Graphical Modeling and Animation of Fracture

A Thesis
Presented to
The Academic Faculty

by

James F. O'Brien

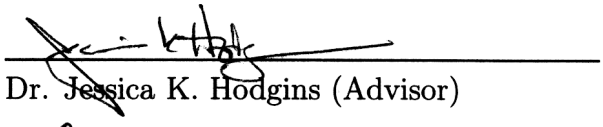
In Partial Fulfillment
of the Requirements for the Degree of
Doctor of Philosophy in Computer Science

Georgia Institute of Technology
July 2000

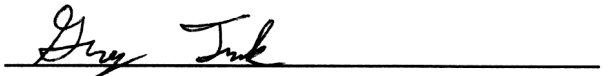
Copyright © 2000 by James F. O'Brien

Graphical Modeling and Animation of Fracture

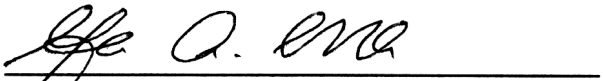
Approved:



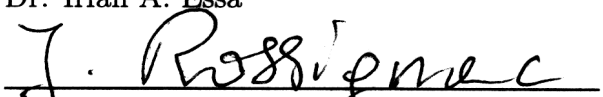
Dr. Jessica K. Hodgins (Advisor)



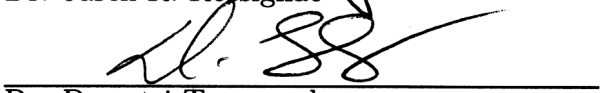
Dr. Greg Turk



Dr. Irfan A. Essa



Dr. Jarek R. Rossignac



Dr. Demetri Terzopoulos
University of Toronto

Date Approved 7/17/2000

Acknowledgments

I would like to thank my family, friends, and colleagues who provided me with their friendship and support during my time in graduate school. My parents and family were, and continue to be, a constant source of encouragement. They inspired me to set ambitious goals for myself, and they heartened me when those goals seemed distant. The members of the GVU Center, especially the other students in the Animation Lab and Geometry Group, created the camaraderie that made long hours in the lab often seem more like play than work. They were there to lend a hand when deadlines loomed all too near, and to lend an ear when deadlines seemed all too many.

I am deeply grateful to my advisor, Jessica Hodgins, who's excellent advice and example were invaluable; and to the members of my committee, Greg Turk, Irfan Essa, Jarek Rossignac, and Demetri Terzopoulos, who provided insightful criticism and direction. I am also thankful to Jim Foley and Norberto Ezquerra who helped to set me on the right track during my first years of graduate school.

Finally, and most importantly, I thank my wife, Heather. Her patient love, support, and companionship form the foundation on which my accomplishments have been built.

Contents

List of Tables

List of Figures

Summary

This thesis addresses the problem of graphically modeling and animating the realistic behavior of materials that can undergo fracture due to deformation-induced stress. Using an approach based on linear elastic fracture mechanics and non-linear finite element analysis, three-dimensional volumes are modeled using a mesh of tetrahedral elements. By analyzing the stresses created as the mesh deforms, the simulation determines where cracks should begin and in what directions they should propagate. The system accommodates arbitrary propagation directions by dynamically retesselating the mesh. Because cracks are not limited to element boundaries, the models can form irregularly shaped features as they shatter. This technique overcomes limitations of previous methods that made it difficult to represent the shape of the fracture's surface. Results are presented to demonstrate that this method can be used to animate complex, real-world situations in a compelling, realistic fashion.

Chapter 1

Introduction

The task of specifying the motion of even a simple animated object, like a bouncing ball, is surprisingly difficult. In part, the task is difficult because humans are very skilled at observing movement and quickly detect motion that is unnatural or implausible. Additionally, the motion of many objects is complex and specifying their movement requires a huge amount of data. For example, a piece of cloth can bend and twist in a wide variety of ways, and the breaking bunny statue shown in Figure 1.1 involves many hundreds of individual shards.

Three primary techniques exist for generating synthetic motion: keyframing, motion capture, and procedural methods. Both keyframing and motion capture require that the motion be specified by some external source while procedural methods use an algorithm to automatically compute original motion. Many procedural methods are based on informal heuristics, but a subclass known as physically based modeling makes use of numerical simulations of physical systems to generate synthetic motion of virtual objects. With the introduction of simulated water in *Antz*[54] and

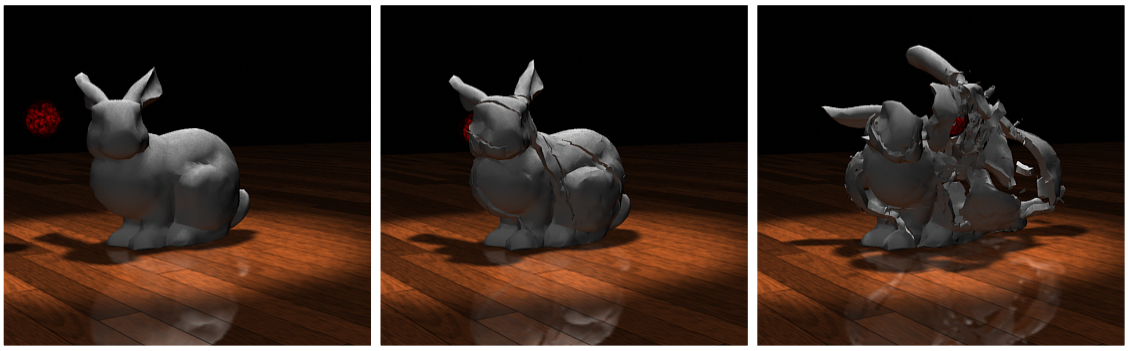


Figure 1.1: **Shattered bunny sequence** – These images show the results of using the technique described in this thesis to simulate the behavior of a hollow, ceramic bunny as it is struck by a heavy, fast-moving weight.

clothing in *Stuart Little* [55], physically based modeling was clearly demonstrated to be a viable technique for commercial animation.

Physically based modeling has proven to be especially effective for animating passive systems which model inanimate objects without an internal source of energy. The advantage of using simulation for passive objects is not surprising, as they tend to have many degrees of freedom, making keyframing or motion capture difficult. Furthermore, while passive objects are often essential to the plot of an animation and to the appearance or mood of a piece, they are not characters and do not require the same control over the subtle details of the motion. Therefore, simulations in which the motion is controlled only by initial conditions, physical equations, and material parameters are often sufficient to produce appealing animations of passive objects. A few examples of passive systems that have been successfully modeled with

physically based simulation include water [17, 28], smoke and other gases [70, 18], sand and other ground surfaces [59, 33], cloth and clothing [9, 68, 65], and hair [26]. Although physically based simulation has been used to model a wide range of passive phenomena, the problem of modeling fracture has not been extensively addressed in the computer graphics literature.

The research described in this thesis addresses the problem of automatically generating realistic synthetic motion for three-dimensional solid objects that can break, crack, or tear. The objects to be animated are modeled using a fast, tetrahedral finite element method that employs linear shape functions within the elements. A nonlinear strain metric is employed with a linear elastic stress-to-strain relationship that has been modified to account for plastic deformation. In order to model fracture initiation and propagation, a metric, the separation tensor, has been developed to encode information about the forces acting on the nodes in the mesh. The separation tensor provides information about when and where a failure should occur as well as the orientation of the failure plane. The system accommodates arbitrary propagation directions by dynamically restructuring the mesh. Because cracks are not limited to the original element boundaries, the objects can form irregularly shaped shards and edges as they shatter.

Although problems similar to the one addressed in this thesis have been investigated in other engineering fields, the methods developed here are unique. The differ-

ences arise because the requirements in graphics and engineering contexts are very different. Engineering applications require that the simulation predict real-world behaviors accurately. In computer animation, simulations of physical phenomena are tools that allow the animator to realize a preconceived behavior. As a result, numerical accuracy is less important and issues relating to visual appearance, ease of use, and computational efficiency become critical. The techniques described in this thesis draw heavily from the fields of fracture mechanics and finite element analysis, however, these different requirements allow simulation techniques for computer animation to make use of simplifications that may be unacceptable in an engineering context.

The specific contributions of this thesis to the field of computer graphics are

- A demonstration that simulation is a viable animation technique for phenomena such as fracture where the motion is too complex to be animated using other techniques.
- A finite element model for elastic and plastic deformation that, although not new to other fields, improves upon previous techniques in the field of computer graphics.
- Introduction of the separation tensor as the basis for a method of determining where a material failure should occur and the orientation of the resulting fracture plane.

- A dynamic re-meshing procedure that can be used to refine an existing tetrahedral finite element mesh and that allows fracture propagation in arbitrary directions independent of the original mesh.
- Several optimizations for using bounding hierarchies to detect collisions between, and self-collisions within, deforming objects of changing topology, and a robust method for responding to collisions.

The following chapters detail these contributions and the results obtained from their implementation. Chapter 2 discusses the relevant previous work. Chapter 3 describes the deformation model that is used to model the basic behavior of a material. The discussion includes both the description of the material as a continuous medium as well as the finite element discretization of the continuous formulation. Chapter 4 focuses on the primary contributions of this thesis which are the failure criteria and the procedures for dynamically re-meshing the finite element mesh to accommodate a fracture. Chapter 5 describes techniques for detecting collisions along with a robust collision response method. Chapter 6 presents the results that have been obtained with these methods and addresses the issue of evaluating methods for animation techniques. Finally, Chapter 7 concludes with potential directions for future work.

Chapter 2

Background

Although this thesis describes a technique for generating motion for computer animation, it does so by adopting an approach that is very similar to analysis techniques that have been used in other fields such as mechanical engineering. Thus, not only is previous work in the field of computer graphics relevant, but so to is previous work that has been done in other fields, particularly the field of fracture mechanics.

2.1 Computer Graphics and Animation

Two papers in the computer graphics literature describe methods for modeling dynamic, deformation-induced fracture. In 1988, Terzopoulos and Fleischer [63, 64] presented a general technique for modeling viscoelastic and plastic deformations. Their method used three fundamental metric tensors to define energy functions that measured deformation over curves, surfaces, and volumes. These energy functions provided the basis for a continuous deformation model that they simulated using several different discretization methods. One of their methods employed a finite

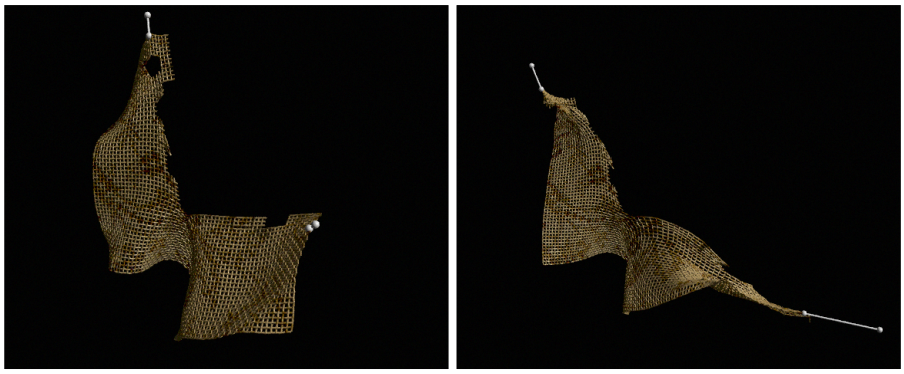


Figure 2.1: **Image from "Deformable models"** – This image demonstrates the technique of Terzopoulos and Fleischer [63] being used to model a sheet of cloth that can be torn. Used with permission.

differencing technique defined by controlled continuity splines [62]. This formulation allowed them to demonstrate how a limited range of fracture effects could be animated by setting the elastic coefficients between adjacent nodes to zero whenever the distance between the nodes reached a threshold. They demonstrated this technique with square sheets of flexible material that could be torn apart. An example of the results they achieved with their method is shown in Figure 2.1.

In 1991, Norton and his colleagues presented a technique for animating 3D solid objects that broke when subjected to large strains [46]. Using their technique, they animated a teapot that shattered when dropped onto a table. (See Figure 2.2.) The technique employed a mass and spring system to model the behavior of the object. When the distance between two attached mass points exceeded a threshold, the simulation severed the spring connection between them. Because mass and spring

Figure 2.2: **Image from "Tipsy Turvy"** – This image, taken from the computer animation "Tipsy Turvy," shows a teapot breaking as it falls onto a hard table. The animation was generated using the mass and spring based technique developed by Norton *et al.* [46]. Used with permission.

systems do not provide a direct way of resisting shear or bending, they constructed their mesh as a diagonally braced lattice. To avoid having flexible strings of partially connected material hanging from the object, their simulation broke an entire cube of springs at once, and the object would break apart into block-shaped fragments.

There are two significant problems with these methods. First, when the material fails, the exact location and orientation of the fracture is not known. Rather the failure is defined as the entire connection between two existing nodes, and the orientation of the fracture plane is left undefined. As a result, the range of effects that these techniques can realistically model is limited to those that occur on a scale much larger than the inter-node spacing.

The second problem is that because the fracture boundary is defined in terms of the initial structure, in these approaches it is subject to directional artifacts similar to the "jaggies" that occur when rasterizing a polygonal edge. These artifacts are particularly noticeable if the discretization follows a regular pattern, and they are easily discernible in both the sheets shown in [63], and the teapot in [46]. If an irregular grid is used, then the artifacts may be partially masked, but the fractures will still be forced onto a path that follows the element boundaries and the results may appear unnatural.

Other relevant research in the computer graphics literature includes techniques for modeling static crack patterns and fractures induced by explosions. Hirota and colleagues described how phenomena such as the static crack patterns created by drying mud can be modeled using a mass and spring system attached to an immobile substrate [22]. The technique is essentially the same as the one of Terzopoulos and Fleischer with the addition of a set of anchored springs that model the substrate. Mazarak *et al.* used a voxel-based approach to model solid objects that break apart when they encounter a spherical blast wave [40]. Fractures along a regular, rectilinear lattice are created using a heuristic based on the strength of a spherical blast wave. Another technique for modeling explosions is that of Neff and Fiume [43]. They use a recursive pattern generator to divide a planar region into polygonal shards that fly apart when acted on by a spherical blast wave.

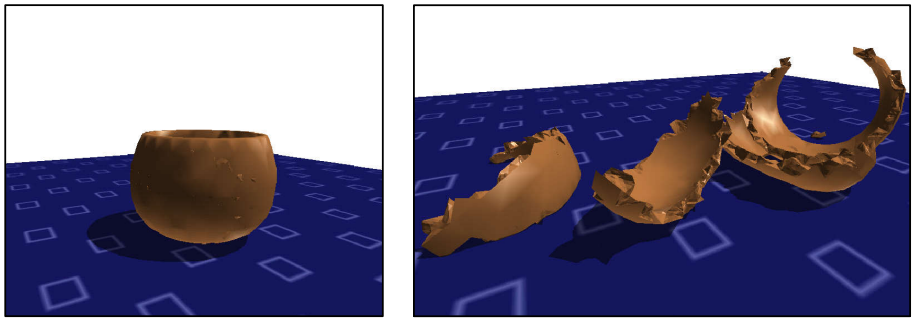


Figure 2.3: **Images from "Fast and Controllable Simulation of the Shattering of Brittle Objects"** – This image demonstrates the technique of Smith, Witkin, and Baraff [57] being used to model a glazed ceramic bowl. Used with permission.

More recently Smith, Witkin, and Baraff have described a technique that makes use of constraint methods to model rigid objects[1] that can crack when subjected to large external forces [57]. The object is modeled as a set of rigid elements that are held together by constraint forces, and the fracture is assumed to occur under static loading conditions. Because the individual elements are rigid and the constraints prevent them from moving with respect to each other, the object cannot deform and it moves like a rigid body. When the magnitude of one of the forces required to enforce a constraint exceeds a threshold, the constraint is removed and fracture is created in the object. An example of the results they achieved with their method is shown in Figure 2.3.

[1]The authors characterize the class of materials that can be modeled with their method as elastic because the materials are rigid and do not deform. However, this rationale is incorrect in that a material's elasticity and rigidity are separate parameters. In fact, their method can only model materials that are completely rigid. Additionally, because elasticity is defined in terms of the relationship between stress and strain (deformation), the classification of a completely rigid material as elastic or inelastic is poorly defined.

The advantage of this constraint based approach is that it is relatively fast. However, because the objects cannot deform, they act like perfectly rigid and ideally elastic objects. As a result the model can only represent a limited range of behaviors. Because the model cannot represent deformation, it cannot model materials, such as rubber, that deform elastically prior to failure. Nor can it model materials such as plastics or most metals that deform plastically. Plastic deformation is particularly important because it plays a critical role in fracture, and it is one of the key factors in determining if a material fractures in a way that would commonly be called tearing or cracking. Furthermore, because the model does not account for the non-rigid dynamics of the object it cannot model dynamic, or rapid, fracture. Situations with moving impacts, such as a glass dropping onto the floor, exhibit dynamic fracture characteristics. Also, because the deformation dynamics are not modeled, the motion of the fragments after the object has fractured will be incorrect.

A final limitation is that the elements are treated as atomic components so that, as with the methods of Terzopoulos and Fleischer and of Norton *et al.*, the object can only break apart along predetermined boundaries. Because the object is divided into elements using a Delaunay tetrahedralization of a quasi-random point set, the boundaries are not regular. Nevertheless, significant artifacts are noticeable in the results generated with the technique.

2.2 Fracture Mechanics

Fracture has been studied extensively in the mechanics literature. The research papers in this field often focus on specific, controlled situations where the assumptions employed in deriving the model are valid. These situations fall into one of four categories depending on whether the crack is static or dynamic, and whether the external forces are static or dynamic [4]. Of these, only the situation where both the crack and the external forces are dynamic is relevant to the problem of animating breaking objects. Additionally, even among those techniques that address dynamic fracture under dynamic conditions, many make use of assumptions that are too restrictive to be used in a general animation context.

