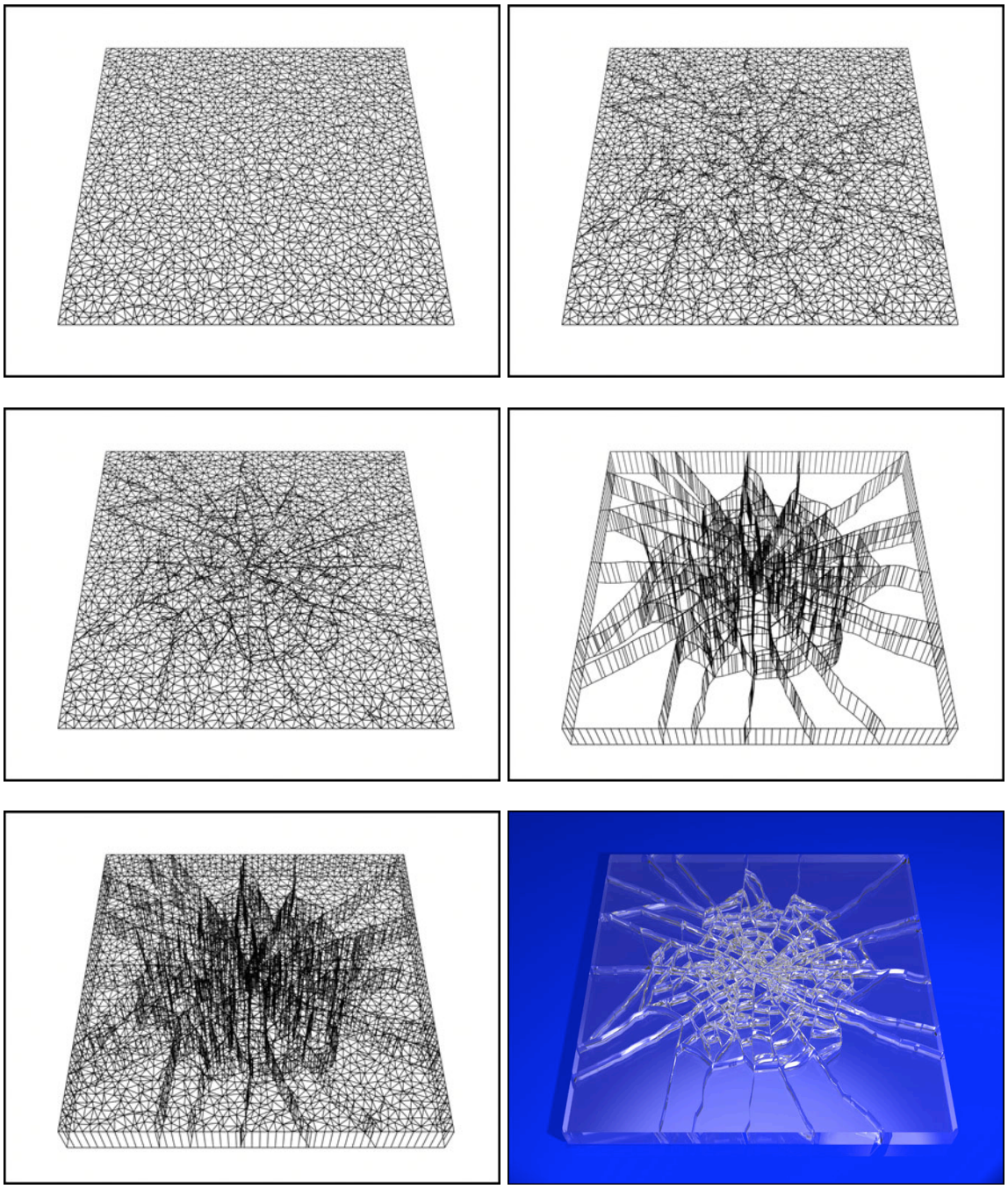


Figure 6.5: I start with a triangle mesh (*top left*) and run the surface cracking algorithm (*top right*). In the post-processing step, I first create gaps in the mesh (*middle left*) and then fill in the gaps by generating rectangles (*middle right*). These two sets of geometry (*bottom left*) are then used to render the final image (*bottom right*).

vertices by using the following equations. If node n is not a crack node, then I set the displacement vector to

$$\boldsymbol{d}_n = l_c \left((1.0 - w)\, \hat{\boldsymbol{c}}_{avg} + w\, \hat{\boldsymbol{n}}_n \right) \tag{6.4}$$

where $w = \gamma - i(\gamma/m)$, i the current iteration and m the total number of iterations. For my examples, I found $m = 10$ and $\gamma = 0.2$ to give good results. If the node is a crack node, I instead give more weight to the curling force and use the equation

$$\boldsymbol{d}_n = w\, \boldsymbol{c}_{avg} + (1.0 - w)\, \hat{\boldsymbol{n}}_n \quad . \tag{6.5}$$

Consequently, the crack vertices receive more curl, giving the overall desired effect.

I give an example of curled mud produced using this method in Figure 6.7. Note that there is also mud underneath the layer of curled mud, as demonstrated by the close-up view on the *right*. As the top layer of mud dries and curls, it often pulls away from an underlying moist layer. To produce this effect, I output the mesh after widening cracks but before curling. I then curl the mesh as described above and output another file. Rendering

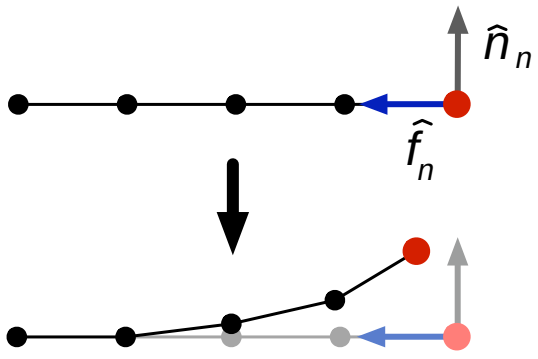


Figure 6.6: The *top row* shows one segment of mesh vertices from a profile view. The node on the far right is crack node n with surface normal $\hat{\boldsymbol{n}}_n$. The horizontal vector is $\hat{\boldsymbol{f}}_n$, the normalized force displacement vector for that node. The *bottom row* shows the mesh vertices after being displaced using my curling method.

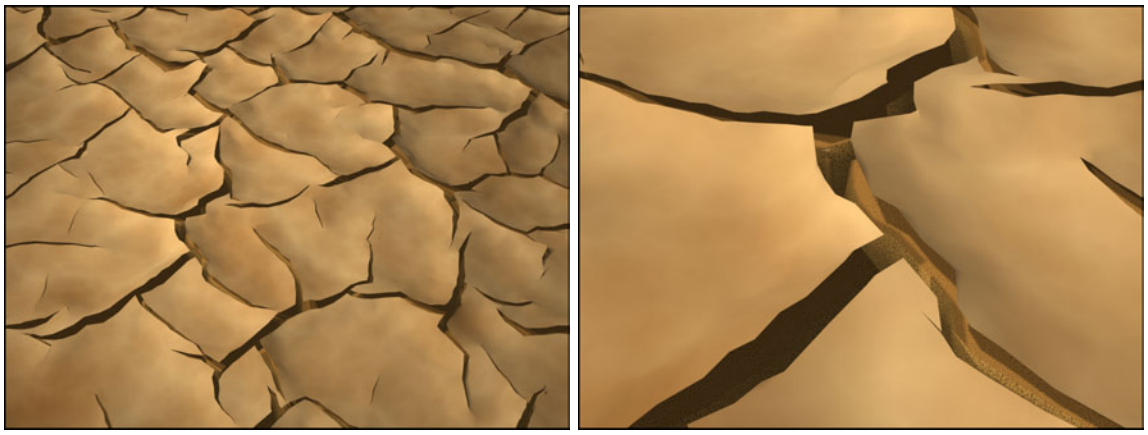


Figure 6.7: Example of adding a curling effect to the cracked mud from Figure 5.5, shown on the *left*. I give a close-up view of the curled mud on the *right*. This example required 1.5 minutes computation time on a 19,602 triangle input mesh.

both meshes together gives the final result, as shown in Figure 6.7.

6.4 Finishing Cracks

Because I allow the user to choose when the simulation terminates, some cracks may be incomplete. To give the user the ability to have these cracks completed, I devised a method to propagate these crack tips until they terminate at another crack boundary. My simulation keeps track of crack tip vertices while it progresses. However, crack tips may also form when two cracks intersect during growth. I detect this second case by searching for two consecutive crack edges that have an angle θ_f less than a threshold (ex: 75°) between them. By searching for angles between crack boundaries that are small, I ensure that the crack vertex joining the two crack edges is caused by an intersection and is not the continuation of the same crack. See Figure 6.8 for an illustration of the angle used for comparison.

Because the crack edges create open boundaries, there are two possible crack

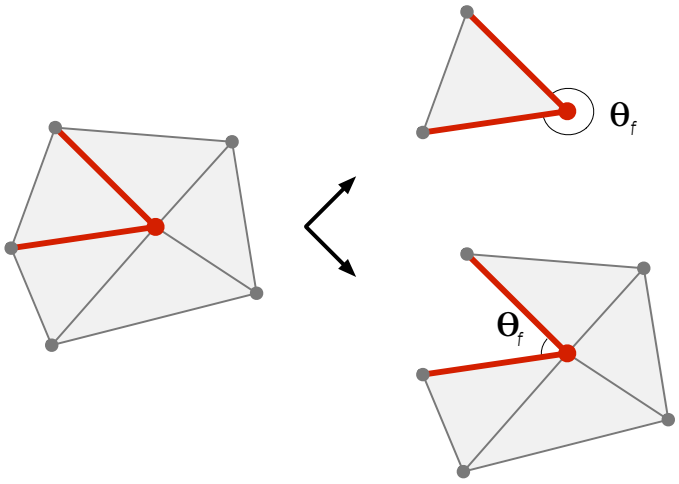


Figure 6.8: To detect crack tips caused by crack intersections, I compute the angle between adjacent crack edges, θ_f. Because the crack edges are inserted as open boundaries, there are actually two cases for each pair of crack edges, as shown on the *right*. When θ_f is less than the threshold for finishing cracks (ex: 75°), I annotate the crack vertex shared by the two crack edges as a crack tip so that the crack can be continued in the appropriate direction.

vertices to annotate as the crack tip for each pair of crack edges. To ensure that the crack continues on the appropriate side of the crack boundary, I compute the angle between the edges of the open boundary, as shown in Figure 6.8, and compare the result to the threshold. When θ_f is below the threshold, I label the associated crack vertex as a crack tip.

After the simulation has stopped, I finish cracks by setting the separation tensor of each crack tip's vertex to its residual tensor. This ensures that I continue cracking in the last known crack direction. I then crack the mesh as before, including the relaxation step because I need to compute the position change for generating gaps in the mesh. Finishing cracks alters the simulation results, so it must be executed before other types of post-processing. This method is used in some of my examples, including Figures 4.2, 4.14, and 5.4.

Chapter 7

Modeling Cross-Cutting Joint Sets

The simulation discussed in previous chapters generates crack patterns where new cracks terminating when they reach pre-existing cracks. However, there are some cracks patterns in nature where new cracks ignore older cracks, passing through them as if they were not present, called "cross-cutting." In geology, *joints* are fractures with no measurable shear displacement, primarily caused by tensile stress [6]. Thus, joints are cracks in rock created by stresses similar to the cracks in mud, glass and ceramic glaze modeled in earlier chapters. In this chapter, I present a method to model multiple cross-cutting joint patterns by recognizing that each joint set can be modeled and generated independently. I create the final image by combining the crack patterns produced during the different simulations into one mesh.

Cross-cutting joint patterns occur in two distinct cases. When the first joint set undergoes large normal compression, the fractured material is compressed together, creating a high resistance to slip between the two sides of the fracture. The resulting effect is that

Figure 7.1: Photograph of multiple cross-cutting joint sets found in Eagle-neck Hawk on the Tasman Peninsula in Australia (copyright 2006 Jonathan Mao, http://www.jonathanmao.com/gallery/main.php). This geological feature is also named "Tessellated Pavement" due to its regularity.

the rock acts as one piece of material instead of two separate pieces, allowing cross-cutting to occur. In the second way that these patterns form, minerals fill the gaps in an older joint set to produce veins. The vein creates a solid for the new joint set to pass through without inhibition.

Although a variety of cross-cutting patterns are found in nature, the most frequently occurring are two continuous orthogonal joints sets [13]. These patterns occur in many locations around the world, including the limestone in Malham Cove, England (referred to as "limestone pavement") and siltstone in Tasman, Australia. See Figure 7.1 for an example of a cross-cutting joint pattern.

I examine the simplest multiple-joint pattern resulting from two distinct joint sets [13], enabling me to easily describe my method. Using this model, I am able to generate

crack patterns found in limestone, sandstone, and siltstone using two simulations. I could easily extend my method to model more complicated multiple joint patterns by running a simulation for each joint set. However, I would also need to store the crack information specific to each additional simulation.

7.1 Algorithm for Two Simulations

I use the method presented in Section 3.1 to generate each distinct simulation. For each joint set, I initialize an appropriate stress field using heuristics and then run the physically based simulation. I augment this original algorithm by storing information during the first simulation that I later use for merging the crack edges. I use the resulting connected mesh for the second simulation. After completion, I again use stored information to recreate the first joint set on top of the second joint set, yielding the cross-cutting joint pattern.