In his survey article, Nishioka [44] states that there are three primary discretization methods, finite difference, finite elements, and boundary elements, and several other less commonly used methods including the discrete element method, the finite volume method, and the element-free Galerkin method. Of the techniques described in his survey paper, finite difference, finite elements, and the element-free Galerkin method appear to be the most applicable to computer graphics because they can be used to model networks of multiple three-dimensional cracks. The work described in this thesis makes use of a finite element method.

The finite difference method makes use of numerical differentiation, finite differencing, to compute material derivatives over a regular grid or lattice. Early work in fracture mechanics made use of this technique, but because it is not well-suited for representing irregular geometries it has been largely abandoned in favor of other techniques [44, 4, 31]. Additionally, unless the fracture is aligned with the lattice, or is static, the finite difference method does not provide a way to represent the exact location of the discontinuity and the fracture boundary is subject to the directional artifacts described above.

The finite element method divides the material into a set of disjoint elements that tile the object being modeled. Material derivatives are computed from shape functions that interpolate the material within an element. Because there is a great deal of freedom in how the elements are designed and how additional forces may be introduced to the system, the finite element method has been widely used for modeling fracture problems.

For many finite element techniques, the crack's path is predetermined, and the mesh is built so that element boundaries or specialized elements lie along this path. (See for example [42, 32, 52, 66, 24, 45, 3, 10] and [44, 4] for further discussion.) These techniques are not suitable for computer animation because one of the main reasons for using physically based modeling is to avoid the need to specify where the object should break. Other techniques make use of very fine meshes in the regions where

the crack is expected to develop, and snap the crack to existing element boundaries. (See for example [71] and again [44, 4] for further discussion.) These methods are not suitable for computer graphics because the extremely high resolution meshes are also extremely expensive computationally and because the regions where cracks are expected to develop must still be specified. Furthermore, some of these techniques make use of rectilinear grids, and as demonstrated by previous graphics techniques, viewers tend to be highly sensitive to the directional artifacts that result from forcing the fracture to existing element boundaries.

Techniques that allow fractures to propagate in arbitrary directions have more promise in terms of their being applicable to computer graphics. Within the framework of finite elements, propagation in arbitrary directions may be modeled by dynamically re-meshing an object as the crack advances. Swenson and Ingraffea [61] describe a method for locally re-meshing a two-dimensional domain as a crack moves through it. Their technique makes use of triangular elements. The crack tip is surrounded by an octagonal region that is triangulated radially from a central node located at the crack tip. As the tip advances, the central node moves with the octagonal region and the eight surrounding triangles are distorted to accommodate this motion. When the direction of the crack changes, or the tip comes close to the boundary of the octagonal region, an area with twice the radius of the octagonal region is re-meshed. The authors state that their re-meshing technique is not

robust, and occasional user interaction is required when the re-meshing occurs. Furthermore, the technique may encounter difficulty if the crack turns more than once within a distance equal to the radius of the octagonal region.

Rashid [53] has proposed a finite element alternative to re-meshing for two-dimensional systems. The techniques overlays a special mesh patch around the crack tip in a two-dimensional quadrilateral mesh. This patch is designed to accommodate the tip of the crack and the region of the crack immediately following the tip. As the crack advances through the material, the patch is moved to stay over the crack tip. Elements of the mesh that are partially overlapped by the patch or that are intersected by the trailing crack are modeled using a special integration scheme. Although this technique is developed assuming infinitesimal strains, the author states that it could be extended to handle the general case.

The element-free Galerkin method [11, 37, 58] is a mesh-less technique for modeling material behavior. Like the finite difference and finite element methods, a set of nodes is distributed throughout the area/volume of the object being modeled. The deformed state at any given evaluation point in the material is defined by fitting a moving least-squares approximation to the nodes in a local area about the point. A node's contribution to the least-squares approximation is controlled by a weight that is a function of the node's location relative to the evaluation point. An explicit representation of the boundary is maintained, and fractures are created by

extending or creating internal boundaries. A node's weight at an evaluation point is adjusted to account for any intervening boundaries.

The chief advantage of the element-free Galerkin method is that it is able to handle changes to the object boundary, or refinement by the creation of new nodes without complicated re-meshing. The two main disadvantages of the method are related to the moving least-squares approximation. Because each evaluation point requires computing a least-squares fit to the surrounding node and determining obstructions due to boundary inclusions, the method has a relatively high computational cost. Additionally, the least-squares fit is not an interpolant as it does not actually pass through the node points, and as a result essential boundary conditions must be satisfied using constraints. Because the benefits of the element-free Galerkin method are only relevant around a growing crack, research has been done to build hybrid methods that couple finite element and element-free Galerkin models together [12].

The technique presented in this thesis is a finite element method that makes use of tetrahedral elements with linear shape functions. Because the shape functions are linear, the mesh results in an object description that is only $\mathcal{C}^0$ continuous. This lack of continuity results in errors that would probably be unacceptable for predictive applications. However, as discussed in Chapter 3, the linear elements have several advantages over smoother elements in terms of their computational

efficiency. Because an animation method must be essentially autonomous, the re-meshing is designed to operate without any user input. It is also designed to be able to handle complex networks of intersecting cracks, whereas the methods listed above are intended for situations where a single crack is studied in isolation.

2.3 Molecular Dynamics

While the techniques found in the field of fracture mechanics abstract away from individual particles and treat the material as a continuum, molecular dynamics simulations focus on the individual molecules or atoms that comprise a material. The material behavior, including how it fractures, emerges from the interactions of millions or even hundreds of millions of particles that each represent a single molecule or atom.

This type of simulation is free from nearly all the assumptions and approximations that are required in a continuum approach; the only "rules" built into the simulation are those that govern the interaction between the particles. As a result, molecular dynamics simulations create highly accurate results and can be used to conduct experiments that would be extremely difficult to conduct physically. For example, Abraham [1] has been able to conduct research that may help explain why observed cracks do not reach the theoretical limit of the Rayleigh speed in a material.

Conversely, in a continuum approach, the crack speed is determined by heuristics selected by the researchers to match observed phenomena.

Unfortunately, the computation cost of this extreme accuracy is prohibitively large for most applications involving macroscopic phenomena. Even with massively parallel computers, it can take days to generate a few microseconds of simulated data for a tiny material specimen, on the scale of nanometers. Computer graphic applications often require that objects on the scale of meters, or larger, be simulated for tens of seconds, or longer.

Chapter 3

Deformation Model

The goal of this research is to model the fractures created in a material as it deforms. A prerequisite for accomplishing this goal is a deformation model that adequately describes the material's behavior. The deformation model detailed in this thesis is derived by defining a set of differential equations that describe the aggregate behavior of the material in a continuous fashion, and then using a finite element method to discretize these equations for computer simulation. This approach is fairly standard, and many different deformation models can be derived in this fashion. The one presented here was designed to be simple, fast, and suitable for fracture modeling in the context of computer graphics applications. The first section of this chapter defines the mathematical description of an object's material as a continuous medium, the second section describes how the continuous model can be discretized using a finite element approach to create a model that is suitable for computer simulation.

3.1 Continuous Model

The continuous model provides a description of how a material behaves as it deforms and is based on a continuum mechanics approach. An excellent introduction to this area can be found in the text by Fung [20]. The primary assumption in the continuum approach is that the scale of the effects being modeled is significantly greater than the scale of the material's composition. Therefore, the behavior of the molecules, grains, or particles that compose the material can be modeled as a continuous media. Although this assumption is often valid for modeling deformations, macroscopic fractures can be significantly influenced by effects that occur at small scales where this assumption may not be valid. Because we are interested in graphical appearance rather than rigorous physical correctness, we will put this issue aside and assume that a continuum model is adequate.

The description of the continuous model begins by defining material coordinates that parameterize the volume of space occupied by the object being modeled. Let $\boldsymbol{u} = [u, v, w]^\mathsf{T}$ be a vector in $\Re^3$ that denotes a location in the material coordinate frame as shown in Figure 3.1. The deformation of the material is defined by the function $\boldsymbol{x}(\boldsymbol{u}) = [x, y, z]^\mathsf{T}$ that maps locations in the material coordinate frame to locations in world coordinates. In areas where material exists, $\boldsymbol{x}(\boldsymbol{u})$ is continuous, except across a finite number of surfaces within the volume that correspond to fractures in the material. In areas where there is no material, $\boldsymbol{x}(\boldsymbol{u})$ is undefined.

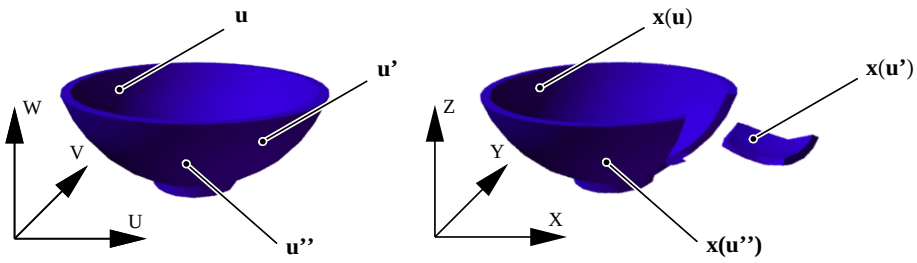


Figure 3.1: **Material and world coordinates** – The material coordinates define a 3D parameterization of the object. The function $\boldsymbol{x}(\boldsymbol{u})$ maps points from their location in the material coordinate frame to their location in the world coordinates. A fracture corresponds to a discontinuity in $\boldsymbol{x}(\boldsymbol{u})$.

3.1.1 The Strain and Strain Rate Tensors

Green's strain tensor, $\boldsymbol{\epsilon}$, is used to measure the total local deformation of the material [19]. It is a nonlinear function of node displacement that has been used extensively in the engineering literature, and can be represented as a 3×3 symmetric matrix defined by

$$\epsilon_{ij} = \left(\frac{\partial \boldsymbol{x}}{\partial u_i} \cdot \frac{\partial \boldsymbol{x}}{\partial u_j} \right) - \delta_{ij} \tag{3.1}$$

where δ_{ij} is the Kronecker delta:

$$\delta_{ij} = \begin{cases} 1 & : \quad i = j \\ \\ 0 & : \quad i \neq j \ . \end{cases} \tag{3.2}$$

This strain metric only measures deformation; it is invariant with respect to rigid body transformations applied to $\boldsymbol{x}$ and vanishes when the material is not deformed. Because it is a tensor, its invariants do not depend on the orientation of the material or world coordinate systems. Green's strain tensor provides a nonlinear model of

finite strain, not a linear model of infinitesimal strain. This nonlinearity allows it to be valid for both stiff materials that deform only a small amount as well as softer materials that exhibit large deformation. The Euclidean metric tensor used by Terzopoulos and Fleischer differs only by the δ_{ij} term [63].

In addition to the strain tensor, we make use of the strain rate tensor, $\boldsymbol{\nu}$, which measures the rate at which the strain is changing. It can be derived by taking the time derivative of (3.1) and is defined by

$$\nu_{ij} = \left(\frac{\partial \boldsymbol{x}}{\partial u_i} \cdot \frac{\partial \dot{\boldsymbol{x}}}{\partial u_j}\right) + \left(\frac{\partial \dot{\boldsymbol{x}}}{\partial u_i} \cdot \frac{\partial \boldsymbol{x}}{\partial u_j}\right) \tag{3.3}$$

where an over dot indicates a derivative with respect to time. For example $\dot{\boldsymbol{x}}$ is the material velocity expressed in world coordinates.

3.1.2 Plasticity

The term plastic deformation describes what happens when a solid object is deformed to an extent that it will no longer return to its original rest configuration once the deforming forces are removed. Many common materials, such as metals, will deform elastically only to a certain extent after which they deform plastically. Most common objects do not experience plastic deformation unless they are damaged, and as a result elastic models are usually sufficient for animating most objects.

However, the focus of this research is to model damage to objects, specifically fracture. The same situations that can cause an object to break, crack, or tear will

also cause it to bend or dent. Thus for damaged objects to be modeled realistically, some form of plastic deformation must be included. Additionally, nearly all materials will undergo some amount of plastic deformation prior to failure and this plastic deformation can have a significant effect on the appearance of the resulting fracture. In particular, the presence of plastic deformation prior to fracture plays an important role in determining whether the resulting behavior would be commonly described as cracking or tearing: the fracture of materials that deform in a primarily elastic fashion is commonly referred to as shattering or cracking, while materials that exhibit significant amounts of plastic deformation tear.

The behavior of the plastic model used here is determined by two parameters, k_1 and k_2. The first parameter, k_1, determines where a material transitions from purely elastic behavior to elasto-plastic behavior. The second parameter, k_2, limits the extent of plastic deformation.

In order to compare the constants, k_1 and k_2, to the amount of deformation a material has experienced , some metric must be used to measure the extent to which the material has been deformed. The strain tensor, $\boldsymbol{\epsilon}$, describes the deformation of the material, but it includes dilation (change of volume). Because solid materials do not change their volume as they deform plastically, a description of the strain that omits dilation is required. The tensor of strain deviation, $\boldsymbol{\epsilon}'$, does just that. It

is defined by

$$\epsilon'_{ij} = \epsilon_{ij} - \frac{1}{3}\sum_{k=1}^{3} \epsilon_{kk}\delta_{ij}. \tag{3.4}$$

The magnitude of $\boldsymbol{\epsilon}'$ is measured by the invariant $J(\boldsymbol{\epsilon}')$ defined by

$$J(\boldsymbol{\epsilon}') = \sum_{i=1}^{3}\sum_{j=1}^{3} \epsilon'_{ij}\epsilon'_{ij} \ . \tag{3.5}$$

So long as $J(\boldsymbol{\epsilon}') \leq k_1$, the material will behave elastically. However, once $\boldsymbol{\epsilon}'$ has exceeded k_1 the material strain, $\boldsymbol{\epsilon}$ is separated into two components: the elastic strain, $\boldsymbol{\epsilon}^e$, and the plastic strain, $\boldsymbol{\epsilon}^p$, so that

$$\boldsymbol{\epsilon} = \boldsymbol{\epsilon}^e + \boldsymbol{\epsilon}^p \ . \tag{3.6}$$

Initially, $\boldsymbol{\epsilon}^p$ is set to zero. As the material deforms the nominal change in $\boldsymbol{\epsilon}^p$ is computed by

$$\Delta\boldsymbol{\epsilon}^p = \begin{cases} \frac{(\boldsymbol{\epsilon}'-\boldsymbol{\epsilon}^p)\left(\sqrt{J(\boldsymbol{\epsilon}'-\boldsymbol{\epsilon}^p)}-k_1\right)}{\sqrt{J(\boldsymbol{\epsilon}'-\boldsymbol{\epsilon}^p)}} & : \quad \sqrt{J(\boldsymbol{\epsilon}'-\boldsymbol{\epsilon}^p)} > k_1 \\ \mathbf{0} & : \quad \text{else} \ . \end{cases} \tag{3.7}$$

To enforce the limit on the amount of plastic deformation, the update of $\boldsymbol{\epsilon}^p$ is done according to

$$\boldsymbol{\epsilon}^p := (\boldsymbol{\epsilon}^p + \Delta\boldsymbol{\epsilon}^p)\min\left(1, \frac{k_2}{\sqrt{J(\boldsymbol{\epsilon}^p + \Delta\boldsymbol{\epsilon}^p)}}\right) \ . \tag{3.8}$$

The behavior of the plasticity model is illustrated in Figure 3.2. Each diagram represents a different state of a single point in a material that is undergoing deformation. The plane shown in each diagram represents the space spanned by the strain

deviation tensor, $\boldsymbol{\epsilon}'$. Although the range of possible strain deviations, $\boldsymbol{\epsilon}'$, defines a five-dimensional space, these two-dimensional diagrams suffice to illustrate plastic behavior.

The inner circle defines elastic limit at $J(\boldsymbol{\epsilon}') = k_1$. The outer circle defines the limit of plastic deformation at $J(\boldsymbol{\epsilon}^p) = k_2$. While the point representing $\boldsymbol{\epsilon}'$ stays within the inner region, as shown in Figure 3.2.a, the material behaves elastically. Once $\boldsymbol{\epsilon}'$ moves beyond the elastic region, as shown in Figure 3.2.b, plastic deformation occurs and the boxed point representing $\boldsymbol{\epsilon}^p$ leaves the origin. Figure 3.2.c shows that as $\boldsymbol{\epsilon}'$ changes, so does $\boldsymbol{\epsilon}^p$. In effect, $\boldsymbol{\epsilon}^p$ moves as if it were being dragged behind $\boldsymbol{\epsilon}'$ by a rope of length k_1. However, $\boldsymbol{\epsilon}^p$ cannot be dragged beyond the plastic limit, k_2, as shown in Figure 3.2.d. Further deformation beyond this limit is resisted elastically. Note that the condition that causes the material to fracture, discussed in Chapter 4, may occur at any point in the elastic or plastic regimes depending on how the fracture parameters for the material have been selected.

3.1.3 The Stress Tensors

The stress tensor, $\boldsymbol{\sigma}$, combines the basic information from the strain and strain rate with the material properties and determines forces internal to the material. Like the strain and strain rate tensors, the stress tensor can be represented as a 3×3 symmetric matrix. It has two components: the elastic stress due to elastic strain,

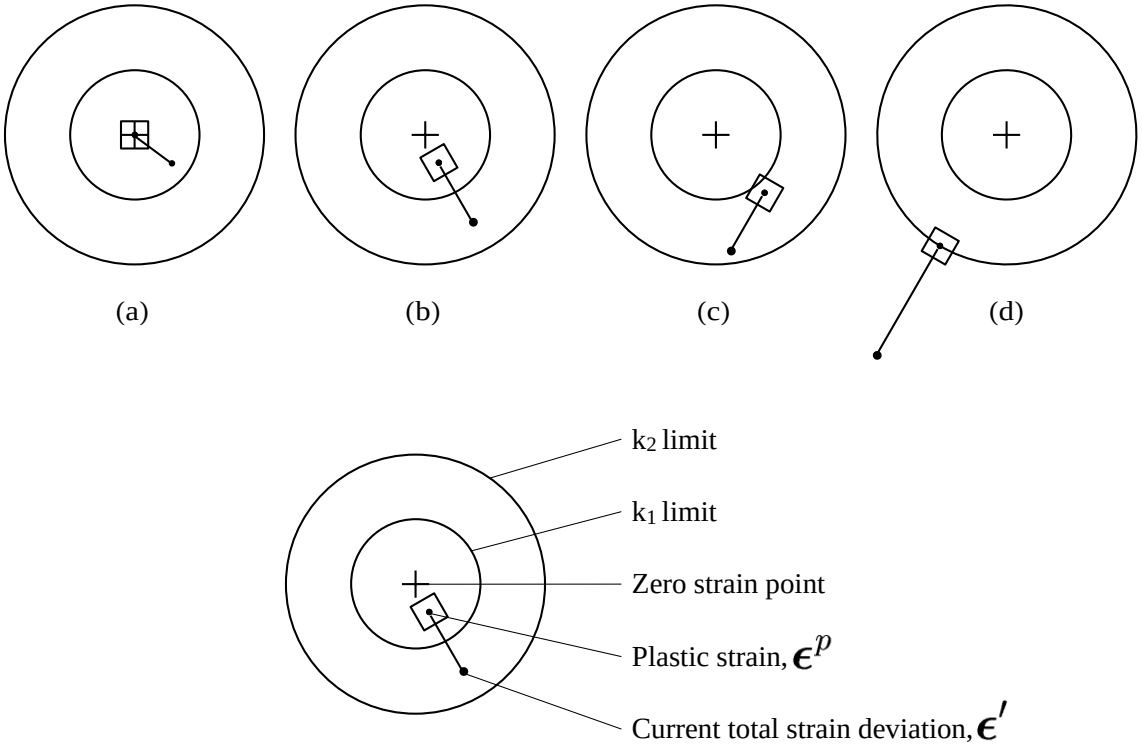


Figure 3.2: **Diagram of plasticity model** – These diagrams illustrate the behavior of the plastic model. **(a)** Elastic deformation. **(b)** and **(c)** Plastic deformation. **(d)** Limit of plastic yield.

$\boldsymbol{\sigma}^{(\epsilon)}$, and the viscous stress due to strain rate, $\boldsymbol{\sigma}^{(\nu)}$. The total internal stress, is the sum of these two components with

$$\boldsymbol{\sigma} = \boldsymbol{\sigma}^{(\epsilon)} + \boldsymbol{\sigma}^{(\nu)} \ . \tag{3.9}$$

The elastic stress and viscous stress are respectively functions of the strain and strain rate. In their most general linear forms, they are defined as

$$\sigma_{ij}^{(\epsilon)} = \sum_{k=1}^{3} \sum_{l=1}^{3} C_{ijkl}\, \epsilon_{kl}^{e} \tag{3.10}$$

$$\sigma_{ij}^{(\nu)} = \sum_{k=1}^{3} \sum_{l=1}^{3} D_{ijkl}\, \nu_{kl} \tag{3.11}$$

where $\boldsymbol{C}$ is the rank-four tensor that encodes the elastic relationship between $\boldsymbol{\epsilon}^e$ and $\boldsymbol{\sigma}^{(\epsilon)}$, and $\boldsymbol{D}$ is the rank-four tensor that defines the material's damping properties.

In three dimensions, each rank-four tensor can be expressed as 81 independent scalars, however because both $\boldsymbol{\epsilon}^e$ and $\boldsymbol{\sigma}^{(\epsilon)}$ are symmetric, many of the entries in $\boldsymbol{C}$ are either redundant or constrained, and $\boldsymbol{C}$ can be reduced to 36 independent values that relate the six independent values of $\boldsymbol{\epsilon}^e$ to the six independent values of $\boldsymbol{\sigma}^{(\epsilon)}$.

If we impose the additional constraint that the material is isotropic, then $\boldsymbol{C}$ collapses further to only two independent values, μ and λ, which are the Lamé constants of the material. With this assumption of isotropy, (3.10) reduces to

$$\sigma_{ij}^{(\epsilon)} = \sum_{k=1}^{3} \lambda \epsilon_{kk}^{e} \delta_{ij} + 2\mu \epsilon_{ij}^{e} \ . \tag{3.12}$$

The material's rigidity is determined by the value of μ, and the resistance to changes in volume (dilation) is controlled by λ. Other methods of describing a material's isotropic elastic properties, for example, the bulk modulus of elasticity and Poisson's ratio, can be related to the Lamé constants using standard formulae [19].

Similarly, $\boldsymbol{D}$ can be reduced to two independent values, ϕ and ψ and (3.11) then reduces to

$$\sigma_{ij}^{(\nu)} = \sum_{k=1}^{3} \phi \nu_{kk} \delta_{ij} + 2\psi \nu_{ij} \ . \tag{3.13}$$

The parameters ϕ and ψ will control how quickly the material dissipates internal kinetic energy. Since $\boldsymbol{\sigma}^{(\nu)}$ is derived from the rate at which ϵ is changing, $\boldsymbol{\sigma}^{(\nu)}$ will not damp motions that are locally rigid, and has the desirable property of dissipating only internal vibrations.

Depending on whether the material being modeled deforms isotropically or not, either equations (3.10) and (3.11), or equations (3.12) and (3.13) should be employed. Whichever is selected, it is a local decision that only affects the computation of $\boldsymbol{\sigma}$.

Unlike the displacement to strain relationship defined by (3.1), the stress to strain relationships given by (3.10) and (3.12) are linear. A linear model provides an adequate description for the behavior of most materials [20]. However, for some materials, particularly biological tissues, a linear stress to strain relationship is inadequate. In these cases, the coefficients that related stress to strain, for example μ and λ, become functions of strain instead of constants.

3.1.4 The Energy Potentials

Once we have the strain, strain rate, and stress tensors, we can compute the elastic potential density, η, and the damping potential density, κ, at any point in the material using

$$\eta = \frac{1}{2} \sum_{i=1}^{3} \sum_{j=1}^{3} \sigma_{ij}^{(\epsilon)} \epsilon_{ij}^{e} \ , \tag{3.14}$$

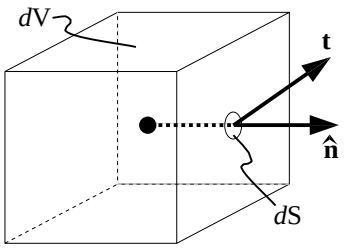


Figure 3.3: **Traction at point in material** – Given a point in the material, the traction, $\boldsymbol{t}$, that acts on the surface element, dS, of a differential volume, dV, centered around the point with outward unit normal, $\hat{\boldsymbol{n}}$ is given by $\boldsymbol{t} = \boldsymbol{\sigma}\,\hat{\boldsymbol{n}}$.

and

$$\kappa = \frac{1}{2}\sum_{i=1}^{3}\sum_{j=1}^{3}\sigma_{ij}^{(\nu)}\nu_{ij}\ . \tag{3.15}$$

These quantities can be integrated over the volume of the material to obtain the total elastic and damping potentials. The elastic potential is the internal elastic energy of the material. The damping potential is related to the kinetic energy of the material after subtracting any rigid body motion and normalizing for the material's density.

The stress can also be used to compute the forces acting internal to the material at any location. Let $\hat{\boldsymbol{n}}$ be an outward unit normal direction of a differential volume centered about a point in the material. (See Figure 3.3.) The traction (force per unit area), $\boldsymbol{t}$, acting on a face perpendicular to the normal is then given by

$$\boldsymbol{t} = \boldsymbol{\sigma}\,\hat{\boldsymbol{n}}\ . \tag{3.16}$$

3.2 Finite Element Discretization

Before we can model a material's behavior using this continuous model, it must be discretized in a way that is suitable for computer simulation. As mentioned in the previous background section, the two most commonly used techniques are the finite difference and finite element methods.

A finite difference method divides the domain of the material into a regular lattice and then uses numerical differencing to approximate the spatial derivatives required to compute the strain and strain rate tensors. This approach is well-suited for problems with a regular structure but becomes complicated when the structure is irregular.

A finite element method partitions the domain of the material into distinct subdomains, or elements as shown in Figure 3.4. Within each element, the material is described locally by a function with some finite number of parameters. The function is decomposed into a set of orthogonal shape, or basis, functions that are each associated with one of the nodes on the boundary of the element. Adjacent elements will have nodes in common, so that the mesh defines a piecewise function over the entire material domain.

The discretization used here employs tetrahedral finite elements with linear polynomial shape functions. By using a finite element method, the mesh can be locally

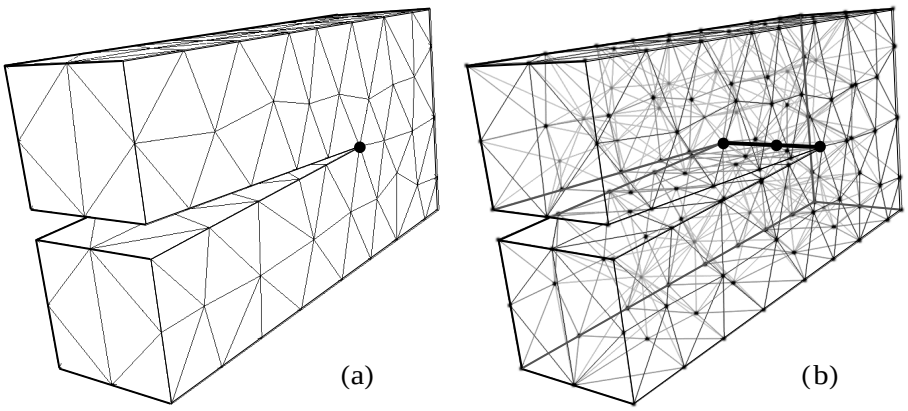


Figure 3.4: **Tetrahedral mesh for a simple object** – Tetrahedral mesh for a simple object. In **(a)**, only the external faces of the tetrahedra are drawn, while in **(b)** the internal structure is shown.

re-meshed to align with the fracture surfaces, thus avoiding directional artifacts on the fracture surfaces. Just as triangles can be used to approximate any surface, tetrahedra can be used to approximate arbitrary volumes. Additionally, when tetrahedra are split along a fracture plane, the resulting pieces can be decomposed exactly into more tetrahedra. Although the linear elements only offer $\mathcal{C}^0$ continuity, they are computationally efficient and the errors introduced do not significantly impact the visual appearance.

Linear elements were selected because higher-order elements are not cost effective for modeling fracture boundaries. Although higher-order polynomials provide individual elements with many degrees of freedom for deformation, they have few degrees of freedom for modeling fracture because the shape of a fracture is defined as a boundary in material coordinates. In contrast, with linear tetrahedra, each

degree of freedom in the world space corresponds to a degree of freedom in the material coordinates. Furthermore, whenever an element is created, its basis functions must be computed. For high-degree polynomials, this computation is relatively expensive. For systems where the mesh is constant, the cost is amortized over the course of the simulation. However, as fractures develop and parts of the object are re-meshed, the computation of basis matrices can become significant. An additional expense arises because computing the internal mesh forces requires integrating the energy potential functions over the element volume. For linear elements, the integration is accomplished trivially. For nonlinear elements, the integration, like the basis computation, becomes expensive.

One potential drawback to using a solid model, as opposed to a surface model, of an object is that most existing computer graphics applications have been designed to work with surface models which most commonly described by a collection of polygons. However, many aspects of a solid object's behavior, for example inertial effects, are not well modeled using surface representations. Luckily, a number of mesh generation software packages are available for creating tetrahedral meshes from polygonal boundaries. The models that I use in my examples were generated either from a CSG (constructive solid geometry) description or a polygonal boundary representation using NETGEN, a publicly available mesh generation package [56].

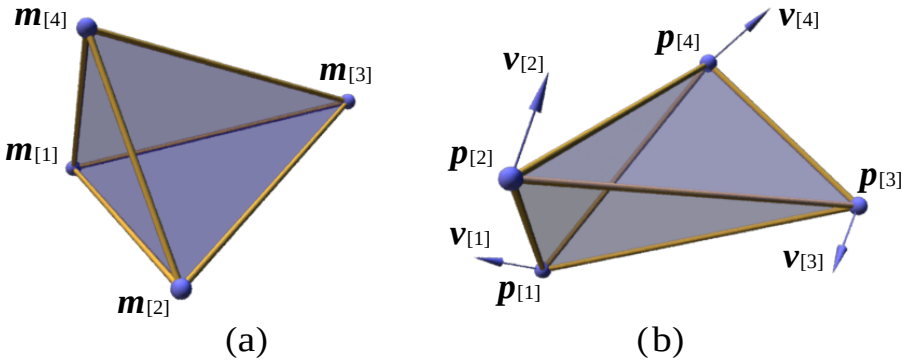


Figure 3.5: **Definition of tetrahedral element** – A tetrahedral element is defined by its four nodes. Each node has **(a)** a location in the material coordinate system, and **(b)** a position and velocity in the world coordinate system.

Once the initial mesh has been generated, each tetrahedral element is defined by four nodes. A node has a position in the material coordinates, $\boldsymbol{m}$, a position in the world coordinates, $\boldsymbol{p}$, and a velocity in world coordinates, $\boldsymbol{v}$. The nodes of a given element are referred to by indexing with square brackets. For example, $\boldsymbol{m}_{[2]}$ is the position in material coordinates of the element's second node. (See Figure 3.5.)

Barycentric coordinates provide a natural way to define the linear shape functions within an element. Let $\boldsymbol{b} = [b_1, b_2, b_3, b_4]^\mathsf{T}$ be barycentric coordinates defined in terms of the element's material coordinates so that

$$\begin{bmatrix} \boldsymbol{u} \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{m}_{[1]} & \boldsymbol{m}_{[2]} & \boldsymbol{m}_{[3]} & \boldsymbol{m}_{[4]} \\ 1 & 1 & 1 & 1 \end{bmatrix} \boldsymbol{b} \,. \tag{3.17}$$

The convention of representing vectors as column matrices is used so that the matrix constructed from the $\boldsymbol{m}_{[i]}$ in (3.17) is a 4×4 matrix.

These barycentric coordinates may also be used to interpolate the node's world position and velocity with

$$\begin{bmatrix} \boldsymbol{x} \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{p}_{[1]} & \boldsymbol{p}_{[2]} & \boldsymbol{p}_{[3]} & \boldsymbol{p}_{[4]} \\ 1 & 1 & 1 & 1 \end{bmatrix} \boldsymbol{b} \tag{3.18}$$

$$\begin{bmatrix} \dot{\boldsymbol{x}} \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{v}_{[1]} & \boldsymbol{v}_{[2]} & \boldsymbol{v}_{[3]} & \boldsymbol{v}_{[4]} \\ 1 & 1 & 1 & 1 \end{bmatrix} \boldsymbol{b} \, . \tag{3.19}$$

To determine the barycentric coordinates of a point within the element specified by its material coordinates, we invert (3.17) and obtain

$$\boldsymbol{b} \, = \, \boldsymbol{\beta} \begin{bmatrix} \boldsymbol{u} \\ 1 \end{bmatrix} \tag{3.20}$$

where $\boldsymbol{\beta}$ is defined by

$$\boldsymbol{\beta} \, = \begin{bmatrix} \boldsymbol{m}_{[1]} & \boldsymbol{m}_{[2]} & \boldsymbol{m}_{[3]} & \boldsymbol{m}_{[4]} \\ 1 & 1 & 1 & 1 \end{bmatrix}^{-1} \, . \tag{3.21}$$

Combining (3.20) with (3.18) and (3.19) yields functions that interpolate the world position and velocity within the element in terms of the material coordinates:

$$\boldsymbol{x}(\boldsymbol{u}) = \boldsymbol{P} \, \boldsymbol{\beta} \begin{bmatrix} \boldsymbol{u} \\ 1 \end{bmatrix} \tag{3.22}$$

$$\dot{\boldsymbol{x}}(\boldsymbol{u}) = \boldsymbol{V} \, \boldsymbol{\beta} \begin{bmatrix} \boldsymbol{u} \\ 1 \end{bmatrix} \tag{3.23}$$

where $\boldsymbol{P}$ and $\boldsymbol{V}$ are defined as

$$\boldsymbol{P} = \begin{bmatrix} \boldsymbol{p}_{[1]} \, \boldsymbol{p}_{[2]} \, \boldsymbol{p}_{[3]} \, \boldsymbol{p}_{[4]} \end{bmatrix} \tag{3.24}$$

$$\boldsymbol{V} = \begin{bmatrix} \boldsymbol{v}_{[1]} \, \boldsymbol{v}_{[2]} \, \boldsymbol{v}_{[3]} \, \boldsymbol{v}_{[4]} \end{bmatrix} \, . \tag{3.25}$$

Note that the rows of $\boldsymbol{\beta}$ are the coefficients of the shape functions, and $\boldsymbol{\beta}$ needs to be computed only when an element is created or the material coordinates of its nodes change. For non-degenerate elements, the matrix in (3.21) is guaranteed to be non-singular, however elements that are nearly co-planar will cause $\boldsymbol{\beta}$ to be ill-conditioned and adversely affect the numerical stability of the system.