I summarize the two simulation algorithm with the following steps:

1. Initialize the stress field for crack simulation 1
2. Run the simulation, storing additional data:
   (a) Record crack edge pairs
   (b) Maintain lists of duplicate vertices and vertex forces
3. Merge the crack edges
   (a) Annotate merged cracks as edges from simulation 1
   (b) Update the duplicate vertex list

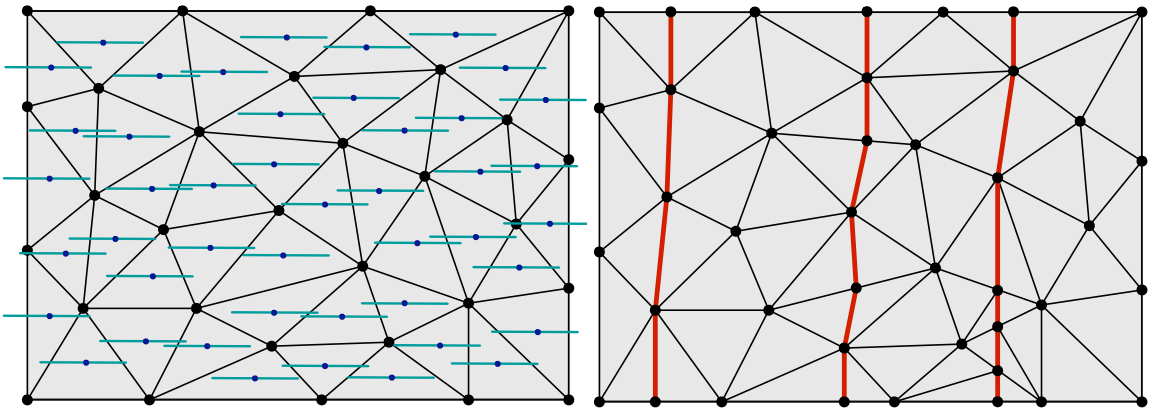


Figure 7.2: I define a stress field that models forces pulling horizontally in the plane of the mesh on the *left*. The resulting crack pattern produced by this stress field are vertical cracks, as shown in the image on the *right*.

4. Initialize the stress field for crack simulation 2

5. Run the simulation

6. Recrack the mesh using the stored data, adding saved force information to the vertices

7. Post-process the mesh to generate gaps between cracks

For example, consider the simulation steps needed to create two orthogonal joint sets. First I would initialize the stress field to model a directional force, as described in Section 4.2.2 and demonstrated by Figure 7.2. After I run the first simulation, I merge the cracks so that my mesh is whole, as shown in Figure 7.3, *right*. I then initialize a second stress field and run another simulation (Figure 7.4). The final result is the combination of these two crack patterns (Figure 7.5).

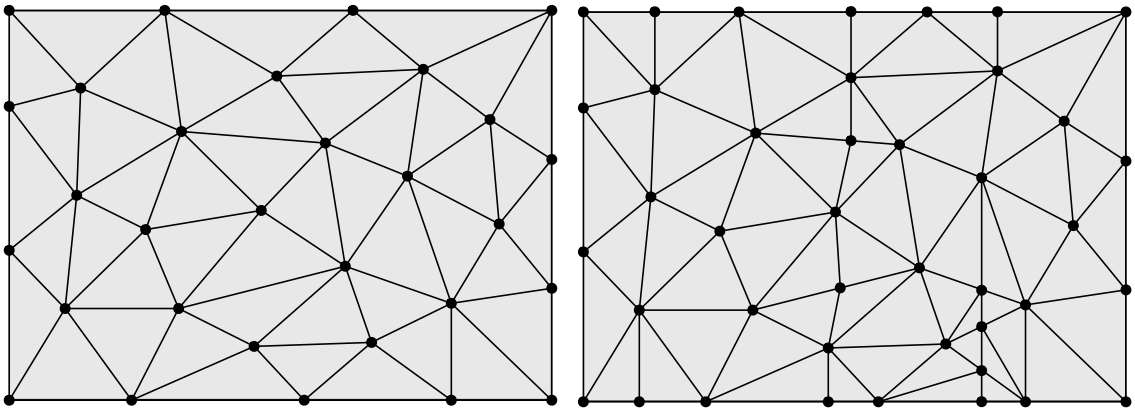


Figure 7.3: The *left* displays the initial mesh, before running the simulation. The *right* gives the mesh after merging the crack edges from Figure 7.2 together. Notice that the merged mesh contains the crack edges from Figure 7.2, *right*; however, these edges are no longer open boundaries in the new mesh.

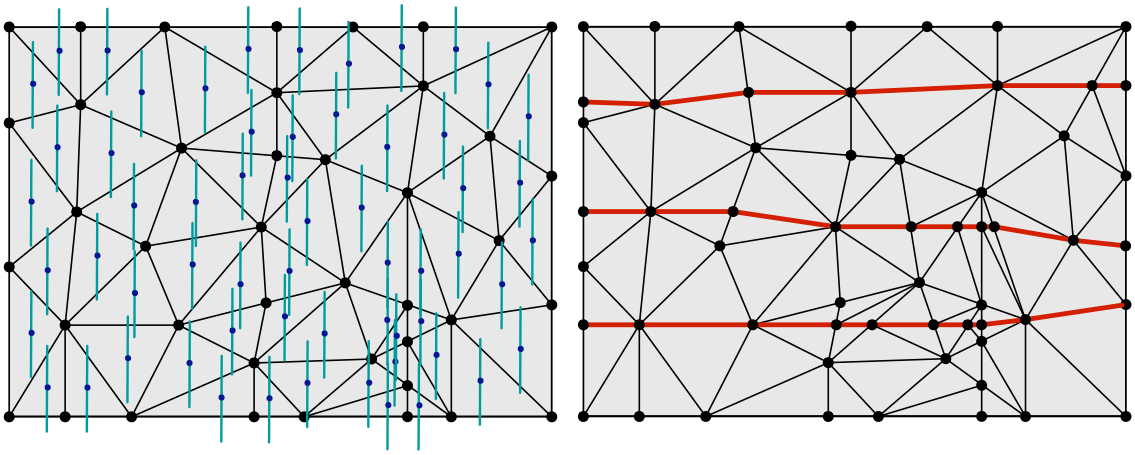


Figure 7.4: Forces pulling vertically in the plane of the mesh can be modeled using the stress field shown on the *left*. The resulting cracks are horizontal, as displayed in the figure on the *right*.

7.2 Data Required for Multiple Simulations

During the first simulation, I keep track of the pairs of crack edges that I will later use for merging and maintain lists of duplicate vertices. After the first simulation finishes, I use the duplicate vertex lists to save the vertex forces for their respective nodes. I record

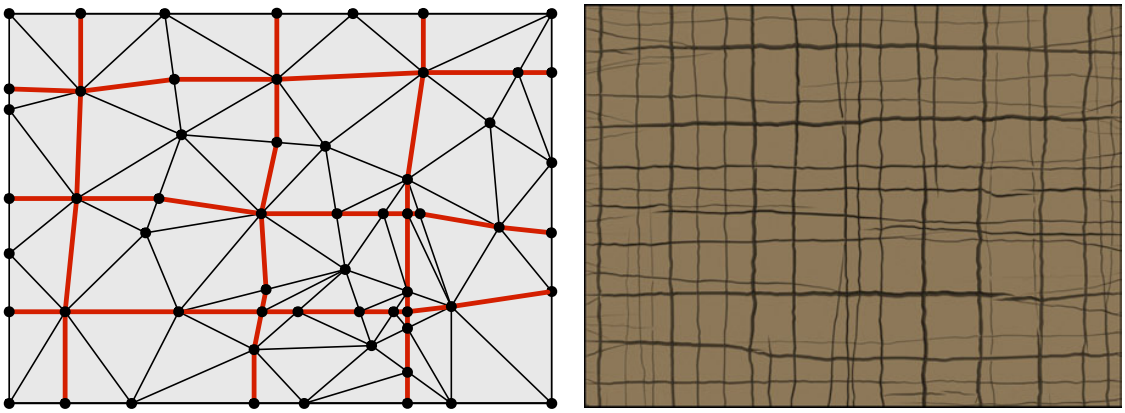


Figure 7.5: The figure on the *left* results from combining the two separate simulations from Figures 7.2 and 7.4. I present an image generated using this multiple simulation approach on the *right*.

this information because I want to create gaps between cracks in the final result for the first joint set; otherwise, I would loose this data during the merge because I delete duplicate vertices. After saving the information, I iterate through the crack edge pairs list, merge the edges as described in Section 7.3 and annotate the single edge as a crack from the first joint set. The resulting mesh contains no open crack boundaries.

I reinitialize the stress field using heuristics and run the second simulation, updating the duplicate vertex lists and old edge annotations from the first simulation as needed. I also interpolate saved vertex forces when a new crack crosses an edge annotated as an old crack. Because the vertex force information is now stored in data lists, this task can be tricky due to the loss of crack boundaries and connectivity information between duplicate vertex lists.

To handle this data loss, I impose an ordering on the vertex forces when initially stored. First, I sort the list of duplicate vertices so that they are in counter-clockwise order around a reference vector. Any vector may be used for the reference vector; in my case,

I chose to use the bisector between the first two crack edges encountered when creating the duplicate vertex list. For each vertex in this list, I compute the bisector between its two crack edges and calculate the counter-clockwise angle between the bisector and the reference vector. These angles determine the final ordering of the vertices by sorting them from smallest to largest. Using the vertex ordering, I can then sort the vertex forces.

7.3 Merging the Crack Edges

After the first simulation finishes, I use the crack edge pairs to merge the crack edges together. I do this step to obtain a mesh with the same genus as the initial mesh, enabling me to run the second simulation independent of the cracks from the first simulation.

Given a pair of crack edges, I first remove edges representing the open boundary. I then merge the vertex rings of the crack edges' duplicate vertices and remove the extra vertex. I also update the duplicate vertex list to reflect this change. Lastly, I remove the duplicate crack edge and mark the remaining edge as a regular edge for the next simulation. At the same time, I also annotate that edge as a crack from the first simulation. I use this information in Section 7.4 to recrack the mesh after the second simulation has finished.

7.4 Recracking the Mesh

When the second simulation completes, I recreate the first joint set on top of the mesh containing the second joint set. By iterating through the edges annotated as old cracks, I reopen boundaries in the mesh in the same manner as when a crack is initially added. Once completed, I have one mesh with a crack pattern representing two separate

simulations, as shown in Figure 7.5.

I then add the saved vertex forces to the appropriate vertices by imposing the same counter-clockwise ordering described above on the current duplicate vertex list. However, new duplicate vertices may have been added to the list during the second simulation, so there is not necessarily a one-to-one correspondence between the vertex forces list and the duplicate vertex list. If they are the same length, I just apply the vertex forces to the corresponding vertices. Otherwise, I need to find the correct pairing between forces and duplicate vertices.

To find the correct vertices for each vertex force, I first check the two crack edges leaving the current vertex of the list. If they are both crack edges from the first simulation, I apply the corresponding vertex force to that vertex and continue. Otherwise, I check the first crack edge to determine which simulation generated the crack. If it is from the first simulation, I apply the vertex force to that vertex and all following vertices until I find the next crack edge generated from the first simulation, looping around to the beginning of the vertex list as needed. This method ensures that all vertices are moved in the appropriate direction during post-processing; otherwise, the crack vertices from the second simulation would not be given the correct position.

When the first crack edge is from the second simulation, I do the same process described above but instead traverse the list backwards, looping to the end of the list as needed. This second case can only occur near the beginning of the list, caused by the choice of reference vector splitting the vertices between first and second simulation crack edges. The resulting combined vertex forces gives the information needed for the post-processing

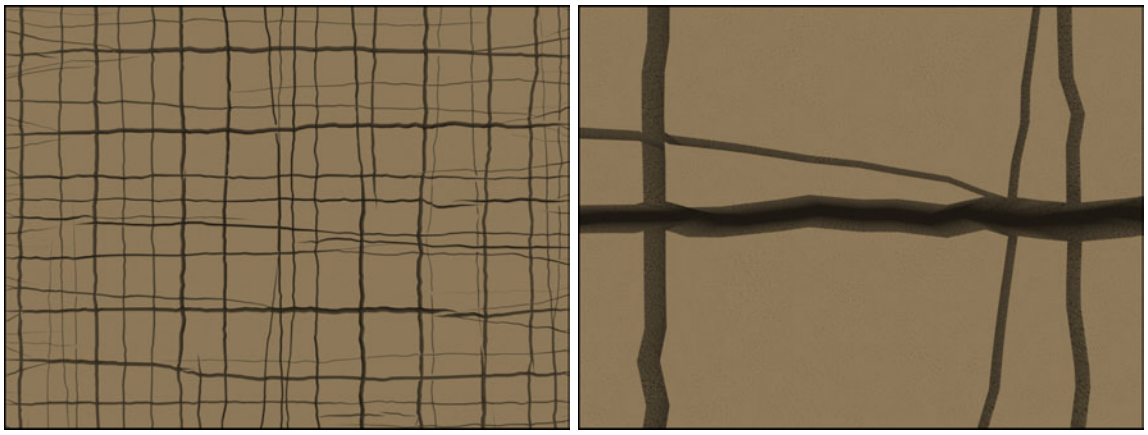


Figure 7.6: The result on the *left* (repeated from Figure 7.5, *right*) shows the cross-cutting of horizontal and vertical joint sets. The image on the *right* gives a close-up view of several junctions near the center of the *left* image. This image demonstrates that the recracking method creates distinct boundaries at the intersections between cracks from different joint sets.

step from Section 6.2 to correctly create gaps between the crack edges for both simulations.

Figure 7.6, *left* shows the result of combining horizontal and vertical joint sets using this method. I give a close-up view of that image in Figure 7.6, *right*, demonstrating that the method creates distinct boundaries between the cracks of different joint sets.

Chapter 8

Results and Discussion

I implemented the method described above in C++ and rendered my 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, I found the simulation times to be reasonable on a Macintosh G5 computer with 2.5 GB of memory. I give these times in the figure captions.

8.1 Results

My method generates crack patterns similar to a variety of cracks found in nature. For example, ceramic glazes often contain cracks caused by a difference in the coefficient of thermal expansion between the pottery and glaze (see Section 1.2.1). Sometimes ceramicsists intentionally create the crack patterns by using a crackle glaze process. My method simulates this effect by uniformly shrinking the surface and evolving the stress field with

Figure 8.1: A rendered example of a crackle glaze cup (*left*) compared to a photograph (*right*). My 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 minutes with a 39,611 triangle input mesh. (Originally presented in Figure 4.2.)