Computing the values of $\boldsymbol{\epsilon}$ and $\boldsymbol{\nu}$ within the element requires the first partials of $\boldsymbol{x}$ and $\dot{\boldsymbol{x}}$ with respect to $\boldsymbol{u}$ that are given by

$$\frac{\partial \boldsymbol{x}}{\partial u_i} = \boldsymbol{P}\,\boldsymbol{\beta}\,\boldsymbol{\delta}_i \tag{3.26}$$

$$\frac{\partial \dot{\boldsymbol{x}}}{\partial u_i} = \boldsymbol{V}\,\boldsymbol{\beta}\,\boldsymbol{\delta}_i \tag{3.27}$$

where

$$\boldsymbol{\delta}_i = [\delta_{i1}\,\delta_{i2}\,\delta_{i3}\,0]^{\mathsf{T}}\ . \tag{3.28}$$

Because the element's shape functions are linear, these partials are constant within the element.

The element will exert elastic and damping forces on its nodes. The elastic force on the ith node, $\boldsymbol{f}_{[i]}^{(\epsilon)}$, is defined as the negative partial of the elastic potential density, η, with respect to $\boldsymbol{p}_{[i]}$ integrated over the volume of the element. Given $\boldsymbol{\sigma}^{(\epsilon)}$, $\boldsymbol{\beta}$, and the positions in world space of the four nodes we can compute the elastic force by

$$\boldsymbol{f}_{[i]}^{(\epsilon)} = -\frac{\mathrm{vol}}{2}\sum_{j=1}^{4}\boldsymbol{p}_{[j]}\sum_{k=1}^{3}\sum_{l=1}^{3}\beta_{jl}\beta_{ik}\sigma_{kl}^{(\epsilon)} \tag{3.29}$$

where

$$\text{vol} = \frac{1}{6}[(\boldsymbol{m}_{[2]} - \boldsymbol{m}_{[1]}) \times (\boldsymbol{m}_{[3]} - \boldsymbol{m}_{[1]})] \cdot (\boldsymbol{m}_{[4]} - \boldsymbol{m}_{[1]}) \ . \tag{3.30}$$

Similarly, the damping force on the ith node, $\boldsymbol{f}_{[i]}^{(\nu)}$, is defined as the partial of the damping potential density, κ, with respect to $\boldsymbol{v}_{[i]}$ integrated over the volume of the element. This quantity can be computed with

$$\boldsymbol{f}_{[i]}^{(\nu)} = -\frac{\text{vol}}{2} \sum_{j=1}^{4} \boldsymbol{p}_{[j]} \sum_{k=1}^{3} \sum_{l=1}^{3} \beta_{jl} \beta_{ik} \sigma_{kl}^{(\nu)} \ . \tag{3.31}$$

Summing these two forces, the total internal force that an element exerts on a node is

$$\boldsymbol{f}_{[i]} = -\frac{\text{vol}}{2} \sum_{j=1}^{4} \boldsymbol{p}_{[j]} \sum_{k=1}^{3} \sum_{l=1}^{3} \beta_{jl} \beta_{ik} \sigma_{kl} \ , \tag{3.32}$$

and the total internal force acting on the node is obtained by summing the forces exerted by all elements that are attached to the node.

As the element is compressed to less than about 30% of its material volume, the gradient of η and κ start to vanish causing the resisting forces to fall off. However, as discussed in the section on plasticity, nearly all solid materials, even those that are highly flexible, conserve their volume to within a few percent. Materials, such as sponge or some biological tissues, that do compress significantly are relatively uncommon. Highly compressible materials would not only require a formulation for η and κ that did not flatten out after compression, but would also require a nonlinear elasticity model.

Using a lumped mass formulation, the mass contributed by an element to each one of its nodes is determined by integrating the material density, ρ, over the element shape function associated with that node. In the case of tetrahedral elements with linear shape functions, this mass contribution is simply $\rho\,\mathrm{vol}/4$, where vol is the volume of the element in material coordinates as defined in (3.30).

3.3 Pseudo Code for Deformation Model

The derivations above are sufficient for a simulation that uses an explicit integration scheme. Additional work, including computing the Jacobian of the internal forces, is necessary for an implicit integration scheme. (See for example [9] and [16].) The following pseudo code illustrates how the inner loop of a simulation using an explicit Euler integration scheme might be implemented. Note that this pseudo code is for illustration only as the Euler integration step performed in lines 17–19 has very poor stability characteristics [51].

Simulation Inner Loop

```
1   foreach ell ∈ elements of ( mesh )
2       compute ε_ell using (3.1) and (3.26)
3       compute ν_ell using (3.3), (3.26) and (3.27)
4       compute ε' using (3.4)
5       if J(ε') > k_1
6           update ε^p_ell according to (3.8)
7       ε^e_ell := ε_ell − ε^p_ell
8       compute σ^(ε) using (3.12)
9       compute σ^(ν) using (3.13)
10      σ_ell := σ^(ε) + σ^(ν)
11  foreach nod ∈ nodes of ( mesh )
12      f_nod := 0
13      foreach ell ∈ elements attached to ( nod )
14          accumulate f_nod due to ell using (3.29)
15      acc_nod := f_nod ÷ mass( nod )
16  compute accelerations due to external forces, i.e. gravity, collisions
17  foreach nod ∈ nodes of ( mesh )
18      x_nod += Δt v_nod + ½(Δt)² acc_nod
19      v_nod += Δt acc_nod
```

Chapter 4

Fracture Model

The finite element deformation model described in the preceding chapter provides a method for modeling the basic physical behavior of a material as it moves and deforms. While the deformation model is a useful animation tool by itself, it also provides the framework that is necessary to physically model the behavior of a material as it fractures.

In this chapter, a fracture model is developed within the framework provided by the deformation model. The key components of the fracture model are a criteria for determining when and where a failure should occur, a method for determining the orientation of the resulting fracture, and a method for updating the finite element mesh to accommodate the new fracture surface.

4.1 Failure Criteria

There are three loading modes by which forces can be applied to a crack causing it to open further. (See Figure 4.1.) In most circumstances, some combination of these

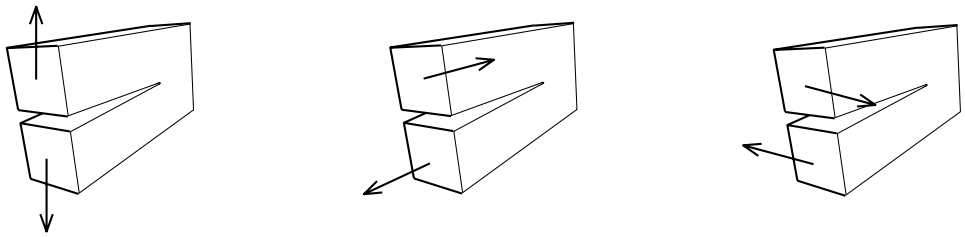


Figure 4.1: **Three loading modes that can be experienced by a crack** – Three loading modes that can be experienced by a crack. Mode I: Opening, Mode II: In-Plane Shear, and Mode III: Out-of-Plane Shear. Adapted from Anderson [2].

modes will be active, producing a mixed mode load at the crack tip. For all three cases, as well as mixed mode situations, the behavior of the crack can be resolved by analyzing the forces acting at the crack tip: tensile forces that are opposed by other tensile forces will cause the crack to continue in a direction that is perpendicular to the direction of largest tensile load, and conversely, compressive loads will tend to arrest a crack to which they are perpendicular. Loads on a node that are not opposed by another load at the node are unbalanced. Unbalanced load will cause a translation of the node and will not cause a fracture to develop. (See Figure 4.2.)

The finite element model describes the surface of a fracture with elements that are adjacent in material coordinates but that do not share nodes across the fracture surface. The curve that represents the crack tip is implicitly defined in a piecewise linear fashion by the nodes that border the fracture surface, and further extension of the crack may be determined by analyzing the internal forces acting on these nodes.

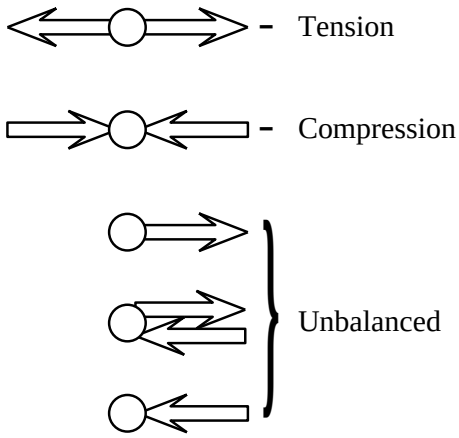


Figure 4.2: **Balanced and unbalanced loads** – Loads that are balanced by opposite loads will tend to promote or arrest a failure. Unbalanced loads will have no effect on the development of a fracture.

The element nodes will also be used to determine where a crack should be initiated. This strategy could potentially introduce unpleasant artifacts. However, because the surface of an object is defined by a polygonal boundary (the outer faces of the tetrahedra) there will always be a node located at any concavities. (See Figure 4.3.) Because concavities are precisely the locations where cracks commonly begin, this limitation is an acceptable one. Additionally, for situations where a failure would occur at an area other than a concavity, for example the bent rod shown in Figure 4.4, then the fact that the finite element model can represent the deformation indicates that a node will be located in an appropriate location.

The fracture algorithm is as follows: after every timestep, the system resolves the internal forces acting on all nodes into their tensile and compressive components,

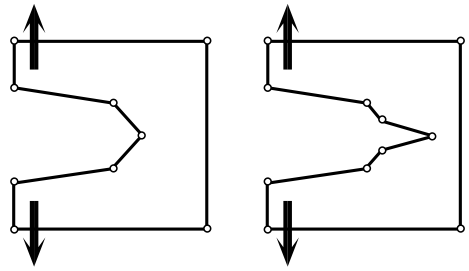


Figure 4.3: **Failures in concavities** – Nodes will occur at sharp features and concavities, locations where failures are likely to occur.

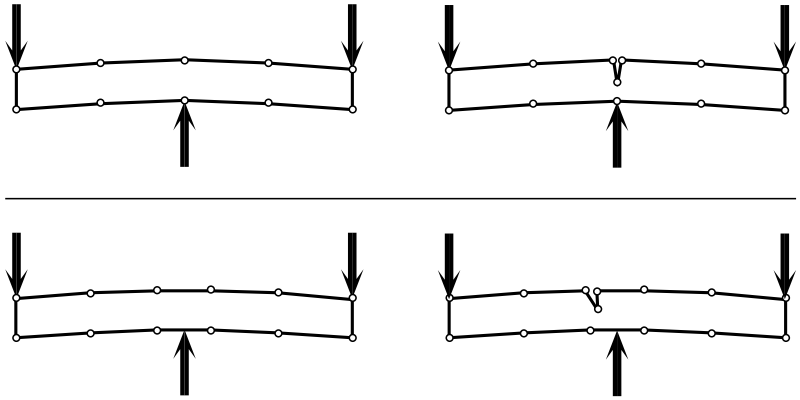


Figure 4.4: **Failures on smooth surfaces** – Failures may also occur on smooth portion of an object's surface. However, if the finite element model is able to represent the deformation then a node should be located in an appropriate place.

discarding any unbalanced portions. At each node, the resulting forces are then used to form a tensor that describes how the internal forces are acting to separate that node. If the action is sufficiently large, the node is split into two distinct nodes and a fracture plane is computed. All elements attached to the node are divided along the plane with the resulting tetrahedra assigned to one or the other incarnations of the split node, thus creating a discontinuity in the material. Any cached values, such

as the node mass or the element shape functions, are recomputed for the affected elements and nodes. The location of a fracture or crack tip need not be explicitly recorded unless this information is useful for some other purpose, such as rendering.

4.1.1 Force Decomposition

The forces acting on a node are decomposed by first separating the element stress tensors into tensile and compressive components, as shown graphically in Figure 4.5. For a given element in the mesh, let $\mathsf{v}^i(\boldsymbol{\sigma})$, with $i \in \{1, 2, 3\}$, be the ith eigenvalue of $\boldsymbol{\sigma}$, and let $\hat{\mathbf{n}}^i(\boldsymbol{\sigma})$ be the corresponding unit length eigenvector. Positive eigenvalues correspond to tensile stresses and negative ones to compressive stresses. Since $\boldsymbol{\sigma}$ is real and symmetric, it will have three real, not necessarily unique or non-zero, eigenvalues. In the case where an eigenvalue has multiplicity greater than one, the eigenvectors are selected arbitrarily to orthogonally span the appropriate subspace [51].

Given a vector $\boldsymbol{a}$ in $\Re^3$, we can construct a 3×3 symmetric matrix, $\mathbf{m}(\boldsymbol{a})$ that has $|\boldsymbol{a}|$ as an eigenvalue with $\boldsymbol{a}$ as the corresponding eigenvector, and with the other two eigenvalues equal to zero. This matrix is defined by

$$\mathbf{m}(\boldsymbol{a}) = \begin{cases} \boldsymbol{a}\,\boldsymbol{a}^{\mathsf{T}}/|\boldsymbol{a}| & : \quad \boldsymbol{a} \neq \mathbf{0} \\ \mathbf{0} & : \quad \boldsymbol{a} = \mathbf{0}\,. \end{cases} \tag{4.1}$$

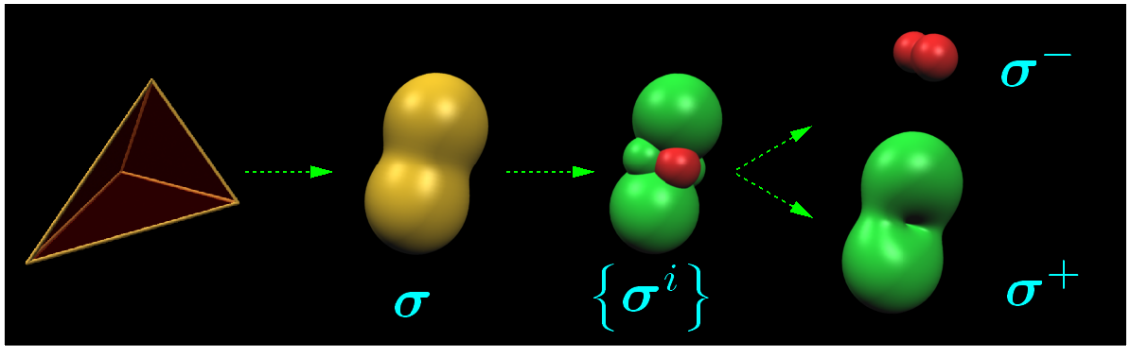


Figure 4.5: **Illustration of decomposing element stress** – The stress tensor for each element is separated into its principal components using an eigen decomposition. The principal components are then recombined to form the tensile and compressive components of the element stress tensor.

The tensile component, $\boldsymbol{\sigma}^+$, and compressive component, $\boldsymbol{\sigma}^-$, of the stress within the element can now be computed by summing over the eigenvectors with positive or negative eigenvalues:

$$\boldsymbol{\sigma}^+ = \sum_{i=1}^{3} \max(0, \mathsf{v}^i(\boldsymbol{\sigma}))\ \mathbf{m}(\hat{\mathbf{n}}^i(\boldsymbol{\sigma})) \tag{4.2}$$

$$\boldsymbol{\sigma}^- = \sum_{i=1}^{3} \min(0, \mathsf{v}^i(\boldsymbol{\sigma}))\ \mathbf{m}(\hat{\mathbf{n}}^i(\boldsymbol{\sigma}))\ . \tag{4.3}$$

Using this decomposition, the force that an element exerts on a node can be separated into a tensile component, $\boldsymbol{f}^+_{[i]}$, and a compressive component, $\boldsymbol{f}^-_{[i]}$. This separation is done by reevaluating the internal forces exerted on the nodes using (3.32) with $\boldsymbol{\sigma}^+$ or $\boldsymbol{\sigma}^-$ substituted for $\boldsymbol{\sigma}$. Thus the tensile component is

$$\boldsymbol{f}^+_{[i]} = -\frac{\mathrm{vol}}{2} \sum_{j=1}^{4} \boldsymbol{p}_{[j]} \sum_{k=1}^{3} \sum_{l=1}^{3} \beta_{jl}\beta_{ik}\sigma^+_{kl}\ . \tag{4.4}$$

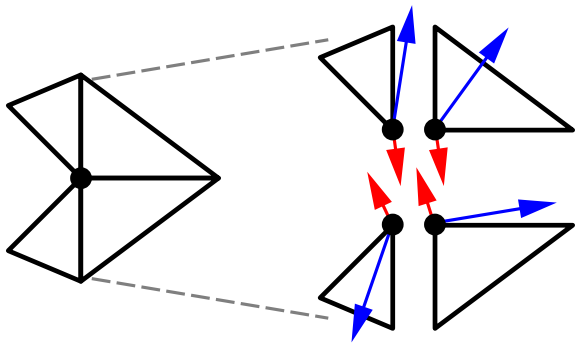


Figure 4.6: **Tensile and compressive forces exerted on a node** – The forces that each element exerts on a node is decomposed into a tensile and a compressive component. Red arrows indicate the compressive force, $\boldsymbol{f}^-_{[i]}$, that an element exerts on a node, and blue arrows represent the tensile forces, $\boldsymbol{f}^+_{[i]}$, that an element exerts on a node.

The compressive component, $\boldsymbol{f}^-_{[i]}$, can be computed similarly with

$$\boldsymbol{f}^-_{[i]} = -\frac{\text{vol}}{2} \sum_{j=1}^{4} \boldsymbol{p}_{[j]} \sum_{k=1}^{3} \sum_{l=1}^{3} \beta_{jl} \beta_{ik} \sigma^-_{kl} \ , \tag{4.5}$$

but because $\boldsymbol{\sigma} = \boldsymbol{\sigma}^+ + \boldsymbol{\sigma}^-$, it can be computed more efficiently using $\boldsymbol{f}_{[i]} = \boldsymbol{f}^+_{[i]} + \boldsymbol{f}^-_{[i]}$.

Each node will now have a set of tensile and a set of compressive forces that are exerted by the elements attached to it as shown in Figure 4.6. For a given node, we denote these sets as $\{\boldsymbol{f}^+\}$ and $\{\boldsymbol{f}^-\}$ respectively. The unbalanced tensile load, $\boldsymbol{f}^+$, is simply the sum over $\{\boldsymbol{f}^+\}$, and the unbalanced compressive load, $\boldsymbol{f}^-$, is the sum over $\{\boldsymbol{f}^-\}$.

4.1.2 The Separation Tensor

The forces acting at the nodes are described using a stress variant that I call the separation tensor, $\boldsymbol{\varsigma}$. The separation tensor is formed from the balanced tensile and compressive forces acting at each node and is computed by

$$\boldsymbol{\varsigma} = \frac{1}{2}\left(-\mathbf{m}(\boldsymbol{f}^{+}) + \sum_{\boldsymbol{f}\in\{\boldsymbol{f}^{+}\}} \mathbf{m}(\boldsymbol{f}) + \mathbf{m}(\boldsymbol{f}^{-}) - \sum_{\boldsymbol{f}\in\{\boldsymbol{f}^{-}\}} \mathbf{m}(\boldsymbol{f})\right) . \tag{4.6}$$

It does not respond to unbalanced actions that would produce a rigid translation, and is invariant with respect to transformations of both the material and world coordinate systems.

The separation tensor is used directly to determine whether a fracture should occur at a node. Let v^{+} be the largest positive eigenvalue of $\boldsymbol{\varsigma}$. If v^{+} is greater than the material toughness, τ, then the material will fail at the node. The orientation in world coordinates of the fracture plane is perpendicular to $\hat{\mathbf{n}}^{+}$, the eigenvector of $\boldsymbol{\varsigma}$ that corresponds to v^{+}. (See Figure 4.7.) In the case where multiple eigenvalues are greater than τ, multiple fracture planes may be generated by first generating the plane for the largest value, re-meshing (see below), and then recomputing the new value for $\boldsymbol{\varsigma}$ and proceeding as above.

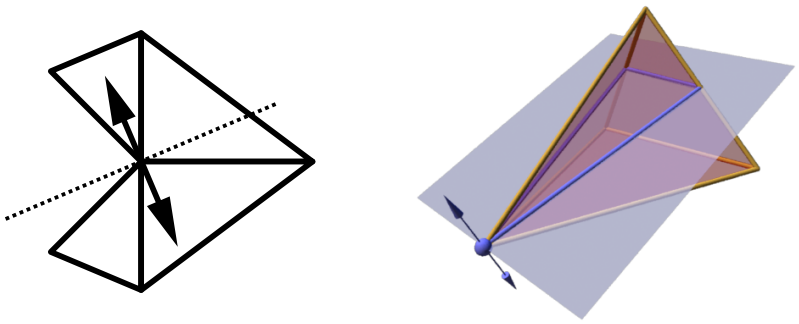


Figure 4.7: **Orientation of Fracture plane** – The fracture plane is perpendicular to $\hat{\mathbf{n}}^+$, the eigenvector of $\boldsymbol{\varsigma}$ that corresponds to $\mathbf{v}^+$. The diagram on the left shows a two-dimensional example. The bold arrow indicates $\hat{\mathbf{n}}^+$, and the dotted line is the resulting fracture boundary. On the right, a three-dimensional diagram is shown. The blue arrow indicates $\hat{\mathbf{n}}^+$, and the blue plane cutting the tetrahedra is the resulting fracture plane.

4.1.3 Local Re-Meshing

Once the simulation has determined the location and orientation of a new fracture plane, the mesh must be modified to reflect the new discontinuity. The orientation of the fracture must be preserved, as approximating it with the existing element boundaries would create undesirable artifacts. Therefore, the algorithm re-meshes the local area surrounding the new fracture by splitting elements that intersect the fracture plane and modifying neighboring elements to ensure that the mesh stays self-consistent.

First, the node where the fracture originates is replicated so that there are now two nodes, q^+ and q^- with the same material position, world position, and velocity. The masses will be recalculated later. The discontinuity passes "between" the two

co-located nodes. The positive side of the fracture plane defined by the plane's normal, $\hat{\mathbf{n}}^+$, is associated with q^+ and the negative side with q^-.

Next, all elements that were attached to the original node are examined, comparing the world location of their nodes to the fracture plane. If an element is not intersected by the fracture plane, then it is reassigned to either q^+ or q^- depending on which side of the plane it lies.

If the element is intersected by the fracture plane, it is split along the plane. (See Figures 4.8 and 4.9.) A new node is created along each edge that intersects the plane. Because all elements must be tetrahedra, in general each intersected element will be split into three tetrahedra. One of the tetrahedra will be assigned to one side of the plane and the other two to the other side. Because the two tetrahedra that are on the same side of the plane both share either q^+ or q^-, the discontinuity does not pass between them. (See detail in Figure 4.9.)

In addition to the elements that were attached to the original node, it may be necessary to split other elements so that the mesh stays consistent. In particular, an element must be split if the face or edge between it and another element that was attached to the original node has been split. (See Figures 4.8 and 4.10.) To prevent the re-meshing from cascading across the entire mesh, these splits are done so that the new tetrahedra use only the original nodes and the nodes created by

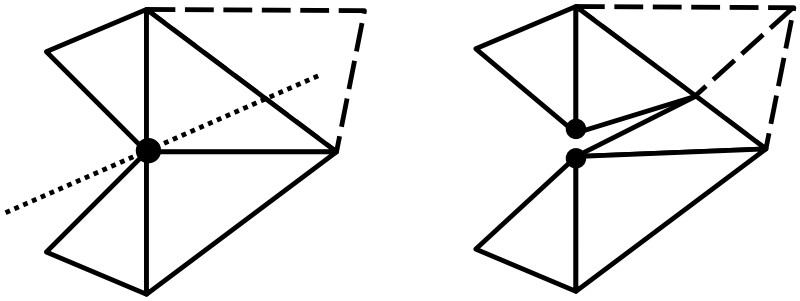


Figure 4.8: **Element split in $\Re^2$** – Elements that intersect the fracture plane are split and the resulting fragments assigned to q^+ and q^-. Edge neighbors must be updated to maintain mesh consistency. The two nodes created from the splitting node are co-located, the geometric displacement shown in the right-hand figure only illustrates the location of the fracture discontinuity.

the intersection splits. Because no new nodes are created, the effect of the local re-meshing is limited to the elements that are attached to the node where the fracture originated and their immediate neighbors. Because the tetrahedra formed by the secondary splits do not attach to either q^+ or q^-, the discontinuity does not pass between them. Finally, after the local re-meshing has been completed, any cached values that have become invalid, such as the element basis matrices or the node masses, must be recomputed.

Two additional subtleties must also be considered. The first occurs when an intersection split involves an edge that is formed only by tetrahedra attached to the node where the crack originated. When this happens, the fracture has reached a boundary in the material, and the discontinuity should pass through the edge.

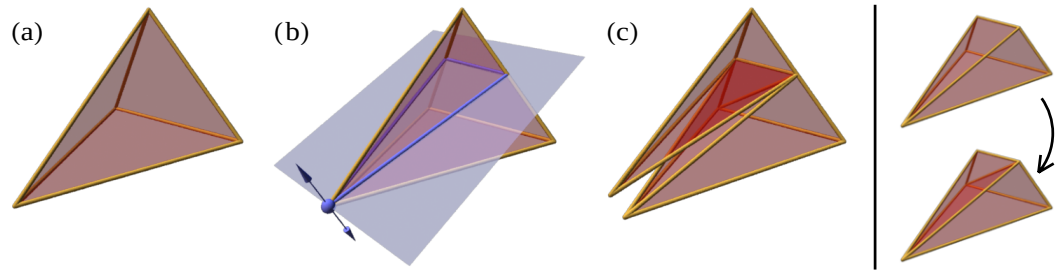


Figure 4.9: **Primary element split in $\Re^3$ –** **(a)** The initial tetrahedral element. **(b)** The splitting node and fracture plane are shown in blue. **(c)** The element is split along the fracture plane into two polyhedra that are then decomposed into tetrahedra. Note that the two nodes created from the splitting node are co-located, the geometric displacement shown in **(c)** only illustrates the location of the fracture discontinuity.

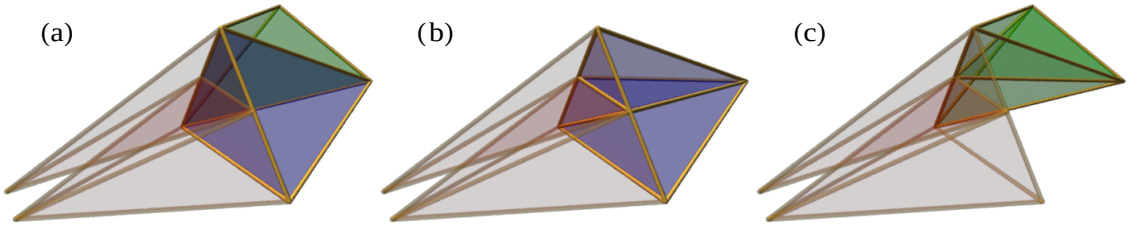


Figure 4.10: **Secondary element split in $\Re^3$ –** Elements that are adjacent to an element that has been split by a fracture plane must also be split to maintain mesh consistency. **(a)** Neighboring tetrahedra prior to split. **(b)** Face neighbor after split. **(c)** Edge neighbor after split.

Re-meshing occurs as described above, except that two nodes are created on the edge and one is assigned to each side of the discontinuity.

Second, the fracture plane may pass arbitrarily close to an existing node producing arbitrarily ill-conditioned tetrahedra. To avoid this, the system employs two thresholds, one on the distance between the fracture plane and an existing node, and the other on the angle between the fracture plane and a line from the node

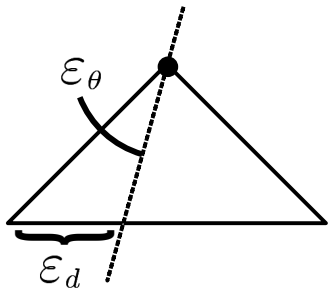


Figure 4.11: **Element split thresholds** – Two thresholds are used to avoid the creation of ill-conditioned elements. One limits the minimum distance between the fracture plane and an existing node and the other limits the minimum angle between the fracture plane and a line from the node where the split originated to the existing node.

where the split originated to the existing node. (See Figure 4.11.) If either of these thresholds are not met, then the intersection split is snapped to the existing node. To prevent visual artifacts from appearing, these thresholds should be kept small. For example values of $\varepsilon_\theta = 0.078$ radians and $\varepsilon_d = 2$ mm were found to be suitable for objects that are approximately one meter in size.

4.1.4 Pseudo Code for Fracture Model

The following pseudo code illustrates the computation fracture model and it would be called within the simulation's inner loop. Referring to the routine described in Section 3.3, an appropriate place to insert this routine would be after the element stress tensors have been computed in lines 1–10. For simplicity, the boundary and threshold tests have been omitted.

Generate Fractures

```
1   foreach ell ∈ elements of ( mesh )
2       compute σ⁺_ell using (4.2)
3       compute σ⁻_ell using (4.3)
4   foreach nod ∈ nodes of ( mesh )
5       f⁺ := 0
6       f⁻ := 0
7       ς := 0
8       foreach ell ∈ elements attached to ( nod )
9           compute f⁺_ell using (4.4)
10          compute f⁻_ell using (4.5)
11          f⁺ += f⁺_ell
12          f⁻ += f⁻_ell
13          ς += m(f⁺_ell) − m(f⁻_ell)
14      ς -= m(f⁺) − m(f⁻)
15      compute v⁺ of ς
16      if v⁺ ≥ τ
17          q⁺ := nod
18          q⁻ := replicate ( q⁺ )
19          pending := elements attached to ( nod )
20          while not empty ( pending )
21              ell := remove item( pending )
22              if split by plane( n̂⁺ , ell )
23                  add items ( pending , split element ( n̂⁺ , ell ) )
24              else if on negative side ( n̂⁺ , ell )
25                  reassign ell from q⁺ to q⁻
```

4.1.5 Example Results with Base Failure Criteria

The proceeding sections of this chapter describe the basic method that I have developed for graphically modeling and animating fracture. Although the bulk of the results that I have obtained with this technique will be presented in Chapter 6, a simple example is presented here to demonstrate that the technique works as described, and to illustrate two problems that must also be addressed.

The example object that I will use to demonstrate the method is shown in Figure 4.12. It is a solid block that has a preexisting crack extending halfway into it from the left side. Although the block is three-dimensional, in order to keep the example simple, it is strained in a two-dimensional fashion and rendered with an orthographic camera. The top edge of the block is held fixed in space while the hole in the lower half is constrained to move downward at a constant velocity. The results of applying the basic fracture algorithm described above to this example is shown in Figure 4.13. As the material is pulled apart, the pre-cut crack extends through the object until it eventually reaches the other side of the block.

4.2 Fast Propagation

Although the orientation of the fracture surfaces is arbitrary, the speed of propagation is not. When the separation at a node exceeds the material toughness

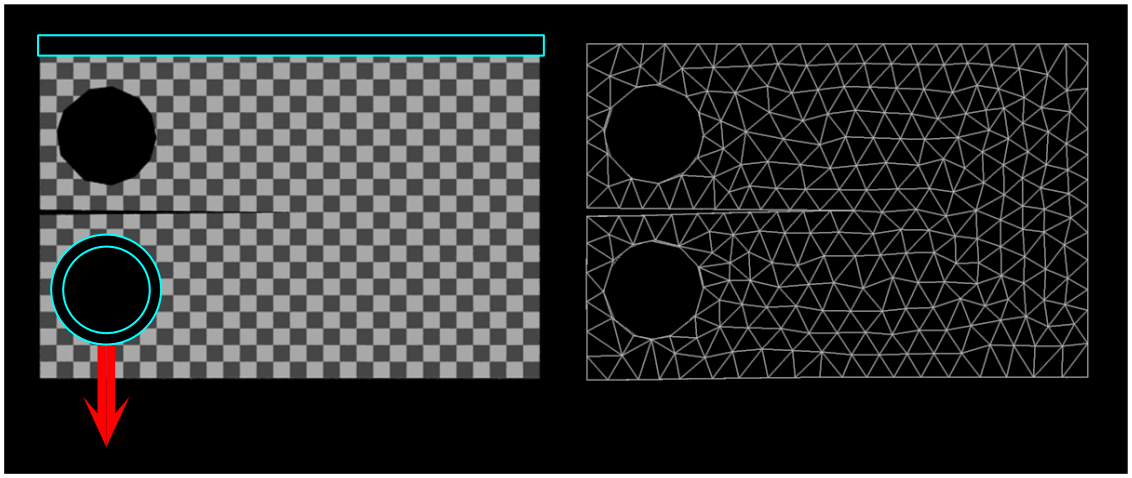


Figure 4.12: **Example block setup and block mesh** – As illustrated on the left, the top edge of the block is held fixed in space while the hole in the lower half is constrained to move downward at a constant velocity. The mesh used to model the block is shown on the right. Although the block is a three-dimensional object, this example situation has been set up as a two-dimensional example for the sake of clarity.

threshold, the fracture only propagates the width of a single element. Thus, the distance that a fracture may travel during a timestep is determined by the size of the existing mesh elements. The crack may either split an element or not; it cannot travel only a fraction of the distance across an element, nor can it travel across multiple elements.

The lower bound on the speed of the crack could result in both temporal and spatial artifacts. Theoretically, if a crack were being opened slowly by an applied load on a model with a coarse resolution mesh, this limitation would lead to a "button popping" effect where the crack would travel across one element, pause

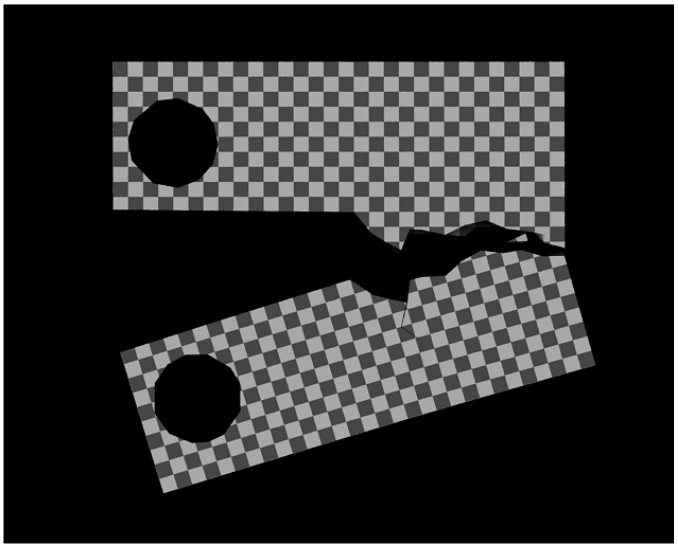


Figure 4.13: **Basic crack in sample block** – The results of applying the basic fracture algorithm to the sample block.

until the stress built up again, and then move across the next element. Additionally, because the fracture can only change its orientation at element boundaries, a very course initial mesh could cause the crack to appear faceted.

Neither of these lower bound related artifacts has proven to be a problem in the examples I have tried. A fracture typically advances quite rapidly, on the order of the speed of sound in a material, and as a result the temporal "button popping" effect does not occur on a visual time scale. Certainly the discrete nature of the crack advance will introduce spurious high frequency vibrations into the simulation and possibly introduce errors in the results. However, these vibrations are at frequencies several orders of magnitude above the video frame rate, and therefore not observable.