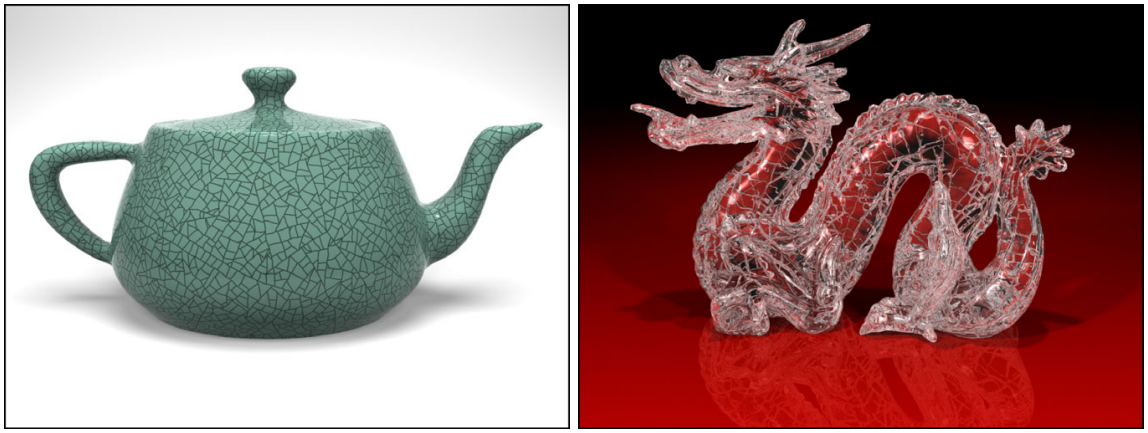


Figure 8.2: *Left:* Example of a crackle glaze teapot, generated by initializing the stress field to uniform shrinkage and evolving it by adding uniform tension. This example required 4.27 hours computation with an input mesh of 60,000 triangles. (Originally presented in Figure 4.3). *Right:* A crackle glass dragon, generated by uniform shrinkage of the surface. This example required 2.17 hours of computation with an input mesh of 75,000 triangles. (Originally presented in Figure 1.1.)

uniform tension. The result of simulating this phenomenon appear in Figure 8.2, *left*. I also provide a comparison of my image with Japanese teacup photograph in Figure 8.1.

Crack patterns also occur in glass for various reasons. When the outer layer of

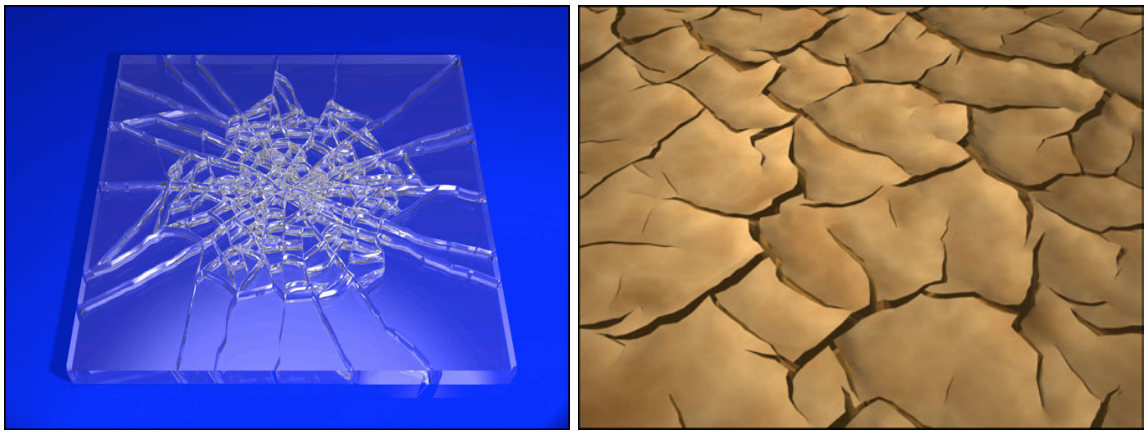


Figure 8.3: *Left:* Cracks generated from initializing the stress field with a pattern modeling low-velocity impacts in flat glass. The field is evolved by a relaxation process, causing the cracks to propagate. Total computation time was 15 seconds with an input mesh of 4,804 triangles. (Originally presented in Figure 4.9.) *Right:* Example of adding a curling effect to the cracked mud from Figure 8.5. This example required 1.5 minutes computation time on a 19,602 triangle input mesh. (Originally presented in Figure 6.7.)

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 crackle glass (see Section 1.2.2). Because this layer is adhered to the inner core of molten glass, the object retains its overall shape. I model this effect by initializing the stress field to uniform shrinkage, as illustrated in Figure 8.2, (*right*). Flat glass also cracks in a particular manner when struck by an object, as described in Section 4.2.3. I allow the user to specify a stress field on the object to model this impact effect and demonstrate my result in Figure 8.3, *left*.

During the mud drying process, cracks form to alleviate the stress that builds on the surface. My method generates similar results by using uniform shrinkage, as demonstrated in the comparison between my results and photographs in Figures 8.4 and 8.5. The simulation generating these results differed in the parameter controlling crack propagation,

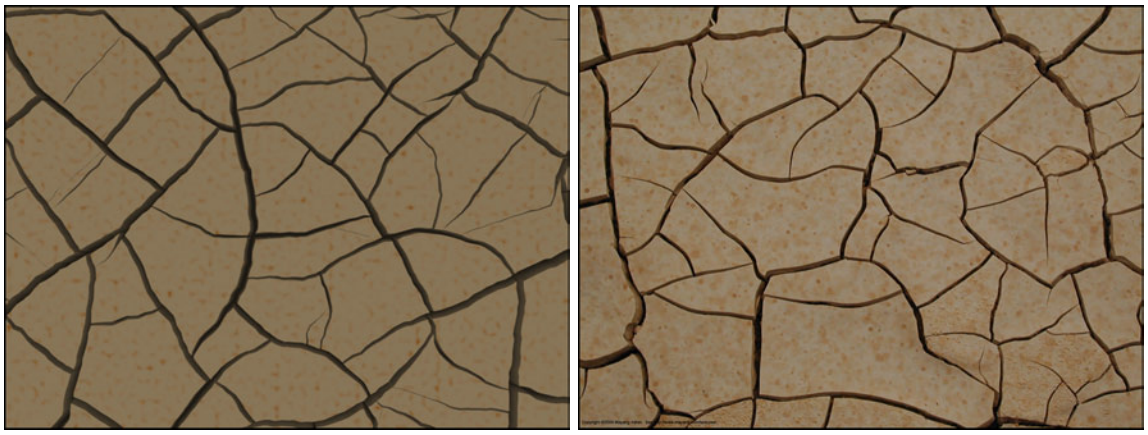


Figure 8.4: A rendered example of dried mud (*left*) compared to a photograph (*right*, copyright 2004 Mayang Adnin). I used uniform shrinkage of the surface and set $\alpha = 0.85$ to propagate the cracks further, requiring 2.05 minutes computation time on a 19,602 triangle input mesh. (Originally presented in Figure 5.4.)

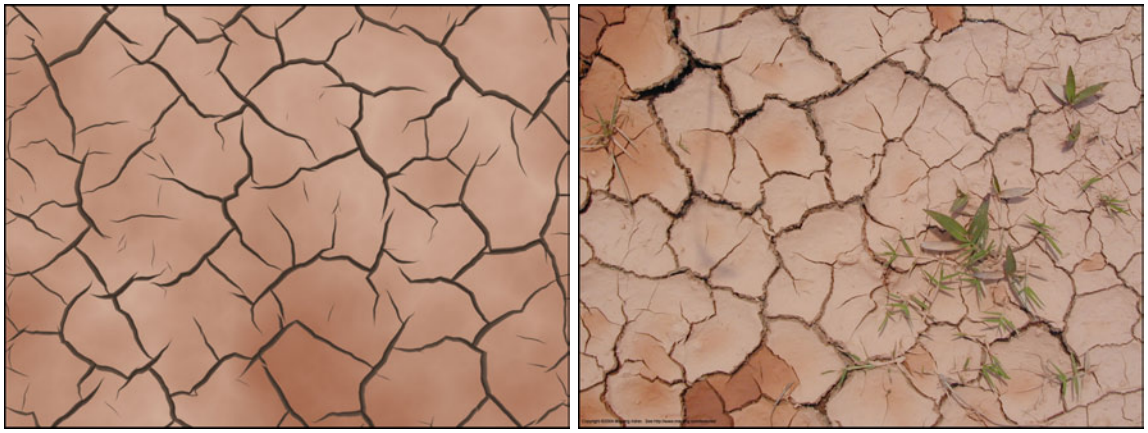


Figure 8.5: A comparison between rendered dried mud (*left*) and a photograph (*right*, copyright 2004 Mayang Adnin). As in Figure 8.4, I used uniform shrinkage of the surface, but changed $\alpha = 0.5$ to demonstrate controlling the crack propagation parameter. Total computation time was 1.5 minutes on a 19,602 triangle input mesh. (Originally presented in Figure 5.5.)

α. Capturing such a mud cracking process with time-lapse photography would be time consuming because drying can be slow. Because physical simulations are temporally coherent, my method is able to generate animations of the cracking process similar to the real

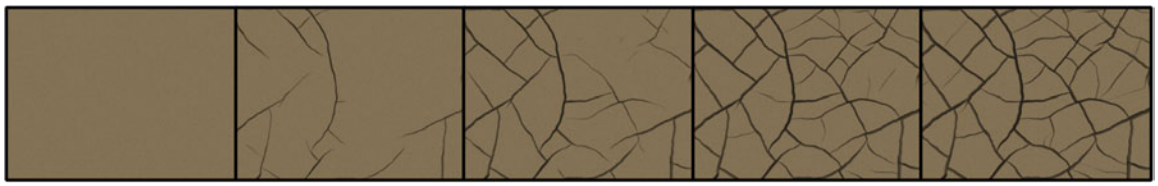


Figure 8.6: An animation of mud drying for Figure 8.4, simulated by setting the stress field to uniform shrinkage of the surface.

world by writing frames while the simulation progresses. I show the results of capturing the cracking process in Figure 8.6. I also created a curled version of Figure 8.5 in Figure 8.3 (*right*). These examples demonstrate the ease of generating different results by varying a few intuitive parameters.

In addition to mud, my method can model certain crack formations found in rock. For example, the crack patterns found the Entrada Sandstone area of Arches National Park, Utah probably formed in two stages (see Section 1.2.4). The crack patterns were likely caused by stress from the strata arching, creating long cracks, followed by bending, forming the second set of cracks. My method allows the stress field to be easily changed during the simulation, enabling me to create a crack pattern such as the one described above. See Figure 8.7, *left* for an example. However, sometimes the second set of crack patterns pass completely through the first set to create a cross-cutting joint pattern. I also am able to model this cracking phenomena by using the two simulation approach described in Chapter 7. The resulting image is shown in Figure 8.7, *right*.

I also model anisotropic materials by using a nonuniform stress field. For example, I model cracks forming along the wood grain by initializing the stress field to lie in the direction perpendicular to the grain. I demonstrate cracks created by this method in

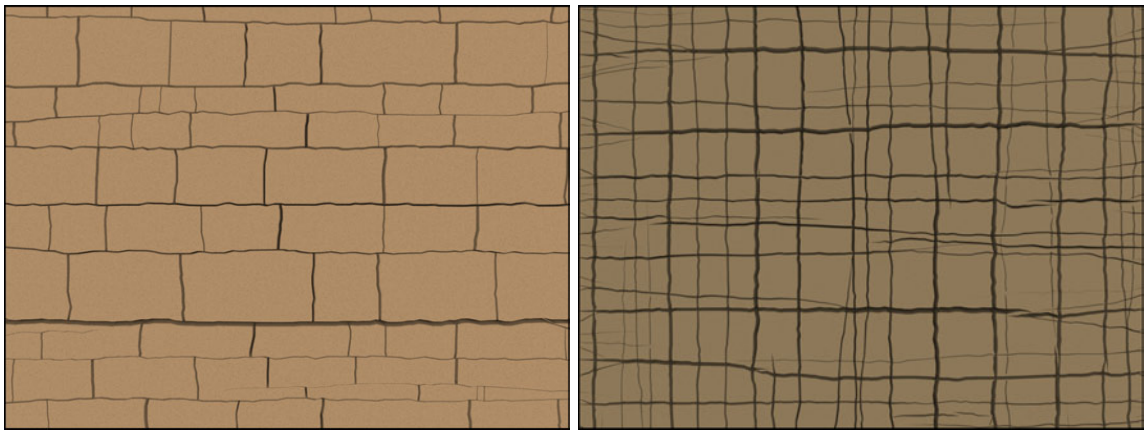


Figure 8.7: *Left*: Example of creating two distinct interacting crack sets by changing the stress pattern during the course of one simulation. I first generated the horizontal cracks by using a vertical stress field. To create new cracks that interact with the first crack set, I reinitialize the mesh to have a horizontal stress field and generate the vertical cracks. Total computation time was 3.1 minutes on a 19,602 element mesh. (Originally presented in Figure 4.7.) *Right*: In contrast, I use multiple simulations to create cross-cutting joint patterns, as found in some rock formations. The first simulation generates the vertical cracks from a horizontal stress field and the second simulation creates vertical cracks from a horizontal stress field. (Originally presented in Figure 7.5.)

Figure 8.8. I can 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 Figure 8.9.

My method can also use the object's geometry to influence crack patterns. For example, the simulation creating Figure 8.10, *left* initialized the stress field to uniform shrinkage and then biased it in the principal curvature directions. This effect is illustrated by the vertical cracks in the left arm and the cracks along the folds of cloth. I can also bias the cracks to form in regions of high curvature by scaling the separation tensor. I use the Frobenius norm of the curvature tensor as a scale factor in Figure 8.11 and the sum of squares of the principal curvature values in Figure 8.10 (*right*), generating a higher concentration of cracks in high curvature regions. I can combine these two methods of

Figure 8.8: An example of cracked wood along the grain (*left*) compared to a photograph (*right*, copyright 2006 Mayang Adnin). I used a vertical stress field to model the anisotropic material. Total computation time was 19.2 minutes on a 79,202 element mesh. (Originally presented in Figure 4.5.)

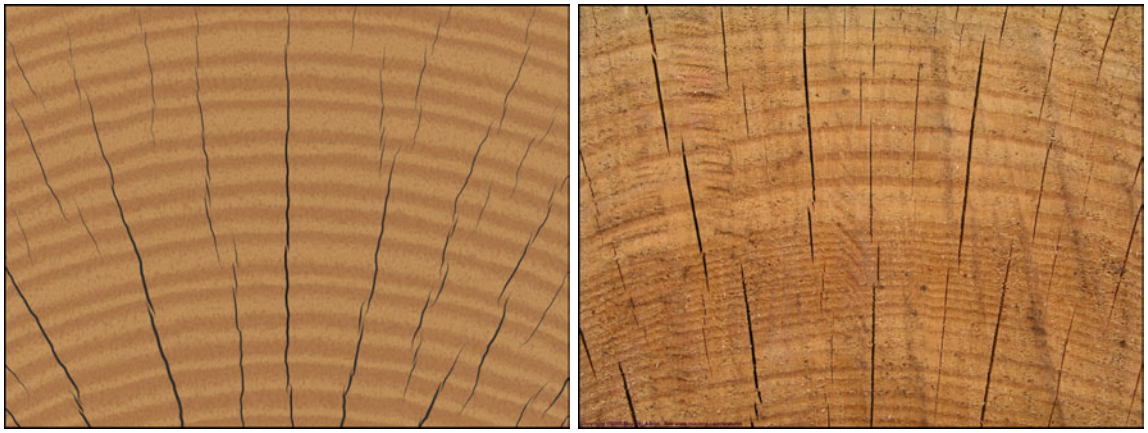


Figure 8.9: I show an example of wood cracking perpendicular to the grain at the sawn end of a tree (*left*) compared to a photograph (*right*, copyright 2005 Mayang Adnin). I used a concentric stress field, radiating outward from the center of the tree, to model the anisotropic stress. Total computation time was 10.1 minutes on a 79,202 element mesh. (Originally presented in Figure 4.6.)

using curvature to model a stress field created from an expanding object as demonstrated by Figure 8.12. See Section 4.2.4 for more details.

The user is also able to arbitrarily set the stress field on the surface. For example,

Figure 8.10: *Left:* As an artistic effect, I 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 hours with an input mesh of 105,772 triangles. (Originally presented in Figure 4.11.) *Right:* 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 hours on a 30,008 element mesh. (Originally presented in Figure 4.14.)

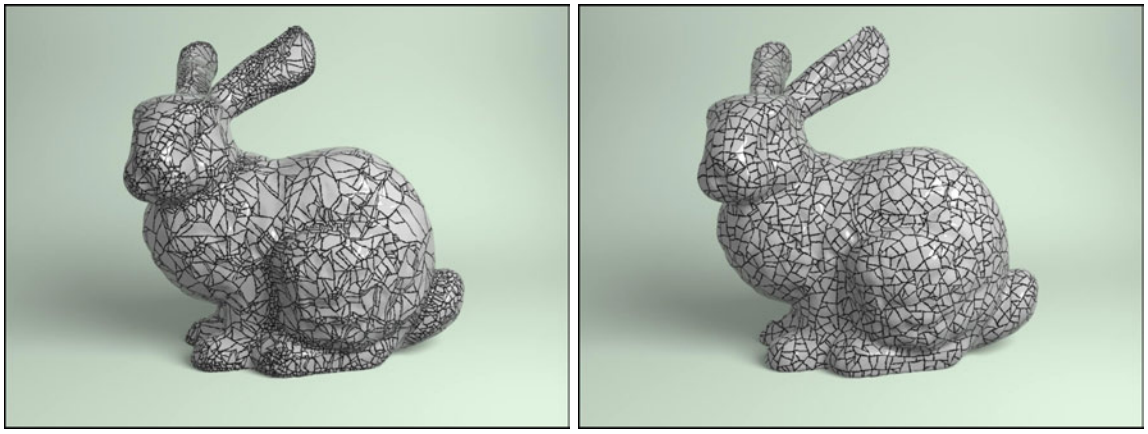


Figure 8.11: Curvature can be used to control crack formation, as demonstrated by the image on the *left* requiring 2.55 hours computation. Notice the high concentration of cracks around areas of high curvature, such as the ears, neck and leg. The *right* image was simulated using only uniform shrinkage of the surface, resulting in more evenly spaced cracks and requiring 2.0 hours. The input mesh size was 51,382 triangles for both examples. (Originally presented in Figure 4.13.)

Figure 8.12: I use curvature to model the stress field and bias the concentration of cracks created by expanding materials. Higher concentrations of cracks are found on the outer part of the handle and the lip of the cup, areas of highest curvature, with less cracks on the body and none inside the cup past the lip. (Originally presented in Figure 4.15, *right*.)

Figure 8.13: I can use an input image (*left*) to define areas with high stress and produce the small scale cracks shown in the center (*right*). I 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. Total computation time was 18.2 minutes on a 72,202 element input mesh. (Originally presented in Figure 4.16.)

I can use a gray-scale input image to define the intensity of the stress field where black has high stress and white has no stress. (See Section 4.2.5 for more details.) I give an example

Parameter	Symbol	Equation	Default	Typical Range
Material toughness threshold	τ	5.2	0.5	0.01 - 0.85
Total number of iterations	N_T	N/A	200	50 - 1400
Cracks created per iteration	η	N/A	3	3 - 15
Relaxation steps per iteration	R	N/A	5	2 - 7
Relaxation rate	Δt	4.1	0.01	0.0001 - 0.02
Initial shrinkage amount	c	4.8	0.1	0.1 - 1.3
Shrinkage rate	c	4.8	0.0	0.0 - 0.25
Crack propagation factor	α	5.3	0.5	0.5 - 0.9
Shrinkage random noise	χ	4.13	0.7	0.1 - 0.7
Stress random noise	ψ	4.14	0.0	0.0 - 0.3

Table 8.1: The primary parameters used by my simulation are listed above. Starting from the left, the first column describes the parameter name, the second column gives the symbolic representation, and the third column lists the equation where it is used (where applicable, see Table 8.2). I provide the default values in the fourth column and the range of values typically used in the last column. Note that the parameters are relative to one another and the scale of the mesh; for example, the simulation requires high shrinkage values to generate cracks when using a large material toughness threshold.

of an input image and the resulting crack pattern in Figure 8.13.

8.2 Exercising Simulation Parameters

The simulation uses a set of parameters to create the general appearance and characteristics of the crack patterns. Some of these parameters are based on physical properties of the material while others are intuitive heuristics. I list these parameters in Table 8.1, giving a brief description, mathematical notation, and equation where I use the parameter (when applicable). Table 8.2 provides these equations for quick reference.

To better understand the effect of these parameters on the simulation results, I generated a series of examples using the same input mesh. For each set of examples, I fix all of the parameters except the one being examined and vary that parameter over a range of typical values. Note that the parameters interact with one another. To highlight the effect

Equation Number	Equation
4.1	$\Delta \boldsymbol{p}_{[n]} = \Delta t \ \boldsymbol{F}_{[n]}$
4.8	$\boldsymbol{\sigma'} = \boldsymbol{\sigma} + c\,\mathbf{I}$
4.13	$\boldsymbol{\sigma'} = \boldsymbol{\sigma} + (c + r_1)\,\mathbf{I}$
4.14	$\boldsymbol{\sigma'} = \begin{bmatrix} \sigma_{xx} & \tau_{xy} + (c + r_2) \\ \tau_{yx} + (c + r_2) & \sigma_{yy} \end{bmatrix}$
5.2	$\mathsf{v}^* = \mathsf{v}^+ - \tau$
5.3	$\boldsymbol{\varsigma'} = \boldsymbol{\varsigma} + \alpha\ \mathbf{m}(\hat{\mathbf{n}}^+)\mathsf{v}^*$

Table 8.2: The above table lists the simulation equations referenced by the set of simulation parameters described in Table 8.1.

of a particular parameter, I tried to choose a set of fixed parameters that would minimize extraneous changes in the simulation results.

The simulation controls the point at which the material fails with the material threshold parameter (τ). In addition to determining when to create cracks, τ is also used to calculate the residual value in Equation (5.2). The simulation results shown in Figure 8.14 illustrate that increasing the threshold causes the cracks to become more irregular and jagged. These results demonstrate that the crack jaggedness is controlled by the material threshold, τ, along with the crack propagation parameter, α (see Figures 8.23 and 8.24 for examples). Note that $\alpha = 0.75$ in Figure 8.14. However, when $\tau = 0.25$, the cracks become jagged because there is not enough residual from the stress field calculation to strongly propagate the crack direction. The user can counter this behavior by either reducing τ, increasing the shrinkage c, or adding a small constant factor to the residual (see Section 5.1). When the material toughness exceeds the stress calculated on the object, no cracks form, as demonstrated by the bottom right result of Figure 8.14.

As described in Section 3.1, the simulation will either run until it can no longer

generate cracks or the user decides to terminate it. The user can directly control the termination point by specifying the total number of iterations the simulation runs, restricting the number of cracks generated. I illustrate the effect of this parameter in Figures 8.15 and 8.16. The longer the simulation runs, the more cracks are created. Note that the number of cracks depends on the interaction between the simulation parameters, including the material toughness threshold (τ) and the shrinkage rate (c). If τ were sufficiently low, the simulation would stop producing cracks sooner than depicted. Similarly, a smaller value for c would give less cracks.

In nature, multiple places in a material will often fail simultaneously instead of sequentially. To model this effect, I have included a parameter for specifying the number of cracks attempted per iteration (η). Figure 8.17 illustrates the effect of increasing this parameter on the results while the number of iterations remains constant. Increasing the parameter leads to more cracks, as did increasing the number of iterations in Figures 8.15 and 8.16. However, some of the cracks formed closer together when increasing η because the cracks may have initiated during the same iteration, leaving less time for relaxation to distribute the stress around the cracks. To see the difference, compare the bottom left images in Figures 8.15 and 8.17. For most of the results in this thesis, the modeled cracks are not so close together. To generated these images, I set the number of cracks created per iteration to three. However, to get the desired spacing of the cracks along the wood grain in Figure 8.8, I increased the η value to 15.

I can also control the general spacing between the cracks with both the number of relaxation steps taken per iteration (R) and the relaxation rate (Δt) because these two

parameters are interrelated. I demonstrate changing R in Figures 8.18 and 8.19 with a simple parameter set, chosen to emphasize the effects of the relaxation step parameter. With no relaxation, the cracks look unrealistic and are very concentrated in one area. (Note that increasing the number of total iterations, N_T, would increase the number of cracks on the mesh as I described above; however, the cracks would maintain similar characteristics.) By increasing the number of relaxation steps, the simulation diffuses stress a larger amount each time relaxation is called, causing the cracks to form further away from one another. However, once the number of relaxation steps has become sufficiently high (Figure 8.19), the crack pattern does not drastically change.

Figure 8.20 shows that increasing the relaxation rate parameter (Δt) also controls the spacing between the cracks. Increasing Δt causes the stress to diffuse more at each relaxation step, increasing the distance between cracks. Setting the relaxation rate too high can cause the simulation to go unstable because of the properties of Euler integration. However, I avoid the instability problems by not exceeding the limit and instead take more relaxation steps (R) when needed.

To give the simulation a starting stress field, I use an initial shrinkage amount parameter (initial c). The effects of increasing this parameter are demonstrated in Figure 8.21. Note that no additional shrinkage amount (c) was added to the simulation to highlight the influence of the initial c value. Because the residual is closely tied to the material toughness threshold and the separation tensor (calculated indirectly from the stress field), changing the initial shrinkage amount influences the jaggedness of the cracks. A low stress field gives low residuals, causing more jagged cracks.

The shrinkage rate parameter (c) added at each iteration of the simulation increases the stress field. Similar to the initial shrinkage amount, a small shrinkage rate gives jagged cracks because of low residuals (see Figure 8.22). However, increasing the parameter leads to interesting patterns with a mixture of jaggedness. In this situation, the simulation has small residual amounts for cracks created at the beginning of the simulation. As the shrinkage rate increases the stress field, the residual may increase depending on the relaxation rate and number of relaxation steps. The resulting effect is straighter cracks later in the simulation as illustrated by the top right image of Figure 8.22. This image demonstrates the dependence between the parameters.

To control the jaggedness of the cracks, I use a crack propagation parameter that controls the amount of residual added from the recently cracked node onto its crack tips. To illustrate the effect of this parameter, I vary the crack propagation factor, α, over its valid range, $[0.0, 1.0]$, while keeping the other parameters constant. I show the results and the associated parameters used to generate them in Figures 8.23 and 8.24. For this parameter set, the cracks are short jagged segments intersecting one another at a variety of angles at $\alpha = 0.0$. As α approaches 1.0, the crack segments become longer and smoother, tending to intersect other cracks at more consistent angles.

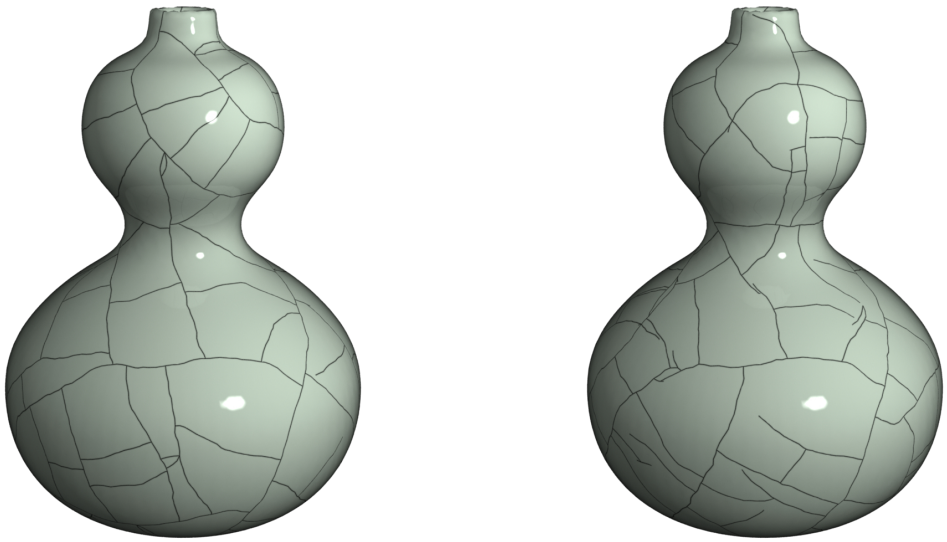
To model material inhomogeneities, I provide two parameters that introduce random noise into the stress field in addition to the randomness introduced by the mesh discretization. One parameter adds noise to the shrinkage amount added to the stress field (χ). Figure 8.25 gives results produced by increasing this parameter. The second parameter adds noise directly to the stress field (ψ) and its effects are shown in Figure 8.26. In both

cases, a small amount of noise changes a small portion of the crack pattern compared to no noise. As the parameter increases, less of the original crack pattern remains in the result. These two noise parameters enable the user to generate multiple crack patterns exhibiting the same characteristics by only changing one number.