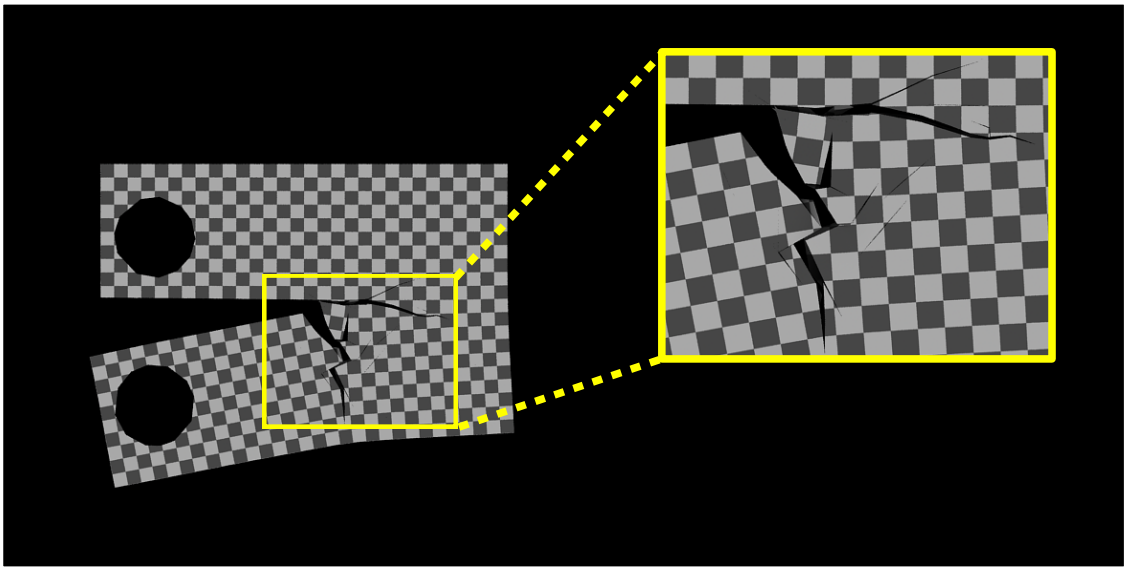


Figure 4.14: **Results of base method with large timestep** – Because the crack should advance more rapidly than the simulation is able to model, spontaneous failures occur ahead of its path. These failures then connect back to the original crack and create spurious bifurcations.

As noted earlier, the goal of this research is to produce visual accuracy so these errors are acceptable.

The upper bound on the speed of the crack is a more serious problem. Figure 4.14 illustrates what happens when the timestep on the simulation is increased by a factor of 10. The crack can only propagate at a speed equal to the size of the next element divided by the timestep. If this speed is less than the speed indicated by the physical parameters of the simulation then a high stress area will race ahead of the crack tip, causing spontaneous failures to occur in the material. In many respects, this limit

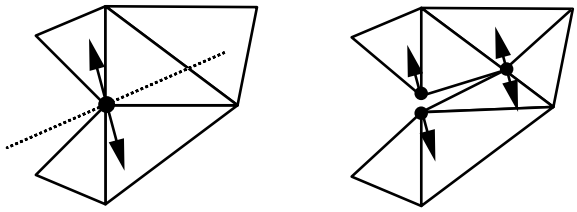


Figure 4.15: **Diagram of pushing residual** – When v^+ at a node exceeds τ a failure occurs. A portion of the residual, v^*, is propagated forward to the new crack tip.

on the maximum timestep imposed by the crack speed is similar to the Courant limit that is related to system stiffness [16].

A solution to this problem would be to allow the crack to propagate across many elements in a single timestep. Unfortunately, simply calling the fracture procedure multiple times each timestep will not have this effect. The nodes just prior to the crack tip must be given simulation time to move apart so that stress accumulates at the new tip.

To approximate this effect, a heuristic is used to propagate separation residuals forward along the crack's path. If timesteps were infinitesimally small, then nodes would fail precisely when $\mathsf{v}^+ = \tau$. However, because the timesteps are finite, failure will occur with $\mathsf{v}^+ \geq \tau$ so that there is some residual, $\mathsf{v}^* = \mathsf{v}^+ - \tau$, beyond the critical value. The heuristic is to propagate this residual as the crack advances to the nodes along the newly created crack tip. This technique is illustrated in Figure 4.15. If a node with separation $\boldsymbol{\varsigma}$ and residual v^* fails along a plane normal to $\hat{\mathbf{n}}^+$ then the

separation at the node on the new crack tip, $\boldsymbol{\varsigma}'$ is modified according to

$$\boldsymbol{\varsigma}' := \boldsymbol{\varsigma}' + \frac{\alpha}{n}\mathbf{m}(\hat{\mathbf{n}}^+)\mathsf{v}^* \ , \tag{4.7}$$

where n is the number of nodes along the crack tip that are connected to the failing node and $\alpha \in [0..1]$ is a coefficient that determines how active the heuristic is.

The results of applying this heuristic to the cut block example are shown in Figure 4.16. As desired, the spurious bifurcations have been largely eliminated from the large timestep trial shown in Figure 4.16.b.

This heuristic also changes the results for small timestep trials. (Compare Figure 4.13 to Figure 4.14.a.) In particular, the crack in the original run made a sharp turn upwards approximately one fifth of the way along its path while the run with $\alpha = 1.0$ continues along a relatively straight path. This difference is explained by considering that large values of α will tend to bias the material towards straight cracks.

4.3 Back-Crack Avoidance

The second limitation in the basic method stems from the fact that while the fracture plane's orientation is well defined, the crack tip's forward direction is not. As shown in Figure 4.18, if a crack turns at an angle greater than half the angle at the crack tip, then a secondary fracture will develop in the opposite direction to the crack's

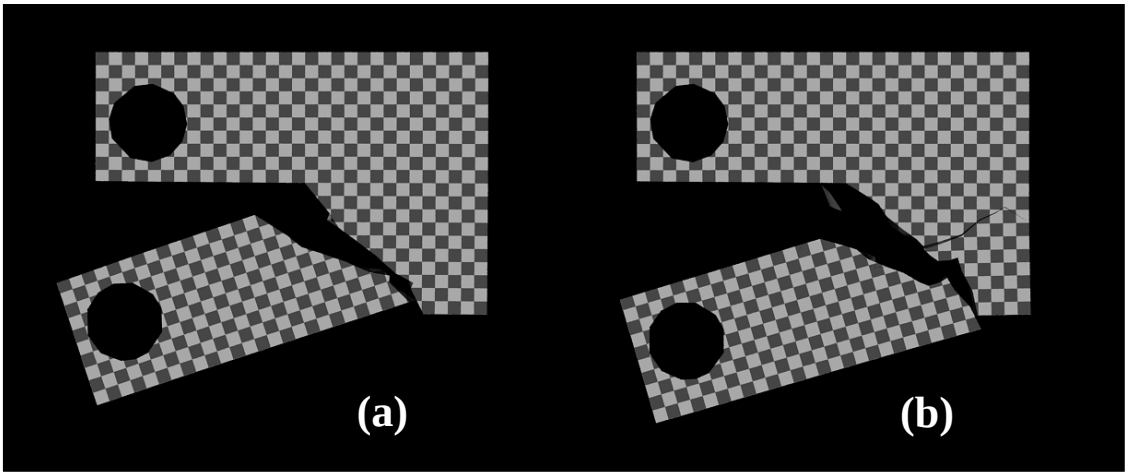


Figure 4.16: **Results of propagating the separation residual** – **(a)** The results of rerunning the example from Figure 4.13 with $\alpha = 1.0$. **(b)** The results of rerunning the large timestep example from Figure 4.14 with $\alpha = 1.0$.

advance. This phenomena, referred to as back-cracking, is undesirable and it can lead to unrealistic artifacts such as that highlighted in Figure 4.17.

The solution to this problem is to avoid creating cracks that make a sharp angle with existing surfaces. The system prevents fractures from forming if they form an angle less than $\theta_{\min}$ with an existing surface. The two-dimensional implementation of this idea is straightforward and it is illustrated in Figure 4.19. In three dimensions, it is more complex because care must be taken to ensure that the fracture surface is not broken up into discontinuous segments. This difference arises because in two dimensions the fracture surface consists of two line segments that only connect through the node, while in three dimensions the fracture surface is cross section through the tetrahedra that surround the node. As the faceted surface defined by

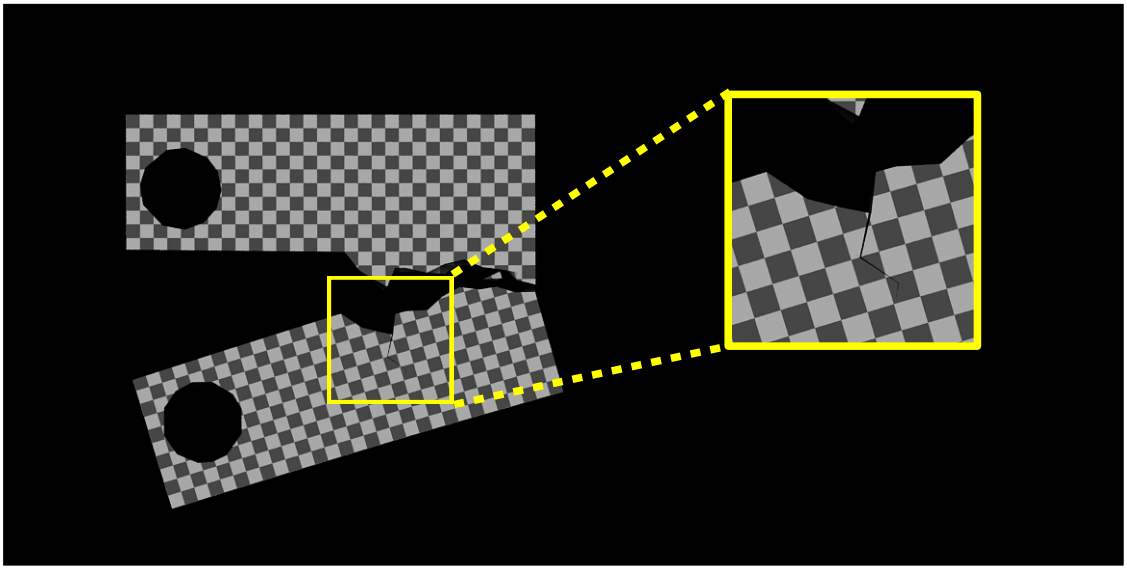


Figure 4.17: **Example of back-cracks** – Example showing how back-cracking can create undesirable artifacts that have a significant effect on results.

this cross section is modified to merge with the existing surface, simply removing facets that made too shallow an angle with the existing surface would result in discontinuities. Instead, the facets are rotated so that they form a joint between the forward portion of the fracture surface and the existing surface. The results of applying the back-crack avoidance algorithm to the cut block example is shown in Figure 4.20.

4.4 Anisotropic Parameters

Materials that deform anisotropically can be modeled using the general form of the stress to strain relationship given by (3.10), however certain materials may also

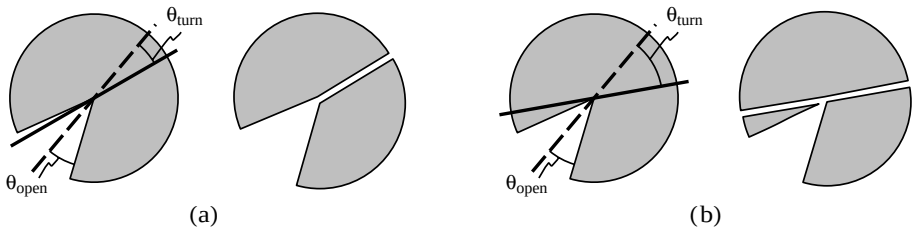


Figure 4.18: **Back-Cracking during fracture advance** – The dashed line is the axis of the existing crack. Cracks advance by splitting elements along a fracture plane, solid line computed from the separation tensor. **(a)** If the crack does not turn sharply, then only elements in front of the tip will be split. **(b)** If the crack turns at too sharp an angle, then the backwards direction may not fall inside of the existing failure and a spurious bifurcation will occur.

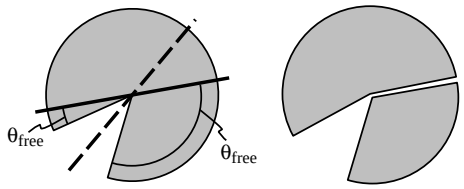


Figure 4.19: **Diagram of back-cracking solution** – When a node is split, only new surfaces that form an angle greater than or equal to $\theta_{\min}$ with existing surfaces are created.

exhibit anisotropic fracture properties, and be more prone to failing along planes oriented in a particular direction. This type of phenomena can be modeled by modifying the computation of the separation tensor so that it is non-uniformly scaled to account for the anisotropic properties of the material.

Let $\boldsymbol{R}$ be a transformation from the material coordinates system to a coordinate system that is aligned with the preferred fracture orientations of the material. Let $\boldsymbol{S}$ be a diagonal matrix with values inversely proportional to the relative toughness

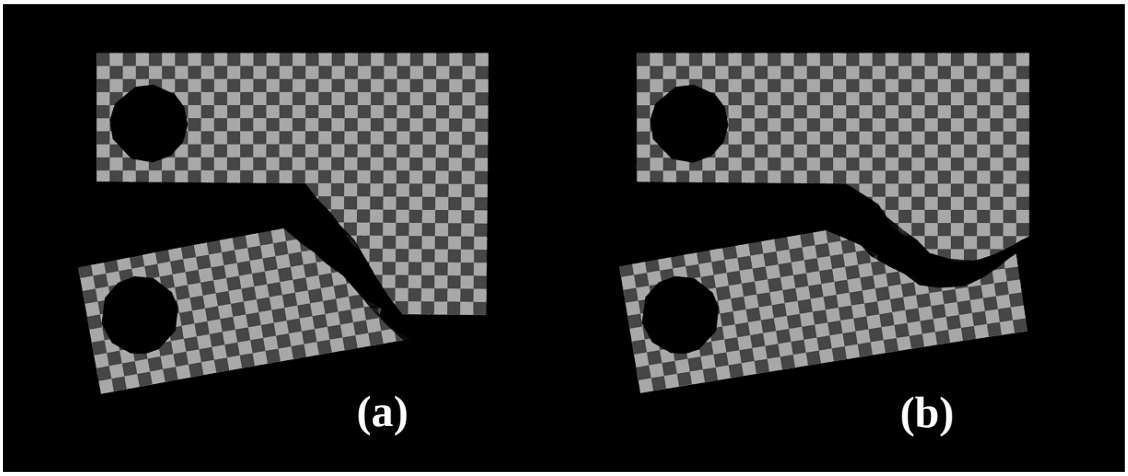


Figure 4.20: **Results of back-crack avoidance** – **(a)** The results of rerunning the example from Figure 4.16.a with back-cracking avoidance active. **(b)** The results of rerunning the large timestep example Figure 4.16.a with back-cracking avoidance active.

of the material in each of the directions defined by $\boldsymbol{R}$. For example, a material with $\boldsymbol{R} = \boldsymbol{I}$ and $\boldsymbol{S} = \text{diag}([1, 0.5, 0.5])$ would only be half as resistant to failure perpendicular to the $\hat{\boldsymbol{x}}$-axis as it would be to failure at other orientations. A modified version of the stress that accounts for the anisotropic toughness of the material can be computed that is non-uniformly scaled according to

$$\boldsymbol{\sigma}^* = (\boldsymbol{R}^T \boldsymbol{S} \boldsymbol{R}) \boldsymbol{\sigma} (\boldsymbol{R}^T \boldsymbol{S} \boldsymbol{R}) \; . \tag{4.8}$$

The scaled stress, $\boldsymbol{\sigma}^*$, is then substituted in place of $\boldsymbol{\sigma}$ in (4.6) when computing the separation tensor. The results of applying this modification to the cut-block example are shown in Figure 4.21.

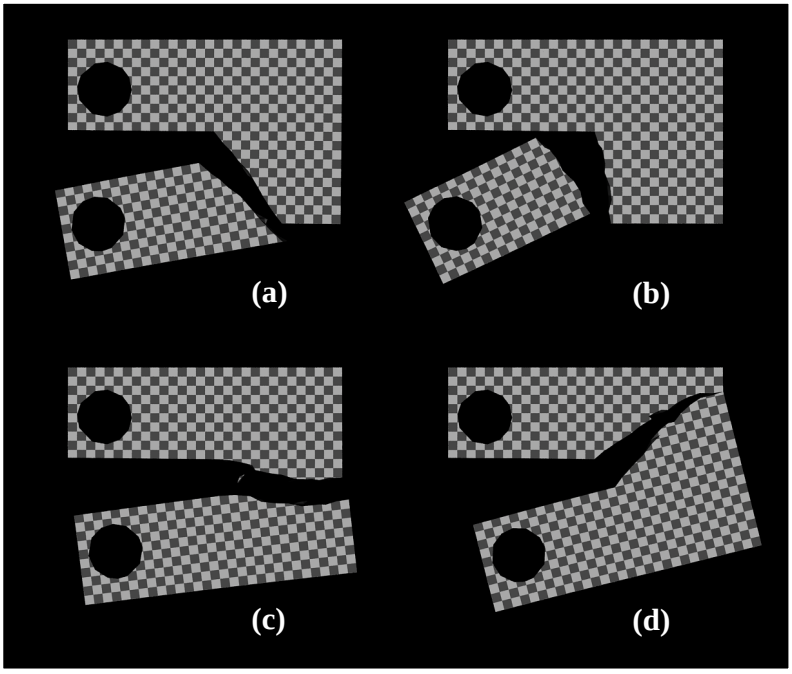


Figure 4.21: **Demonstration of anisotropic fracture** – **(a)** All scale factors are set to one and the result is identical to Figure 4.20.a. **(b)** The scale factors have been set to $[1, 0.5, 0.5]$ causing the failure to run perpendicular to the $\hat{\boldsymbol{x}}$-axis (horizontal axis). **(c)** The scale factors have been set to $[0.5, 0.5, 1]$ causing the failure to run perpendicular to the $\hat{\boldsymbol{z}}$-axis (vertical axis). **(d)** As with (c), the scale factors have been set to $[0.5, 0.5, 1]$ but $\boldsymbol{R}$ now describes a rotation of $\pi/4$ radians about the $\hat{\boldsymbol{y}}$-axis (perpendicular to image plane).