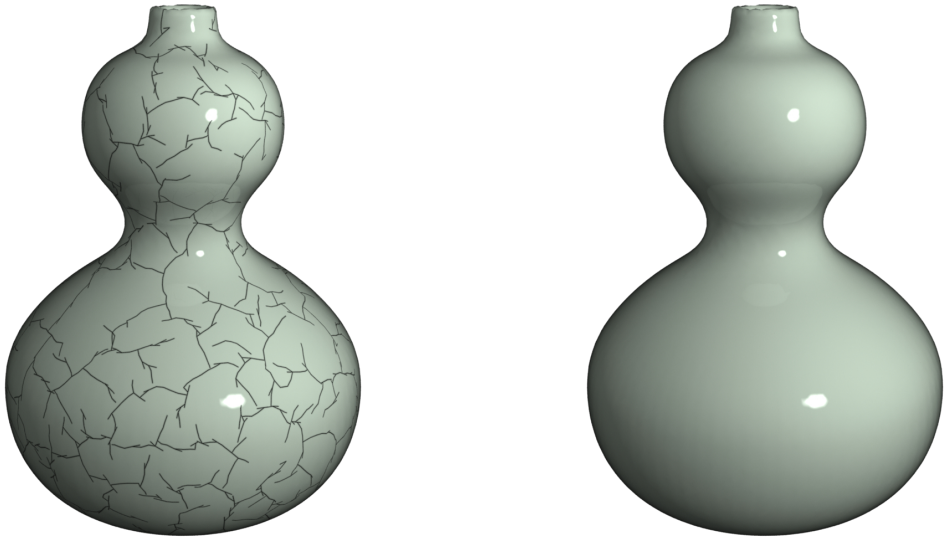
8.3 Summary

The method presented in this thesis generates a variety of visually plausible crack patterns by combining a physically based simulation with traditional appearance driven heuristics. These results include natural processes that occur in mud, rock, and wood, but also from man-made procedures occurring in ceramic glaze and glass. With this approach, I obtain the realism of a physically correct simulation but keep the controllability of a heuristic based method.

I provide the user with a set of parameters controlling the appearance and characteristics of the crack patterns, as discussed in the section above. I have also demonstrated the artistic flexibility of the method with various examples using curvature and an example cracking in a particular stress region. These results show that my approach could possibly be incorporated into an artistic tool, enabling users to paint a stress field on an object to easily generate various cracking results. I compare some of my results, such as the mud, wood, and ceramic glaze, with photographs to demonstrate the realism generated by my method. Although the images are different, the generated crack patterns have qualitatively similar characteristics to the those displayed in the real images.

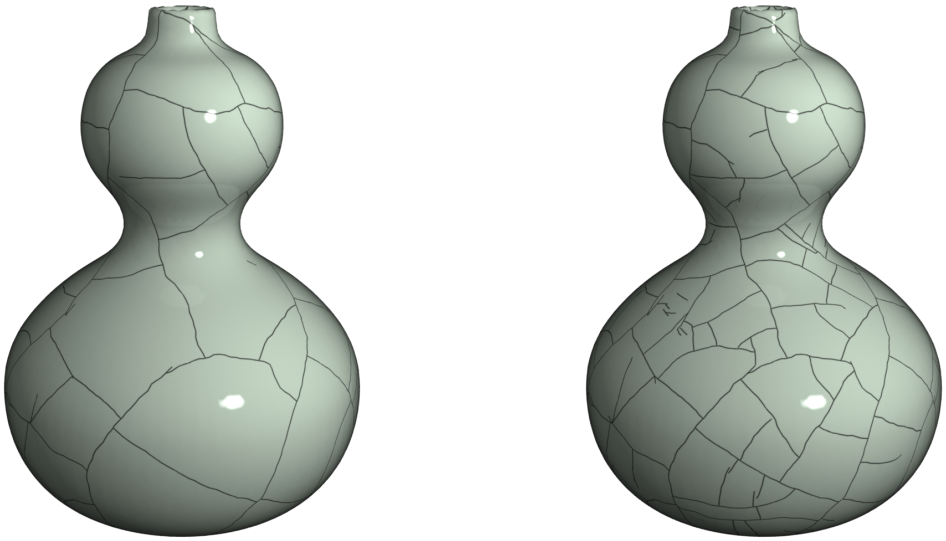


$\tau = 0.05$ $\tau = 0.10$



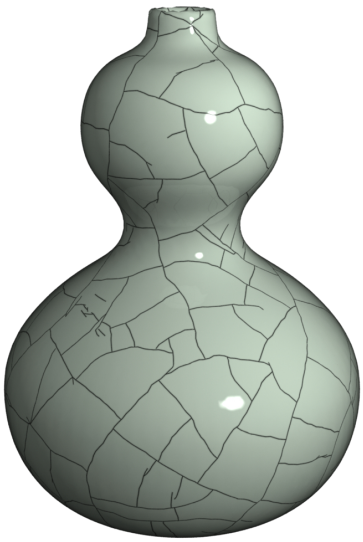
$\tau = 0.25$ $\tau = 0.50$

Figure 8.14: The material toughness threshold (τ) controls the number of cracks in the simulation by specifying the failure point of the material. These examples show how τ changes the jaggedness of the cracks. The other parameters remain the same. $\{N = 150$, $\eta = 3$, $R = 7$, $\Delta t = 0.005$, initial $c = 0.5$, $c = 0.0$, $\chi = 0.0$, $\psi = 0.0, \alpha = 0.75\}$

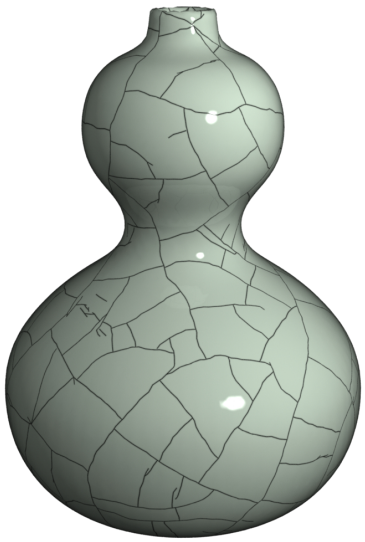


$N_T = 120$

$N_T = 240$

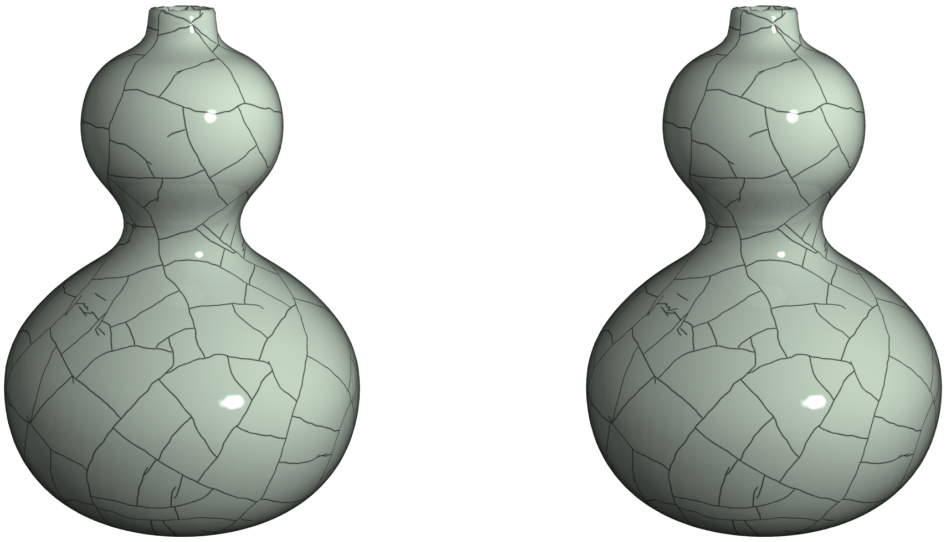


$N_T = 360$



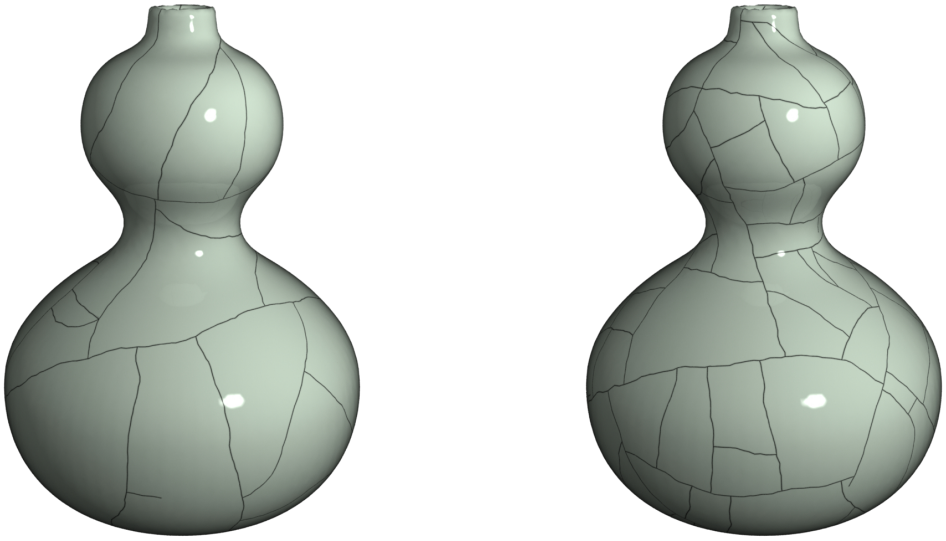
$N_T = 480$

Figure 8.15: This example illustrates the relationship between the number of cracks and the total number of iterations the simulation runs (N_T). (Note that the number of cracks increase only because the stress is high enough to create cracks; if I lowered the material toughness threshold enough, the number of cracks created would change.) All other parameters remain the same for each simulation. $\{\tau = 0.05,\ \eta = 3,\ R = 7,\ \Delta t = 0.009,$ initial $c = 0.15,\ c = 0.001,\ \chi = 0.0,\ \psi = 0.0, \alpha = 0.95\}$

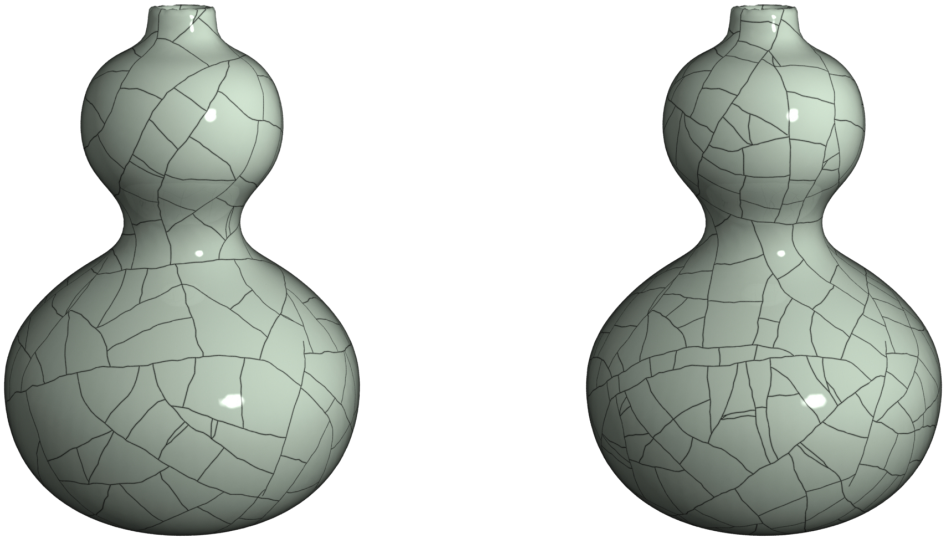


$N_T = 600$ $N_T = 675$

Figure 8.16: Continuation of Figure 8.15. As the number of iterations increases, the number of new cracks decreases to the point where the resulting crack pattern does not drastically change. Note that not many new cracks have formed since $N_T = 360$. All other parameters remain the constant. $\{\tau = 0.05$, $\eta = 3$, $R = 7$, $\Delta t = 0.009$, initial $c = 0.15$, $c = 0.0$, $\chi = 0.0$, $\psi = 0.0, \alpha = 0.95\}$

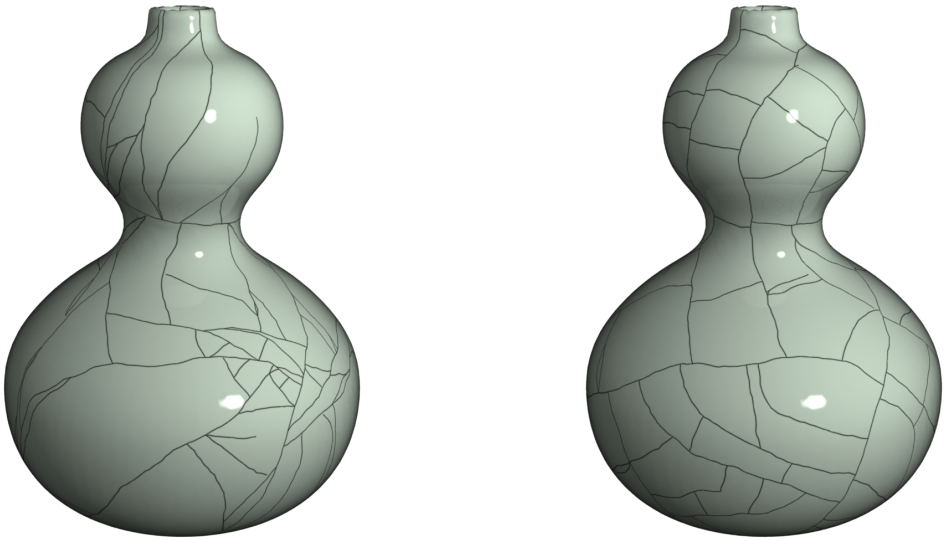


$\eta = 1$ $\eta = 3$

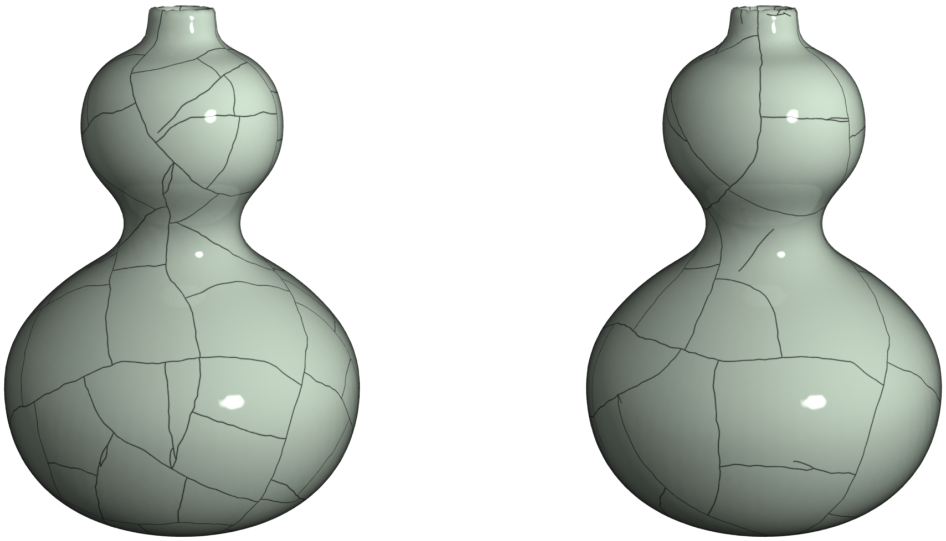


$\eta = 5$ $\eta = 7$

Figure 8.17: These examples were created by increasing the number of cracks created per iteration (η). This parameter effects both the number of cracks created and also their spacing. All other parameters remain fixed. $\{\tau = 0.05,\ N_T = 150,\ R = 7,\ \Delta t = 0.005,$ initial $c = 0.5,\ c = 0.1,\ \chi = 0.0,\ \psi = 0.0, \alpha = 0.75\}$

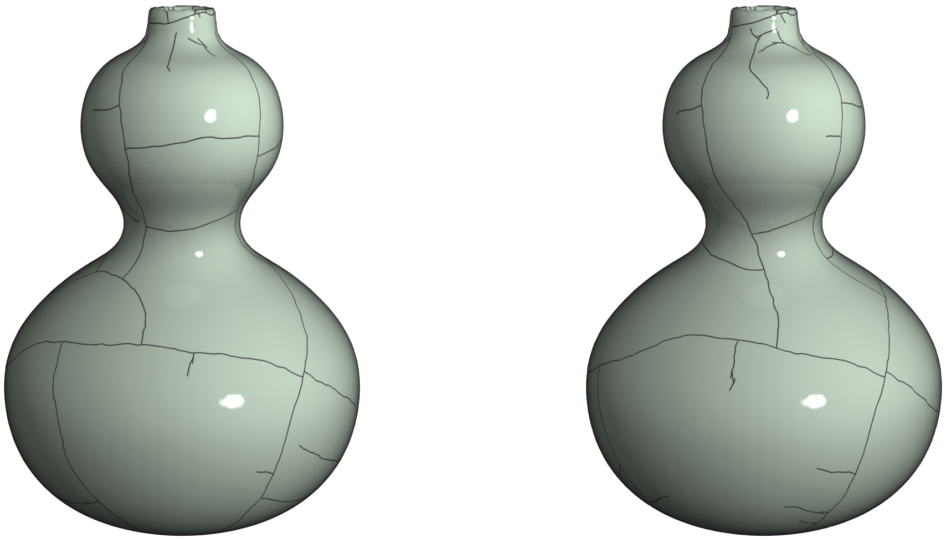


$R = 0$ $R = 5$



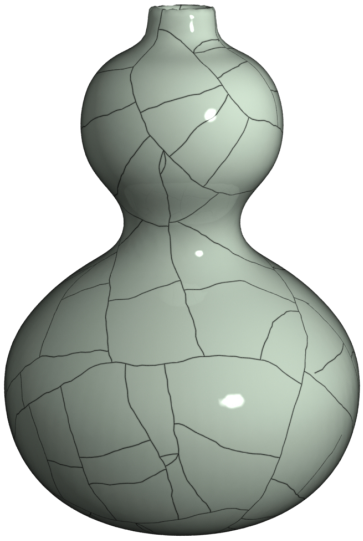
$R = 10$ $R = 50$

Figure 8.18: I vary the number of relaxation steps used per iteration (R) to show its effect on the crack pattern. As R increases, the general spacing between cracks increases and the number of cracks decreases. All other parameters remain the same. $\{\tau = 0.05$, $N_T = 150$, $\eta = 3$, $\Delta t = 0.005$, initial $c = 0.5$, $c = 0.0$, $\chi = 0.0$, $\psi = 0.0, \alpha = 0.75\}$

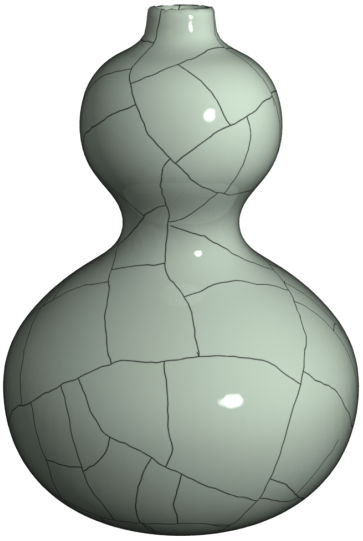


$R = 200$ $R = 400$

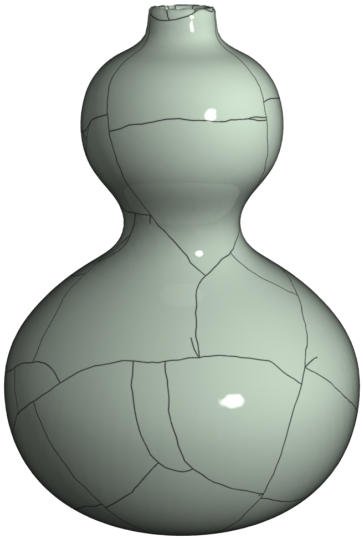
Figure 8.19: Continuation of Figure 8.18. The number of relaxation steps has been increased to the point where it does not drastically change the resulting crack pattern. All other parameters remain the constant. $\{\tau = 0.05,\ N_T = 150,\ \eta = 3,\ \Delta t = 0.005$, initial $c = 0.5,\ c = 0.0,\ \chi = 0.0,\ \psi = 0.0, \alpha = 0.75\}$



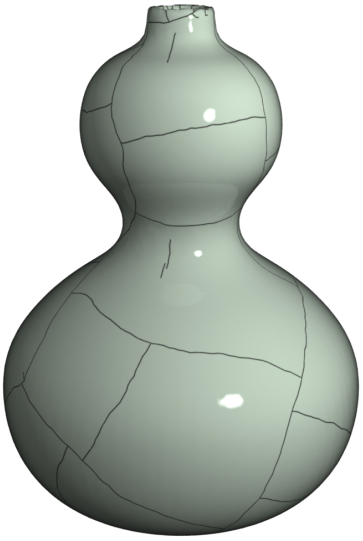
$\Delta T = 0.005$



$\Delta T = 0.009$

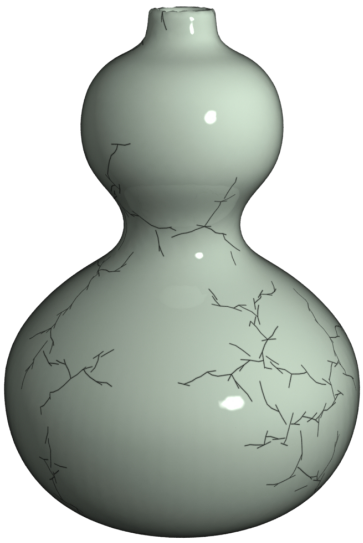


$\Delta T = 0.05$

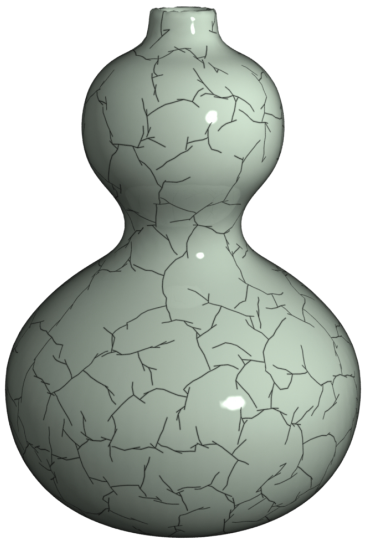


$\Delta T = 0.09$

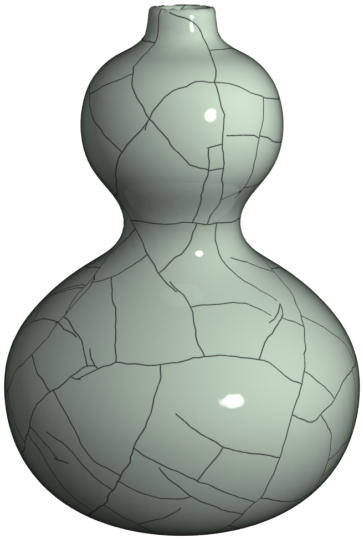
Figure 8.20: The relaxation rate (Δt) parameter also controls the spacing between the cracks, as demonstrated above. All other parameters remain fixed. $\{\tau = 0.05,\ N_T = 150,\ \eta = 3,\ R = 7,$ initial $c = 0.5,\ c = 0.0,\ \chi = 0.0,\ \psi = 0.0, \alpha = 0.75\}$



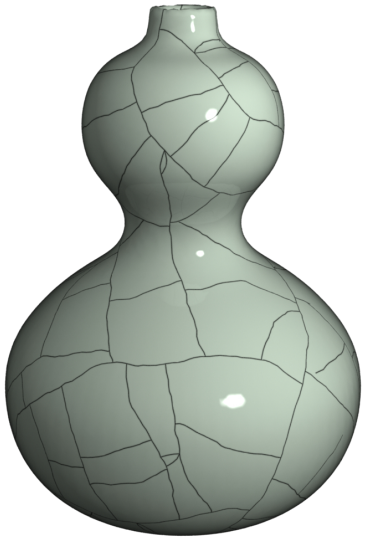
initial $c = 0.08$



initial $c = 0.10$

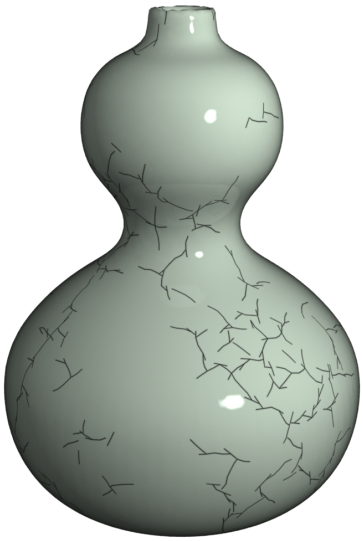


initial $c = 0.25$

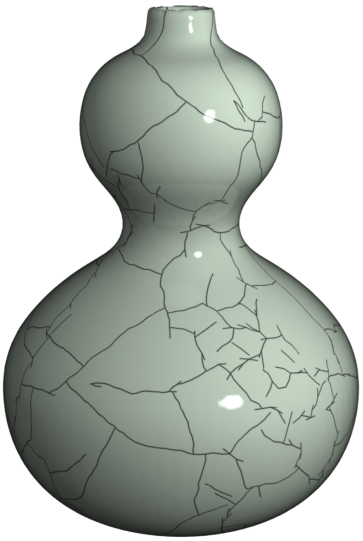


initial $c = 0.50$

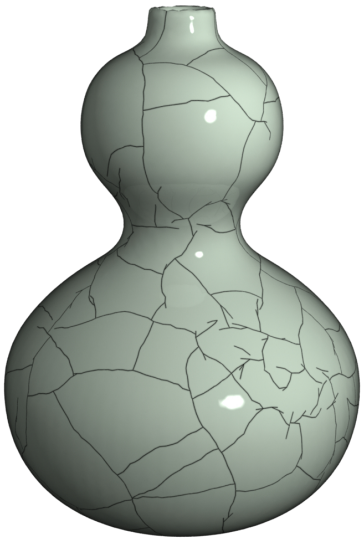
Figure 8.21: I use an inital shrinkage amount (initial c) to define some amount of stress present at the beginning of the simulation. These examples show how increasing this initial amount changes the simulation (note that there is no additional shrinkage being added). The other parameters remain the same for each simulation. $\{\tau = 0.05,\ N = 150,\ \eta = 3,\ R = 7,\ \Delta t = 0.005,\ c = 0.0,\ \chi = 0.0,\ \psi = 0.0, \alpha = 0.75\}$