4.5 Mesh Resolution and Re-Meshing

One of the significant differences between this research and previous fracture work in computer graphics is the use of dynamic re-meshing. This approach allows high-quality results to be obtained with relatively low-resolution meshes.

Figure 4.22 shows the effect of varying the resolution of the finite element mesh used to model the block. The original mesh, with an average element width of

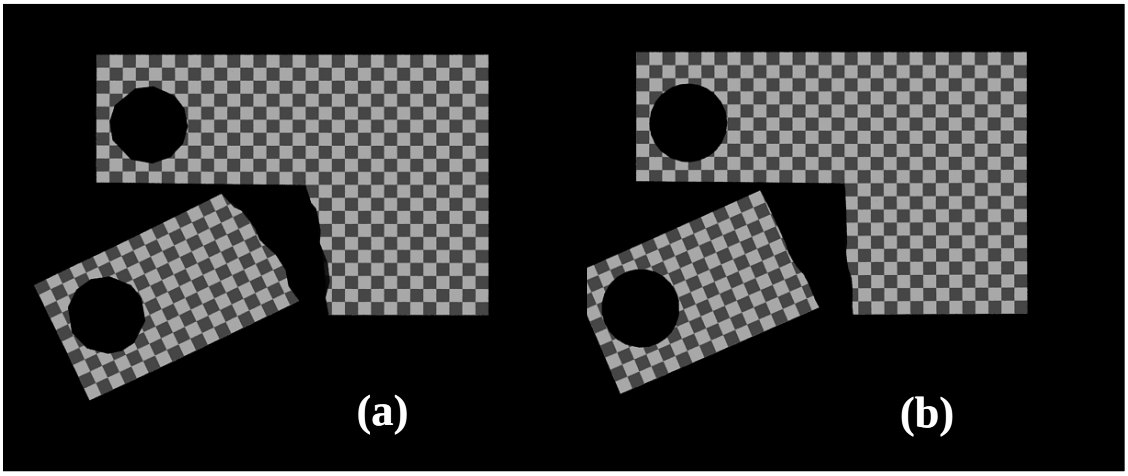


Figure 4.22: **Results using low- and high-resolution meshes** – **(a)** Result with low-resolution mesh computed with original, low-resolution, mesh. (Same as shown in Figure 4.21.b.) **(b)** Same situation recomputed using higher resolution mesh.

approximately 0.08 m, is defined by 1551 elements and 594 nodes. The higher-resolution mesh was generated with an average element width of approximately 0.04 m and is defined by 8571 elements and 2677 nodes. Although the two meshes do not generate identical results, they are qualitatively similar.

Figure 4.24 demonstrates the effect of re-meshing during crack propagation. With the dynamic re-meshing disabled, the fracture is forced to follow the original mesh boundaries, resulting in a unrealistic appearance. This example also demonstrates that the computational overhead required by the re-meshing is not prohibitive. The original simulation took 49 minutes, with re-meshing disabled the simulation required 48 minutes. (Simulation times are for computations performed in an SGI O2 with a 195 MHz R10000 processor.)

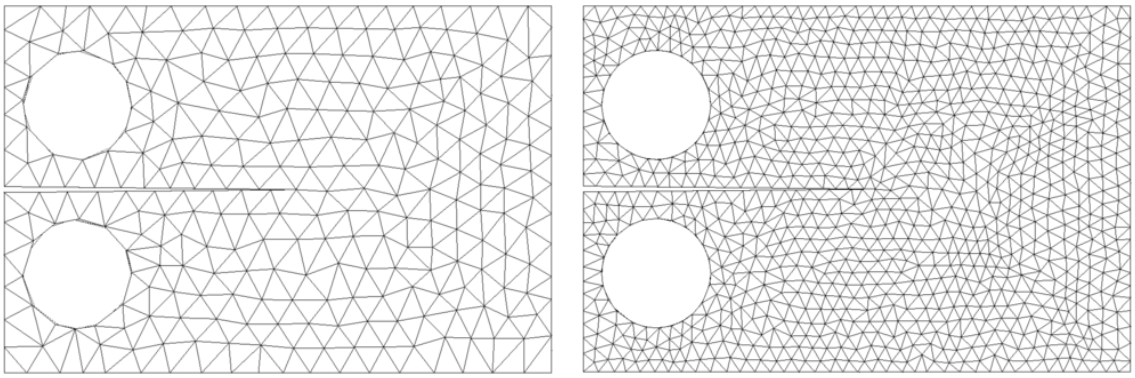


Figure 4.23: **Low- and high-resolution mesh for block** – **(a)** The original, low-resolution, mesh used in the previous examples. **(b)** The higher-resolution mesh used in Figure 4.22.b.

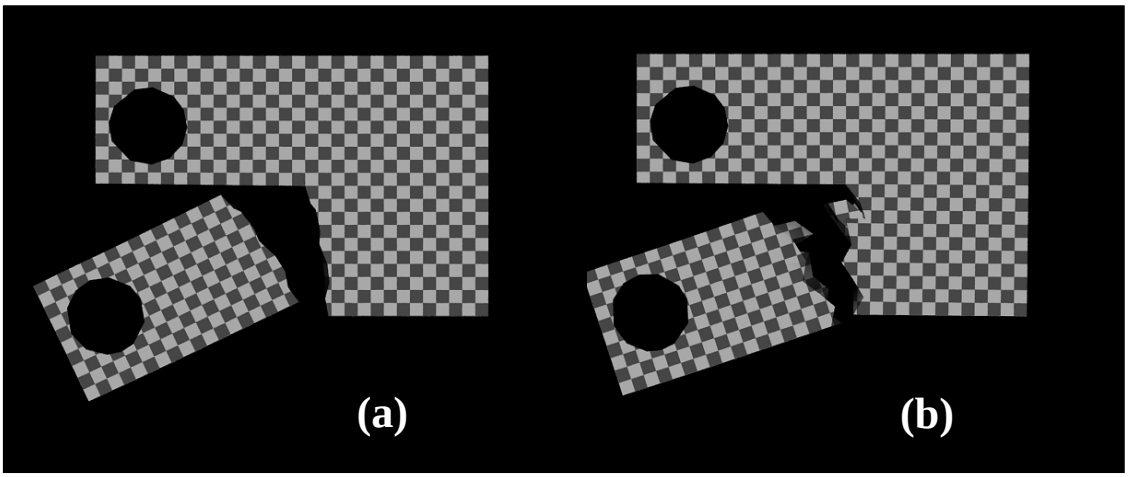


Figure 4.24: **Results with dynamic re-meshing disabled.** – **(a)** Result with re-meshing enabled. (Same as shown in Figure 4.21.b.) **(b)** Result computed with re-meshing disabled.

Chapter 5

Collisions

This chapter describes the collision methods that are used in this thesis. While the topic of collisions is somewhat tangential to the primary focus of modeling fracture, most situations that involve fracture will also involve some type of collision and the fracture model would not be complete if it could not model these situations. Computing the forces that arise due to a collision requires first determining where the collision occurs (collision detection) and then determining the forces acting at that location (collision response). Although these processes are conceptually distinct, the choices made in designing an algorithm for collision response can have a significant impact on the requirements imposed on the detection algorithm, and the reverse is also true. Collisions occur both when two different objects come into contact, and also when two separate locations on a single object come into contact. Known as self-collision, this second situation frequently occurs as an object deforms and then breaks. The following sections present a brief overview of the previous collision research in computer graphics followed by the details of the detection and response methods used here.

5.1 Previous Collision Work

Modeling physical collisions is a well-studied problem in computer graphics. The approaches that have been developed can be generally classified based on whether they prevent inter-penetration or whether they respond once a penetration has occurred. An exact collision method has advantages over one that allows some degree of penetration because once penetration has occurred situations can arise where computing an appropriate restoring force becomes ill-defined. Furthermore, when dealing with fast-moving, thin objects, an inexact method may suffer from "tunneling" problems, allowing two objects to pass completely through each other without detecting the collision.

There are, however, several drawbacks associated with exact collision methods. The first is that the collision forces must be applied at the time when the collision occurs. For numerical simulations, this requirement poses a problem because the simulation timestep must be modified so that the collision occurs on a boundary between timestep intervals. Because the moment of impact is not known *a priori*, determining when the collision occurs can involve a costly root finding procedure wherein each iteration of the procedure involves calling the simulation to compute a variable sized timestep. A second drawback is that computing the collision forces can become very expensive as the number of contacts grows. Baraff has shown

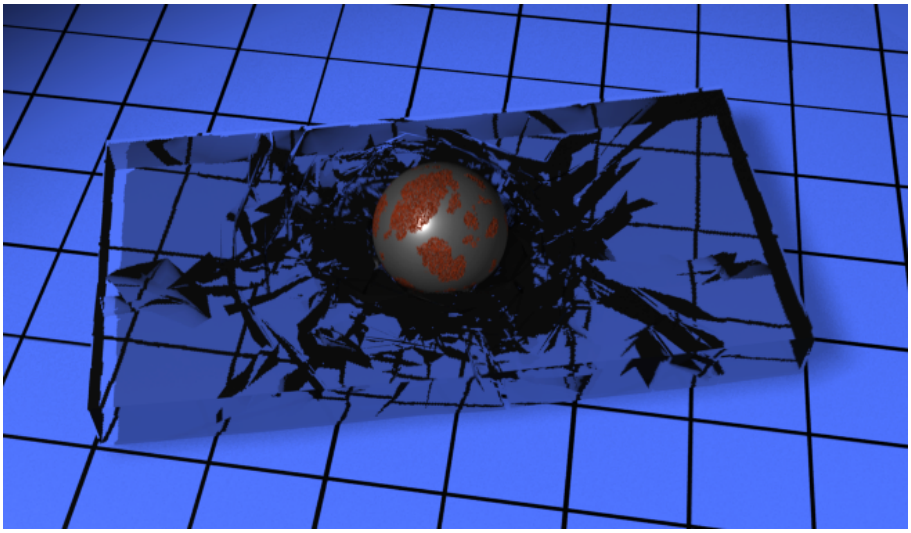


Figure 5.1: **A difficult situation for collision algorithms** – This example shows a situation that presents a challenge to existing collision methods. The ball's weight is being supported by an area of the glass slab that has been crushed into many small fragments and are now resting on top of each other. In addition to the collisions between the small fragments of glass, the contact between the material on opposite sides of a fracture must be modeled as well.

that this problem is potentially NP-Complete, depending on the types of forces that are allowed [7]. Even when using methods that make assumptions to avoid non-polynomial run times, situations that involve many hundreds of contacts, such as the one shown in Figure 5.1, can require undesirably large amounts of computation [8].

Lin and Gottschalk have published a survey article that discusses the techniques that have been developed for efficiently detecting collisions [35]. Most of these techniques focus on the problem of detecting collisions between two or more rigid body objects that are each defined by a large set of polygons. Because the structure of

each object is assumed to be constant, significant preprocessing can be done to build data structures that facilitate efficient collision queries.

The fastest methods currently available for detecting collisions between rigid body objects combine extensive preprocessing with assumptions about temporal coherence and geometric or topological properties [41, 49, 15, 36]. Unfortunately, these methods are not appropriate for modeling objects that can deform and fracture. Deformable objects violate assumptions concerning rigid body motion, rendering precomputed data structures invalid and complicating algorithms that make assumptions about temporal coherence. Detecting self-collisions is obviously incompatible with rigid body assumptions that underly these algorithms. And finally, additional complications with these algorithms arise due to topological changes created as the object fractures.

Other techniques make fewer assumptions about the type of objects they will be applied to. In general, these methods build a hierarchy that can be used to quickly eliminate large sections of a model from consideration during intersection testing. Space partitioning schemes, such as BSP-Trees or oct-trees [6], build a hierarchy based on a division of the space occupied by an object, splitting primitives as necessary along the spatial boundaries.

Alternatively, object partitioning schemes recursively separate the primitives that comprise a model into groups based on a criteria such as spatial proximity.

The result is a tree with leaves that each correspond to a primitive in the model. Each node in the tree has a bounding shape associated with it that encompasses the union of the bounds of the node's children. In the case of leaf nodes, the bound encompasses the associated geometric primitive. Different researchers have experimented with bounding shapes such as axis-aligned bounding boxes (AABBs) [29], spheres [25], object-oriented bounding boxes (OBBs) [27], convex polytopes with discretely oriented faces (k-DOPs) [30], and oriented slabs (QuOSPOs) [21].

Research by Suri, Hubbard, and Hughes has shown that in most situations, bounding schemes, in particular AABBs, provably reduce the computation that must be done to detect collisions [60]. However, the question as to which bounding shape offers the best tradeoff between tightness of fit and the computation required for an overlap test is subject of some debate, and the answer is probably highly dependent on the types of objects being tested.

Although generic space and object partitioning algorithms make fewer assumptions about the objects being tested and their motion, these methods still rely on substantial pre-processing. A limited amount of work has been done to investigate how object and space partitions can be updated as the objects they contain deform. For example, van den Bergen has investigated update methods for hierarchies of axis-aligned bounding boxes that contain deformable bodies [67]. His method up-

dates a tree by recomputing bounds at the leaf nodes and then propagating the changes up the tree to the root.

A limitation in all of these methods is that they do not perform well when detecting self-collisions. Self-collisions pose a problem because somehow a distinction must be made between primitives that are colliding and ones that are adjacent in the mesh. Algorithms that do not take this distinction into account spend the majority of their computation time repeatedly discovering mesh adjacencies that are irrelevant to collision detection.

The curvature-based method developed by Volino and Magnenat-Thalmann [69], is the notable exception to the generalization that previous methods are not well-suited for dealing with self collision. Their technique subdivides a triangulated surface into a hierarchy based on adjacency and curvature information. The resulting algorithm does not waste time detecting mesh adjacencies and the authors report a run time that is roughly proportional to the number of actual self collisions in the mesh. Although this method works very well for polygonal surface meshes, it is not obvious how to extend the method for polyhedral solid meshes.

5.2 Collision Detection

A physical simulation of interacting solid objects must detect collisions at every timestep. Naive collision detection between two objects is an $O(nm)$ algorithm that tests all n geometric primitives in an object against all m geometric primitives in the other object. For self-collisions, the naive algorithm tests all n geometric primitives in an object against all other primitives in that object and runs in $O(n^2)$ time. Other operations that must occur during each timestep are approximately $O(n)$, and collision detection can easily become the simulation's computational bottle neck, rendering anything other than very simple systems intractable. To combat this problem, some method must be used to efficiently determine which pairs of primitives are likely to be colliding so that only those pairs can be tested. The system performs the process of determining which pairs are potentially colliding using a hierarchy of axis-aligned bounding boxes. Once a set of potentially colliding primitive pairs has been found, an intersection test is performed on these primitives and the results are passed to a collision response algorithm.

5.2.1 Bounding Hierarchy

The collision detection method used in this research makes use of a hierarchy of axis-aligned bounding boxes to efficiently find potential collisions. As discussed in the

previous section, this object partitioning scheme is not particularly well-suited for objects that deform and change topology, nor is it ideal for detecting self collisions. In order to accommodate these tasks, several changes have been made to the basic algorithm.

The first step in the basic algorithm is to construct the bound tree for an object. The primitives, in this case tetrahedral elements, that make up an object are partitioned into two sets using some spatial criteria. The most common criteria is to compute the mass center and moments of the group of primitives and then split the group along a plane that passes through the center and is normal to the smallest moment. Primitives that intersect the plane are assigned to one side or the other, but not split. This process continues recursively, with each split creating a deeper level in the tree, until all groups contain only a single primitive. (See Figure 5.2.) The topology of the tree is defined by the recursion path, so that the root node corresponds to the original group, its two children each correspond to one of the groups created by the initial split and so forth down to the leaves which each correspond to a single primitive. An axis aligned bounding box is associated with each of the nodes in the tree. The leaf bounds encompass the primitive associated with the leaf node. The bounds at interior nodes encompass the union of the bounds of the node's children. The two-way split using a plane creates a binary tree, and the moment based method for selecting the split plane is intended to generate a

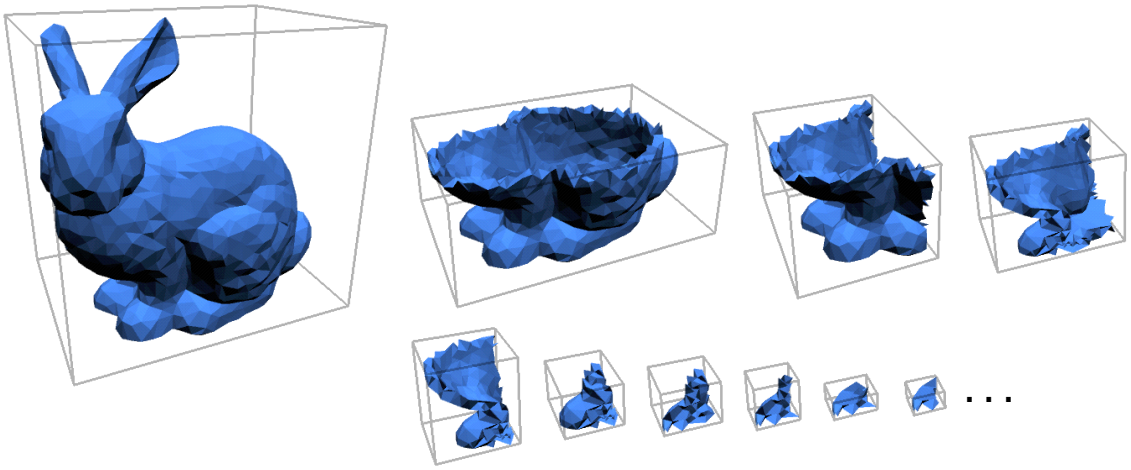


Figure 5.2: **Hierarchical spatial division of the Stanford Bunny Model** – This figure illustrates the process of partitioning a model to build a bound tree. At each split, the geometry associated with a node is divided into two groups along a plane. The figure shows only one of these groups for each split.

well-balanced tree. However, the tree need not be binary and other splitting criteria may be employed.