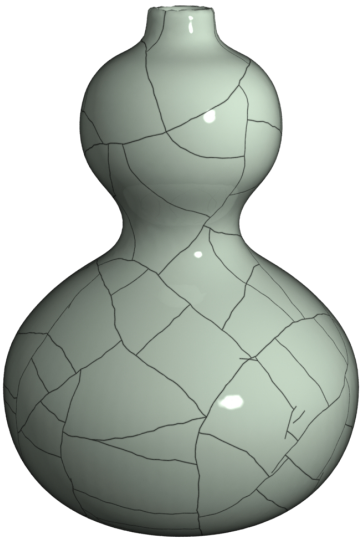
$c = 0.0015$



$c = 0.005$

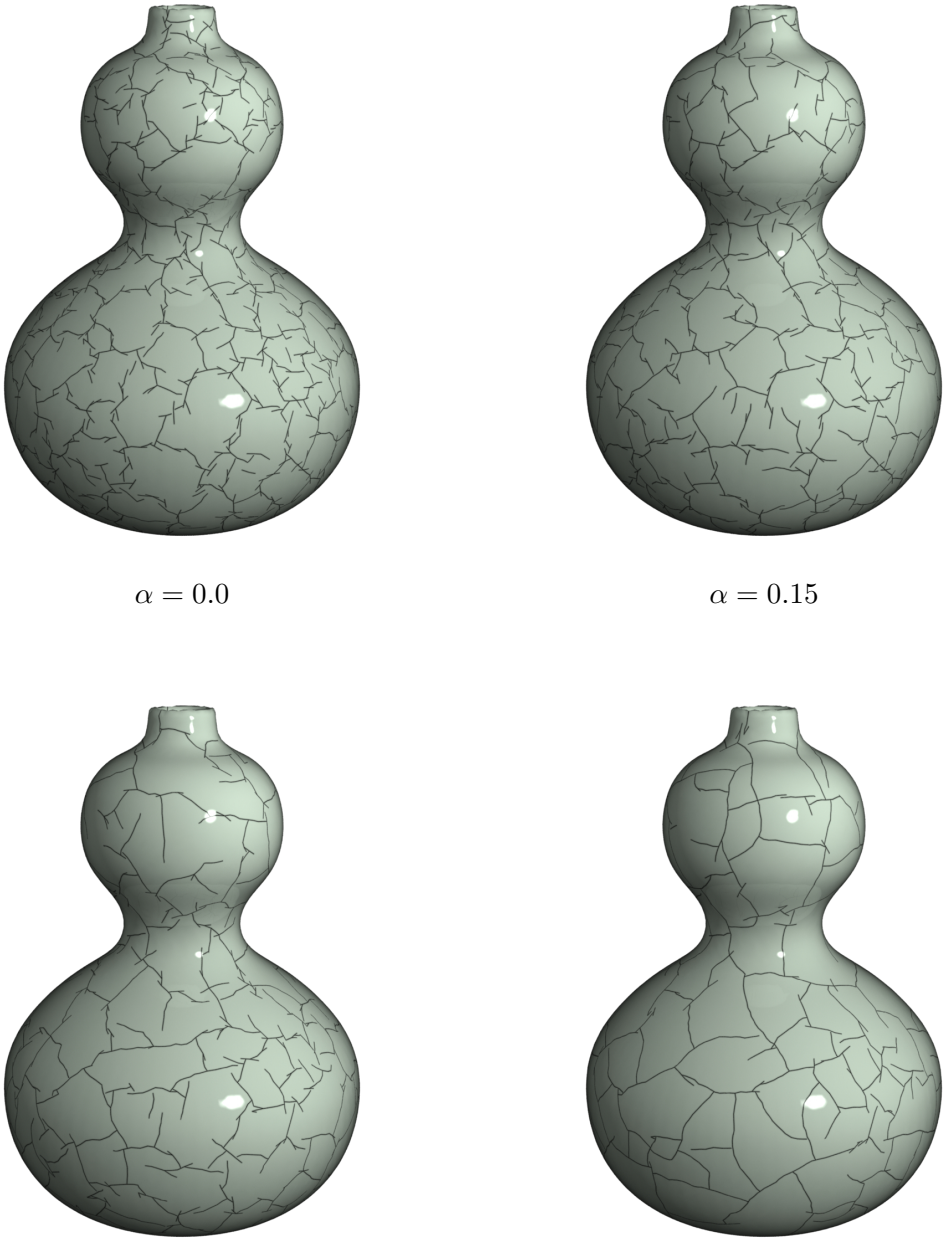


$c = 0.01$



$c = 0.10$

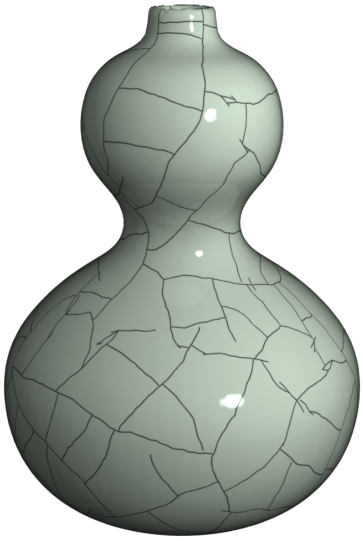
Figure 8.22: I use an shrinkage rate (c) to define the stress added during the simulation. These examples show how increasing this amount changes the simulation. The other parameters remain the same for each simulation. $\{\tau = 0.05,\ N = 150,\ \eta = 3,\ R = 7,\ \Delta t = 0.005,$ initial $c = 0.0,\ \chi = 0.0,\ \psi = 0.0, \alpha = 0.75\}$



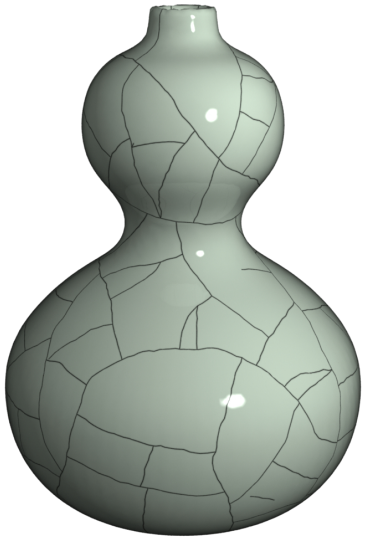
$\alpha = 0.0$ $\alpha = 0.15$

$\alpha = 0.30$ $\alpha = 0.45$

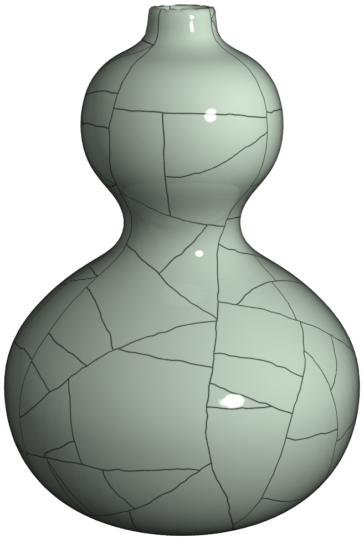
Figure 8.23: In these examples, I change only the crack propagation parameter, α, while keeping all other parameters constant. These crack patterns show that increasing α from 0.0 (no crack propagation) to 0.45 decreases the jaggedness of the crack pattern. $\{\tau = 0.05$, $N_T = 150$, $\eta = 3$, $R = 7$, $\Delta t = 0.005$, initial $c = 0.5$, $c = 0.1$, $\chi = 0.1$, $\psi = 0.11\}$



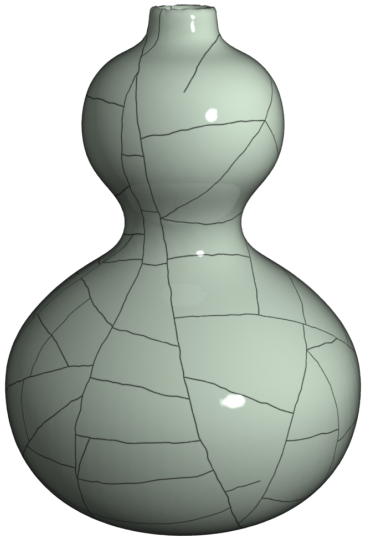
$\alpha = 0.60$



$\alpha = 0.75$

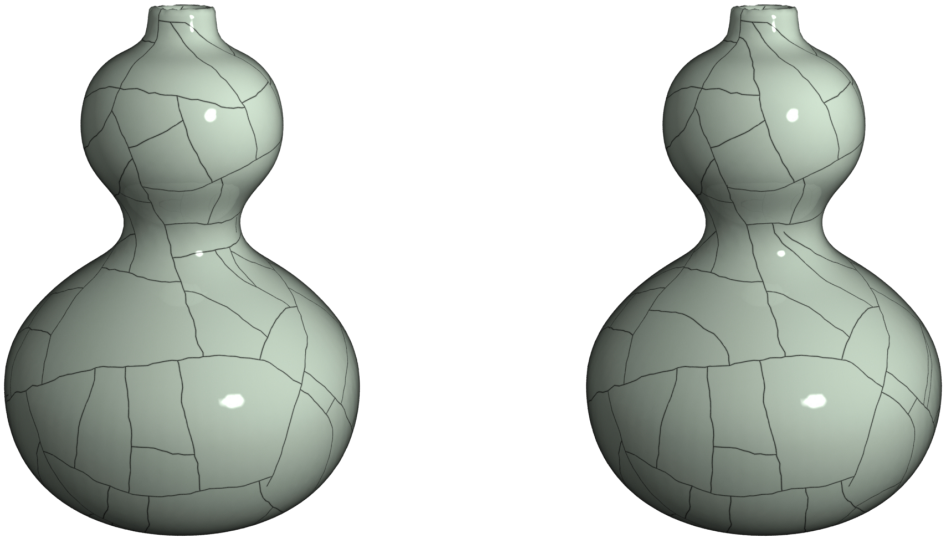


$\alpha = 0.90$

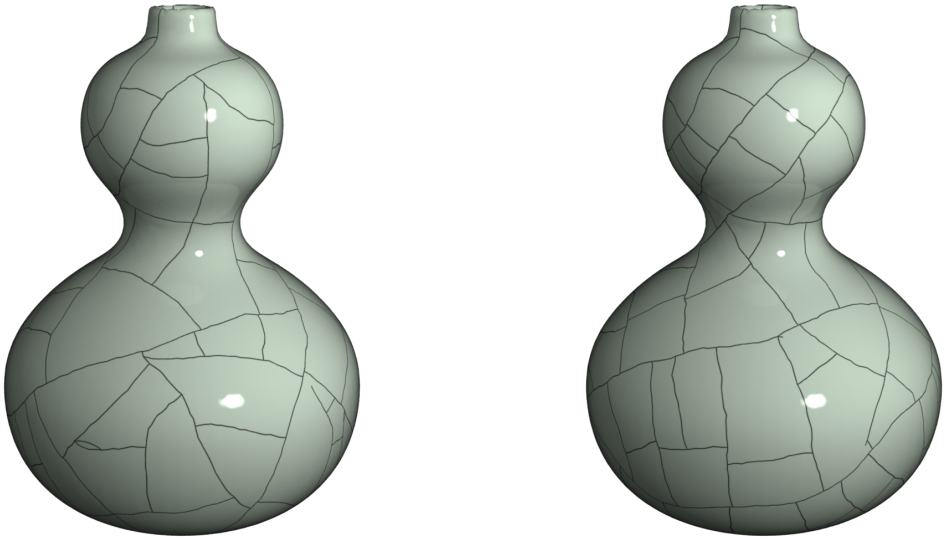


$\alpha = 1.00$

Figure 8.24: Continuation of Figure 8.23. I created these results by only changing the crack propagation parameter, α, from 0.60 to 1.00 (full crack propagation). Note that the cracks become longer as the parameter increases. $\{\tau = 0.05$, $N_T = 150$, $\eta = 3$, $R = 7$, $\Delta t = 0.005$, initial $c = 0.5$, $c = 0.1$, $\chi = 0.1$, $\psi = 0.11\}$

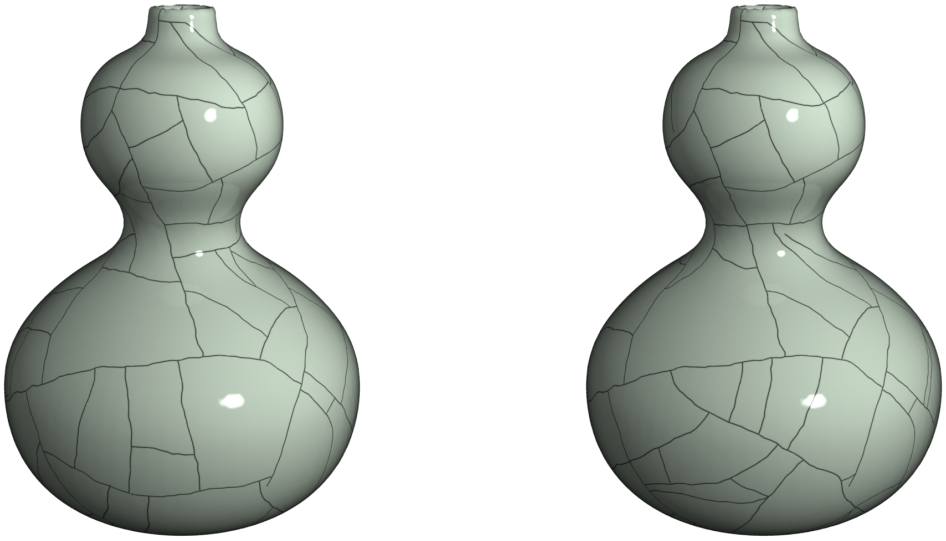


$\chi = 0.0$ $\chi = 0.01$

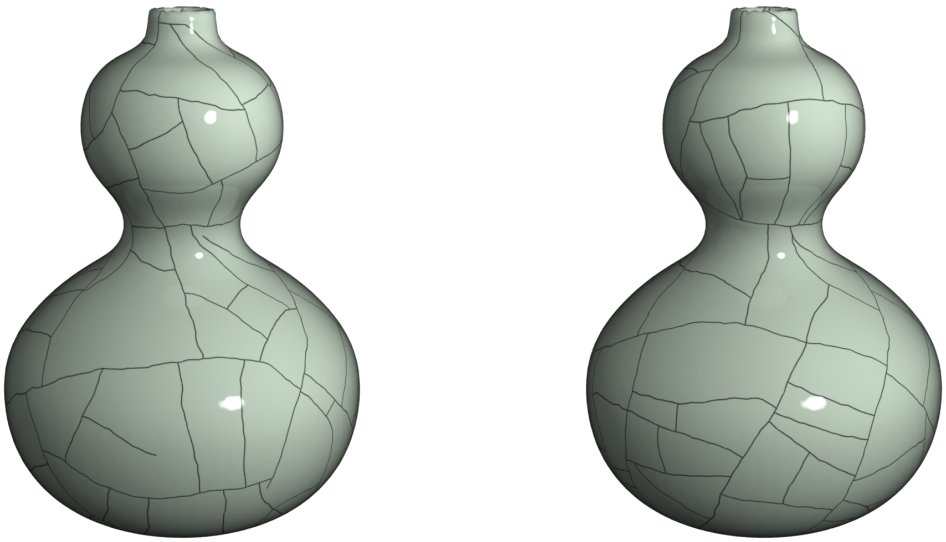


$\chi = 0.05$ $\chi = 0.10$

Figure 8.25: I modify a shrinkage random noise parameter (χ) to change the stress field for different simulations. Increasing the noise parameter increases the difference in the pattern from $\chi = 0.0$ (no noise). For example, note the differences in the crack pattern near the top of the vase between $\chi = 0.0$ and $\chi = 0.01$. The other parameters remain the same. $\{\tau = 0.05$, $N = 150$, $\eta = 3$, $R = 7$, $\Delta t = 0.005$, initial $c = 0.0$, $c = 0.1$, $\psi = 0.0, \alpha = 0.75\}$



$\psi = 0.0$ $\psi = 0.001$



$\psi = 0.005$ $\psi = 0.01$

Figure 8.26: To add randomness to the stress field, I change a stress random noise parameter (ψ). Each example uses a larger random noise parameter than the previous, giving increasing variability. For example, the crack pattern near the top of the vase is the same between $\psi = 0.0$ and $\psi = 0.001$, but the bottom cracks have changed. The other parameters remain the same. $\{\tau = 0.05,\ N = 150,\ \eta = 3,\ R = 7,\ \Delta t = 0.005$, initial $c = 0.0,\ c = 0.1,\ \chi = 0.0, \alpha = 0.75\}$

Bibliography

[1] Pierre Alliez, David Cohen-Steiner, Olivier Devillers, Bruno Lévy, and Mathieu Desbrun. Anisotropic polygonal remeshing. In *Proceedings of ACM SIGGRAPH 2003*, pages 485–493, July 2003.

[2] T. L. Anderson. *Fracture Mechanics: Fundamentals and Applications*. CRC Press, third edition, 2004.

[3] Kimiya Aoki, Ngo Hai Dong, Toyohisa Kaneko, and Shigeru Kuriyama. Physically based simulation of cracks on drying 3d solid. In *10th Pacific Conference on Computer Graphics and Applications*, page 467, 2002.

[4] P. Areias and T. Belytschko. Analysis of three-dimensional crack initiation and propagation using the extended finite element method. *International Journal for Numerical Methods in Engineering*, 63:760–788, 2005.

[5] Zhaosheng Bao, Jeong-Mo Hong, Joseph Teran, and Ronald Fedkiw. Fracturing rigid materials. *IEEE Transactions on Visualization and Computer Graphics*, 13(2):370–378, 2007.

[6] Jean-Louis Blés and Bernard Feuga. *The Fracture of Rocks*. North Oxford Academic Publishers, 1986.

[7] S. Bohn, L. Pauchard, and Y. Couder. Hierarchical crack patterns as formed by successive domain divisions. I. Temporal and geometrical hierarchy. *Physical Review E*, 75, Apr 2005.

[8] S. Bohn, J. Platkiewicz, B. Andreotti, M. Adda-Bedia, and Y. Couder. Hierarchical crack patterns as formed by successive domain divisions. II. From disordered to deterministic behavior. *Physical Review E*, 75, Apr 2005.

[9] Stephen J. Bourne, Franz Brauckmann, Lex Rijkels, Ben J. Stephenson, Alex Webber, and Emanuel J.M. Willemse. Predictive modelling of naturally fractured reservoirs using geomechanics and flow simulation. 2000.

[10] S. N. Chernyshev and W. R. Dearman. *Rock Fractures*. Butterworth-Heinemann, 1991.

[11] Fehmi Cirak, Michael Ortiz, and Anna Pandolfi. A cohesive approach to thin-shell fracture and fragmentation. *Computer Methods in Applied Mechanics and Engineering*, 194:2604–2618, 2005.

[12] David Cohen-Steiner and Jean-Marie Morvan. Restricted delaunay triangulations and normal cycle. In *Proceedings of the 19th Annual ACM Symposium on Computational Geometry*, pages 312–321, 2003.

[13] Committee on Fracture Characterization and Fluid Flow, National Research Council. *Rock Fractures and Fluid Flow: Contemporary Understanding and Applications*. National Academy Press, 1996.

[14] Kenneth M. Cruikshank and Atilla Aydin. Unweaving the joints in entrada sandstone, arches national park, utah, u.s.a. *Journal of Structural Geology*, 17(3):409–421, 1995.

[15] Brett Desbenoit, Eric Galin, and Samir Akkouche. Modeling cracks and fractures. *The Visual Computer*, 21(8–10):717–726, Sep 2005.

[16] Pavol Federl and Przemyslaw Prusinkiewicz. A texture model for cracked surfaces, with an application to tree bark. In *Proceedings of the 7th Western Computer Graphics Symposium*, pages 23–29, 1996.

[17] Pavol Federl and Przemyslaw 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.

[18] Pavol Federl and Przemyslaw Prusinkiewicz. Finite element model of fracture formation on growing surfaces. *Lecture Notes in Computer Science*, 3037:138–145, Jan 2004.

[19] Y. C. Fung. *A First Course in Continuum Mechanics*. Prentice Hall, third edition, 1994.

[20] P.A. Gillespie, J.J. Walsh, J. Watterson, C.G. Bonson, and T. Manzocchi. Scaling relationships of joint and vein arrays from The Burren, Co. Clare, Ireland. *Journal of Structural Geology*, 23:183–201, 2001.

[21] Yotam Gingold, Adrian Secord, Jefferson Y. Han, Eitan Grinspun, and Denis Zorin. A discrete model for inelastic deformation of thin shells. Technical report, Media Research Lab, New York University, August 2004.

[22] Stéphane Gobron and Norishige Chiba. Crack pattern simulation based on 3d surface cellular automaton. In *Proceedings of the International Conference on Computer Graphics*, pages 153–162, 2000.

[23] Stéphane Gobron and Norishige Chiba. Simulation of peeling using 3d-surface cellular automata. In *Proceedings of the 9th Pacific Conference on Computer Graphics and Applications*, pages 338–347, 2001.

[24] Xiaohu Guo, Xin Li, Yunfan Bao, Xianfeng Gu, and Hong Qin. Meshless thin-shell simulation based on global conformal parameterization. *IEEE Transactions on Visualization and Computer Graphics*, 12(3):375–385, May/June 2006.

[25] Robert M. Haralick and Linda G. Shapiro. *Computer and Robot Vision*, volume 1. Addison-Wesley Publishing Company, 1992. Chapter 5.

[26] Koichi Hirota, Yasuyuki Tanoue, and Toyohisa Kaneko. Generation of crack patterns with a physical model. *The Visual Computer*, 14(3):126–137, 1998.

[27] Koichi Hirota, Yasuyuki Tanoue, and Toyohisa Kaneko. Simulation of three-dimensional cracks. *The Visual Computer*, 16:371–378, Nov 2000.

[28] G.W. Horgan and I.M. Young. An empirical stochastic model for the geometry of two-dimensional crack growth in soil (with discussion). *Geoderma*, 96:263–276, 2000.

[29] Hsien-Hsi Hsieh, Wen-Kai Tai, Cheng-Chin Chiang, and Mau-Tsuen Yang. Flexible and interactive crack-like patterns presentation on 3d objects. *Lecture Notes in Computer Science*, 3280:90–99, Jan 2004.

[30] Olive R. Jones and Catherine Sullivan. *The Parks Canada Glass Glossery for the description of containers, tableware, flat glass, and closures.* Minister of the Environment, 1989.

[31] So Kitsunezaki. Fracture patterns induced by dessication in a thin layer. *Physical Review E*, 60(6), Dec 1999.

[32] Pierre L'Ecuyer. Random numbers for simulation. *Communications of the ACM*, 33(10):85–97, 1990.

[33] John C. Lorenz and Scott P. Cooper. Interpreting fracture patterns in sandstones interbedded with ductile strate at the Salt Valley Anticline, Arches National Park, Utah. Technical Report SAND2001-3517, Sandia National Laboratories, November 2001.

[34] Jianye Lu, Athinodoros S. Georghiades, Andreas Glaser, Hongzhi Wu, Li-Yi Wei, Baining Guo, Julie Dorsey, and Holly Rushmeier. Context-aware textures. *ACM Transactions on Graphics*, 26(1), 2007. Article 3.

[35] Donal T. MacVeigh. Emulation of uniform cracking. In *Compugraphics '95*, pages 218–227, 1995.

[36] Aurelien Martinet, Eric Galin, Brett Desbenoit, and Samir Akkouche. Procedural modeling of cracks and fractures. In *International Conference on Shape Modeling and Applications*, pages 346–349, 2004.

[37] W. D. Means. *Stress and Strain: Basic Concepts of Continuum Mechanics for Geologists.* Springer-Verlag, 1976.

[38] Neil Molino, Zhaosheng Bao, and Ron Fedkiw. A virtual node algorithm for changing mesh topology during simulation. *ACM Transactions on Graphics*, 23(3):385–392, 2004.

[39] David Mould. Image-guided fracture. In *Proceedings of Graphics Interface*, pages 219–226, 2005.

[40] Matthias Müller and Markus Gross. Interactive virtual materials. In *Proceedings of Graphics Interface*, pages 239–246, 2004.

[41] Matthias Müller, Matthias Teschner, and Markus Gross. Physically-based simulation of objects represented by surface meshes. In *Proceedings of the Computer Graphics International*, pages 26–33, 2004.

[42] T. Nishioka. Computational dynamic fracture mechanics. *International Journal of Fracture*, 86:127–159, 1997.

[43] Alan Norton, Greg Turk, Bob Bacon, John Gerth, and Paula Sweeney. Animation of fracture by physical modeling. *The Visual Computer*, 7(4):210–219, 1991.

[44] James F. O'Brien. *Graphical modeling and animation of fracture.* PhD thesis, Georgia Institute of Technology, Aug 2000.

[45] James F. O'Brien, Adam Bargteil, and Jessica Hodgins. Graphical modeling and animation of ductile fracture. In *Proceedings of ACM SIGGRAPH 2002*, pages 291–294, Aug 2002.

[46] James F. O'Brien and Jessica Hodgins. Graphical modeling and animation of brittle fracture. In *Proceedings of ACM SIGGRAPH 1999*, pages 137–146, Aug 1999.

[47] Jon Olson and David D. Pollard. Inferring paleostresses from natural fracture patterns: A new method. *Geology*, 17:345–348, April 1989.

[48] Eric Paquette, Pierre Poulin, and George Drettakis. The simulation of paint cracking and peeling. In *Proceedings of Graphics Interface*, pages 59–68, May 2002.

[49] Stephen K. Park and Keith W. Miller. Random number generators: good ones are hard to find. *Communications of the ACM*, 31(10):1192–1201, Oct 1988.

[50] Mark Pauly, Richard Keiser, Bart Adams, Philip Dutré, Markus Gross, and Leonidas J. Guibas. Meshless animation of fracturing solids. In *Proceedings of the ACM SIGGRAPH 2005*, pages 957–964, July 2005.

[51] Edith Perrier, Christian Mullon, Michel Rieu, and Ghislain de Marsily. Computer construction of fractal soil structures: Simulation of their hydraulic and shrinkage properties. *Water Resources Research*, 31:2927–2944, 1995.

[52] David D. Pollard and Atilla Aydin. Progress in understanding jointing over the past century. *Geological Society of America Bulletin*, 100:1181–1204, Aug. 1988.

[53] Stephanie J. Post. Porcelain and glaze manufacturing technology, changing aesthetic ideologies and their relationship to the potter's pursuit of the celadon glaze in post-18th century england and france. Master's thesis, Bard Graduate Center for Studies in the Decorative Arts, 2005.

[54] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C*. Cambridge University Press, second edition, 1994.

[55] T. Rabczuk, P. M. A. Areias, and T. Belytschko. A meshfree thin shell method for non-linear dynamic fracture. *International Journal for Numerical Methods in Engineering*, 72(5):524–548, 2007.

[56] Scientific Working Group for Materials Analysis (SWGMAT). Glass fractures. *Forensic Science Communications*, 7(1), Jan 2005.

[57] Kelly Shorlin, John de Bruyn, Malcolm Graham, and Stephen Morris. Development and geometry of isotropic and directional shrinkage-crack patterns. *Physical Review E*, 61(6), June 2000.

[58] A. T. Skjeltorp and P. Meakin. Fracture in microsphere monolayers studied by experiment and computer simulation. *Nature*, 335:424–426, Sept. 1988.

[59] J. R. Taylor and A. C. Bull. *Ceramics Glaze Technology*. Pergamon Press, 1986.

[60] Demetri Terzopoulos and Kurt Fleischer. Modeling inelastic deformation: Viscoelasticity, plasticity, fracture. In *Computer Graphics (SIGGRAPH '88 Proceedings)*, volume 22, pages 269–278, Aug 1988.

[61] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically deformable models. In *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 205–214, July 1987.

[62] Andrew L. Thomas. Poly3d: A three-dimensional, polygonal element, displacement

discontinuity boundary element computer program with applications to fractures, faults and cavities in the earth's crust. Master's thesis, Department of Geology, Stanford University, 1993.

[63] Greg Turk. Re-tiling polygonal surfaces. In *Computer Graphics (SIGGRAPH '92 Proceedings)*, pages 55–64, 1992.

[64] Gilles Valette, Stéphanie Prévost, and Laurent Lucas. A generalized cracks simulation on 3d-meshes. In *Proceedings of the Eurographics Workshop on Natural Phenomena*, pages 7–14, Sept 2006.

[65] B. Velde. Structure of surface cracks in soil and muds. *Geoderma*, 93:101–124, November 1999.

[66] H.-J. Vogel, H. Hoffmann, A. Leopold, and K. Roth. Studies of crack dynamics in clay soil: II. A physically based model for crack formation. *Geoderma*, 125(3-4):2013–223, April 2005.

[67] H.-J. Vogel, H. Hoffmann, and K. Roth. Studies of crack dynamics in clay soil: I. Experimental methods, results, and morphological quantification. *Geoderma*, 125(3-4):203–211, April 2005.

[68] Stan Weitman and Arlene Weitman. *Crackle Glass: Identification and Value Guide*. Collector Books, 2004.

[69] Martin Wicke, Denis Steinemann, and Markus Gross. Efficient animation of point-sampled thin shells. In *Eurographics*, volume 24, 2005.

[70] Brian Wyvill, Kees van Overveld, and Sheelagh Carpendale. Rendering cracks in batik. In *Proceedings of the 3rd International Symposium on Non-photorealistic Animation and Rendering*, pages 61–69, 2004.

[71] Nan Zhang, Xiangmin Zhou, Desong Sha, Xiaoru Yuan, Kumar Tamma, and Baoquan Chen. Integrating mesh and meshfree methods for physics-based fracture and debris cloud simulation. In *Proceedings of the 3rd Eurographics Symposium on Point-Based Graphics*, pages 145–154, July 2006.