Once the bounding trees have been built, they may be used to determine the locations of potential collisions. This task is accomplished by processing two sets that contain pairs of nodes from the bound trees. The first set *pending* contains the node pairs that remain to be examined, the second, *found*, contains a list of leaf pairs that have been found to have overlapping bounds. The following pseudo-code describes the algorithm.

Test Bound Trees

```
1     pending := {[root of first tree, root of second tree]}
2     found := {}
3     while not empty ( pending )
4         [a,b] = remove item ( pending )
5         if bounds overlap ( a, b )
6             if are leaf nodes ( a, b )
7                 add item ( found, [a,b] )
8             else if is a leaf node ( a )
9                 foreach c ∈ children of ( b )
10                    add item ( pending, [a,c] )
11            else if is a leaf node ( b )
12                foreach c ∈ children of ( a )
13                    add item ( pending, [c,b] )
14            else
15                foreach c1 ∈ children of ( a )
16                    foreach c2 ∈ children of ( b )
17                        add item ( pending, c1,c2] )
```

The first improvement to the basic algorithm is to account for the deformation of the objects stored in the tree in a manner similar to that used by van den Bergen [67]. Ideally, the entire tree would be recomputed each time the object deforms. However, the objects will typically deform every timestep. The cost of rebuilding the tree for an object is essentially the same as doing a binary sort of the primitives, and is too large to be incurred at every timestep. However, the distance that each primitive moves in a single timestep is small and the topology of the existing tree is likely to be the same as the new one that would be built. Therefore, a linear time bottom-up traversal is performed updating only the bounds at each node. The resulting tree is still guaranteed to detect any overlaps, but it is potentially slightly less efficient than the rebuilt tree would have been. The tree is rebuilt periodically to prevent it from becoming highly inefficient. In the examples that are presented in this thesis, the trees are typically rebuilt every 1000 to 10000 timesteps which corresponds to a simulated time interval of about 1/1000 to 1/100 seconds.

This basic algorithm can be used to perform self collisions by calling it with both tree arguments set to refer to the same tree. To prevent checking all collisions twice, the two nested foreach loops are modified so that c_2 does not take on values that have already been taken by c_1 in a previous iteration. Even with this minor modification, this basic algorithm is not efficient for detecting self collisions because the children of each node will, at the very least, overlap themselves. As a result,

each node in the entire tree is visited at least once. In practice, most nodes will be visited many times, once as part of a pair with themselves, and then as part of many pairs with nodes that correspond to elements that are adjacent in the mesh.

Testing to determine that a pair of elements are mesh adjacent, as opposed to colliding, is trivial. However, this test cannot be performed until the pair of elements have actually been located; at that point the expense of traversing the tree has already been incurred. However, as previously pointed out, the mesh does not move a great deal during each timestep. As a result the self traversal performed during one timestep is likely to be the same as the one performed during the next. Thus, rather than recomputing the list of colliding pairs each timestep and then culling adjacent pairs, the list can be computed, adjacent pairs culled, and the resulting list used multiple times.

This traversal caching introduces two types of errors. First, pairs that are no longer overlapping may be left in the list, and second, pairs that have become overlapping may not be in the list. The first problem merely introduces a small inefficiency by finding additional potential collisions. The second problem is more serious as it could cause a collision to be missed.

To avoid missing collisions, the leaf bound computation is modified to grow the bounds slightly. The amount of growth is determined from the velocity of the nodes that define the element and a time interval that the bound is intended to be valid

for. In the case of an axis aligned bound on a tetrahedron, the bound intervals are computed with:

$$\mathbf{bound}_{\max} = \max_{\substack{i \in \{1,2,3,4\} \\ \gamma \in [0..\Delta t]}} (\boldsymbol{p}_{[i]} + \gamma \boldsymbol{v}_{[i]}) + \boldsymbol{\varepsilon} \tag{5.1}$$

$$\mathbf{bound}_{\min} = \min_{\substack{i \in \{1,2,3,4\} \\ \gamma \in [0..\Delta t]}} (\boldsymbol{p}_{[i]} + \gamma \boldsymbol{v}_{[i]}) - \boldsymbol{\varepsilon} \tag{5.2}$$

where Δt is the desired time interval, $\boldsymbol{\varepsilon}$ is a small padding space that allows for minor velocity changes, and $\boldsymbol{p}_{[i]}$ and $\boldsymbol{v}_{[i]}$ are the positions and velocities of the tetrahedron's four nodes as defined in Chapter 3. The effect on the bounding boxes is illustrated in Figure 5.3. Unless $\boldsymbol{\varepsilon}$ is set to a large value (thus creating an inefficient tree), the bounds are not guaranteed to be valid for the intended time interval because the node velocities are subject to change as the simulation advances. Thus at every timestep, a linear time pass over the leaf nodes is performed to determine if the bounds have been violated. If they have, then the violated leaf bounds are recomputed, changes are propagated up the tree, and cached traversals are marked as invalid. This partial update replaces the bottom-up update discussed previously.

Although the above algorithm is a significant improvement over the straightforward use of bounding hierarchies, it is still far from optimal. An approach similar to the curvature based surface segmentation approach described by Volino and Magnenat-Thalmann [69] would probably yield much better results. While it is not immediately clear how their curvature criteria for oriented surfaces could be applied

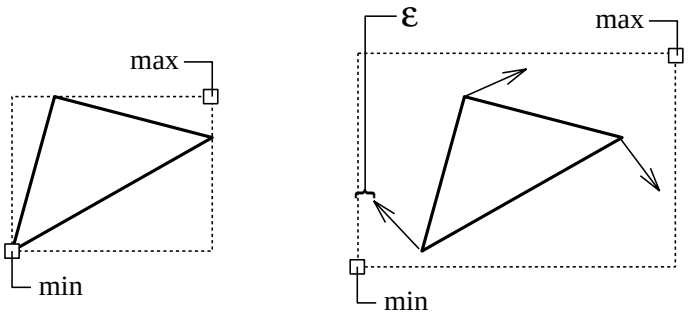


Figure 5.3: **Grown bounding boxes** – To allow the bounding boxes to be used over an interval of time, they are grown based on the velocity of the nodes and a padding distance.

to solid volumes, it may be possible to devise a similar local criteria that can be used to segment a solid object into regions that are not self colliding.

5.2.2 Tetrahedron Intersection Test

Once a set of potentially colliding pairs has been found, the actual object geometry must be tested to determine if a collision has occurred. In the current implementation, there are three possible tests than must be performed: tetrahedral element against another tetrahedral element, tetrahedral element against a plane, and tetrahedral element against a sphere. These tests correspond to testing and mesh object against itself or another mesh object, testing a mesh object against a ground plane, and testing a mesh object against a spherical object such as a wrecking ball.

To determine if two primitives are colliding, their overlap region is computed, if this region is empty then no collision has occurred. The overlap region between

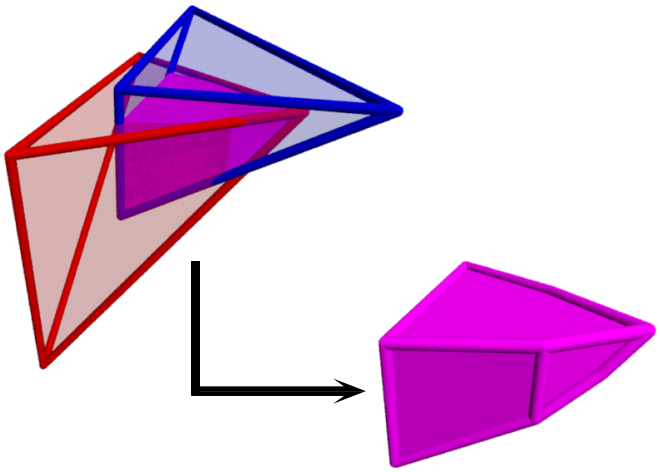


Figure 5.4: **Clipping tetrahedra against each other** – The region defined by the overlap of two tetrahedra is a convex polyhedron. Each face of the polyhedron is a subregion of one of the faces from the tetrahedra.

two tetrahedra is computed by clipping one tetrahedron against the other. As illustrated in Figure 5.4, the result of this computation is a convex polyhedron with up to eight faces. The overlap region between a plane and a tetrahedron is computed by clipping the tetrahedron against the half space defined by the plane, resulting in a convex polyhedron with up to five faces. The sphere to tetrahedron test is a little more difficult because the resulting overlap region is not a polyhedron. Rather than complicate the collision response code, the tetrahedron is clipped against the sphere and any resulting curved faces are approximated as planar.

Each of these three situations has a specialized test that can be performed quickly. For example, a tetrahedron is colliding with a plane if and only if one of the

nodes of the tetrahedron is on the negative side of the plane. However, the collision response method, discussed in the next section, will require information about the region where the two primitives overlap. Because most of the non-colliding primitives have been culled by the bounding box test, it is more efficient to perform a test that computes the overlap region than it is to use a quicker test and then compute the overlap separately.

5.3 Collision Response

Once the collision detection algorithm has determined that the current configuration contains a collision, the collision response algorithm computes a set of forces that represents an appropriate, physical response to the collision. An appropriate physical response is one that acts to separate the colliding objects, conserves linear and angular momentum, and does not add energy to the system.

The collision forces are computed using a penalty method. Penalty methods do not enforce a constraint exactly, instead a correcting force is computed once a constraint has been violated. The function for computing the correction from the constraint error is selected so that the errors remain within an acceptable bound. Penalty methods work well with detection methods, such as the one described above,

that detect penetrations rather than contact because the penetration can be used as the constraint error.

Penalty methods are often criticized for generating stiff, unstable forces that can cause difficulties for numerical integrators. However, I have not found this to be the case for the systems I have worked with. The forces generated by the penalty method are no worse behaved than the forces that are generated by the finite element model. Penalty methods have the advantages that they are fast to compute, easy to implement, and can be used to model a wide range of phenomena.

A collision force must be computed at each location where a penetration has occurred. A penetration location corresponds to a pair of overlapping tetrahedra, or a tetrahedron overlapping a sphere or plane. The force is described by three quantities: a magnitude, a direction, and a point where it is applied. In addition to the primary correcting force, a damping force and a friction force may be applied also.

The magnitude of the penalty force is proportional to the volume of the overlap region. Let $\mathcal{F} = \{F_i\}$ be the set of faces that defines the region, and let $\mathcal{P}_i = \{\boldsymbol{p}_{i,j}\}$ be the clockwise ordered list of the vertices (points) that defines each face[1]. Then

[1]Note that the notation used here, particularly the indexing notation, is distinct from that used in the previous chapters on deformation. Also, for a set, $\mathcal{S}$, $|\mathcal{S}|$ indicates the number of items in the set.

the volume of the region can then be computed using

$$\mathrm{vol}(\mathcal{F}) = \sum_{i=1}^{|\mathcal{F}|} \sum_{j=3}^{|\mathcal{P}_i|} [(\boldsymbol{p}_{i,j} - \boldsymbol{p}_{i,1}) \times (\boldsymbol{p}_{i,j-1} - \boldsymbol{p}_{i,1})] \cdot (\boldsymbol{q} - \boldsymbol{p}_{i,1}) \tag{5.3}$$

where $\boldsymbol{q}$ is an arbitrary projection point [34]. To reduce the effect of roundoff errors $\boldsymbol{q}$ is selected to be one of the $\boldsymbol{p}_{i,j}$. An additional side effect of selecting $\boldsymbol{q}$ in this way is that several of the terms in (5.3) become identically zero and need not be computed.

The direction that the collision force acts in is determined from the faces of the overlap polyhedron. In the case of two tetrahedra colliding, let one of them be designated object A and the other object B. In the case of a tetrahedron colliding with some other primitive, let the tetrahedron be object A and the other primitive be object B. The set $\mathcal{F}^A$ is the subset of $\mathcal{F}$ that corresponds to the faces of the overlap polyhedron that arose from clipping the faces of tetrahedron A. The set $\mathcal{F}^B$ is defined similarly. The direction of the collision force acting on object B is given by

$$\hat{\boldsymbol{f}}^B = \mathrm{normalize}\left(\sum_{i=1}^{|\mathcal{F}^A|} \sum_{j=3}^{|\mathcal{P}_i^A|} (\boldsymbol{p}_{i,j}^A - \boldsymbol{p}_{i,1}^A) \times (\boldsymbol{p}_{i,j-1}^A - \boldsymbol{p}_{i,1}^A) \right) \tag{5.4}$$

where $\mathrm{normalize}(\boldsymbol{x}) = \boldsymbol{x}/|\boldsymbol{x}|$. Because $\mathcal{F}^A \cup \mathcal{F}^B = \mathcal{F}$, $\mathcal{F}^A \cap \mathcal{F}^B = \emptyset$, and the sum of the area weighted face normals for a closed polyhedron is zero, therefore $\hat{\boldsymbol{f}}^A = -\hat{\boldsymbol{f}}^B$.

The location where the force is applied is the mass center of the overlap polyhedron. This location is computed with

$$\mathbf{center}(\mathcal{F}) = \frac{\sum_{i=1}^{|\mathcal{F}|}\sum_{j=3}^{|\mathcal{P}_i|}\text{vol}(\boldsymbol{q},\boldsymbol{p}_{i,1},\boldsymbol{p}_{i,j-1},\boldsymbol{p}_{i,j})(\boldsymbol{q}+\boldsymbol{p}_{i,1}+\boldsymbol{p}_{i,j-1}+\boldsymbol{p}_{i,j})}{4\text{vol}(\mathcal{F})}\ . \tag{5.5}$$

Where $\text{vol}(\mathcal{F})$ refers to the volume computation of the overlap region given in (5.3), and $\text{vol}(\boldsymbol{q},\boldsymbol{p}_{i,1},\boldsymbol{p}_{i,j-1},\boldsymbol{p}_{i,j})$ refers to the volume of the tetrahedra defined by the four points $\boldsymbol{q}$, $\boldsymbol{p}_{i,1}$, $\boldsymbol{p}_{i,j-1}$, and $\boldsymbol{p}_{i,j}$.

However, forces can only be directly applied to a tetrahedral element at its nodes. Therefore, a set of four forces that can be applied at the nodes must be found such that the net force and moment of the set is equal to the single force that is intended to be applied to the overlap center. Additionally, these forces alone should not cause the element to deform, thus they must be parallel to each other. Such a set of forces can be computed for a tetrahedron using the barycentric coordinates of the overlap center with respect to the world locations of the nodes, as shown in Figure 5.5. Given any force, $\boldsymbol{f}$, to be applied to a tetrahedron at any location, $\boldsymbol{x}$, an equivalent set of forces that can be applied to the nodes of the tetrahedron is given by

$$\boldsymbol{f}_{[i]} = b_{[i]}\boldsymbol{f} \tag{5.6}$$

where $b_{[i]}$ are the barycentric coordinates of $\boldsymbol{x}$ with respect to the world locations of the nodes. The computation of barycentric coordinates was described in Section 3.2.

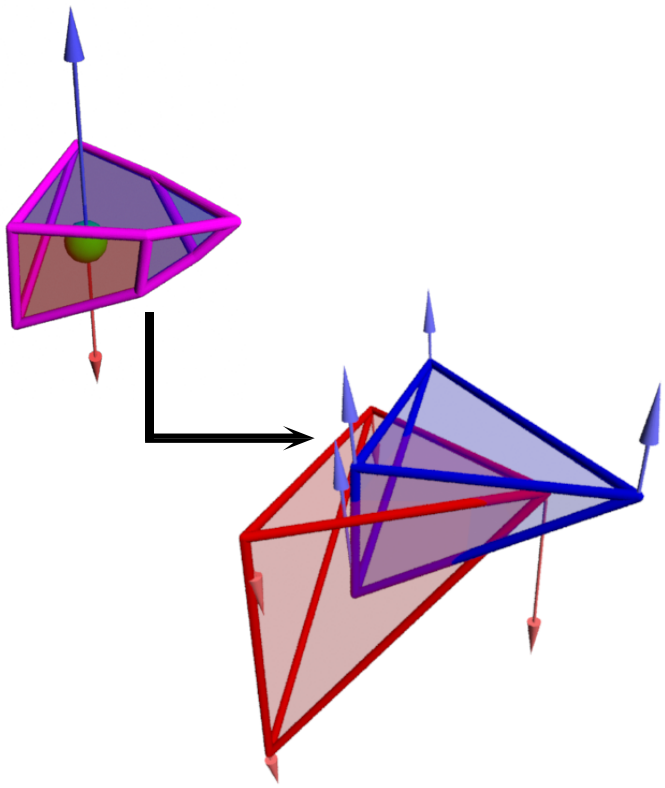


Figure 5.5: **Barycentric distribution of collision forces** – The collision force is applied as if it originated at the center of the overlap region. Although forces can only be applied to an element at its nodes, an equivalent set of forces acting on the nodes can be found to replicate the effect of a single force acting on the element at any location in space.

Once the collision response direction, the volume of the overlap, the center of the overlap, and the barycentric weights have been computed, the information can be used to compute the collision response forces. The force applied by the collision response algorithm to each object is actually a set of three forces: a error response force, a damping force, and a friction force.

The error response force, $\boldsymbol{f}_{\text{err}}$, corrects any penetration that has occurred. It is computed with

$$\boldsymbol{f}^A_{\text{err}} = k_{\text{err}} \hat{\boldsymbol{f}}^A \text{vol}(\mathcal{F}) \tag{5.7}$$

$$\boldsymbol{f}^B_{\text{err}} = k_{\text{err}} \hat{\boldsymbol{f}}^B \text{vol}(\mathcal{F}) \tag{5.8}$$

where k_{err} is a coefficient that relates overlap volume to resisting force.

The damping force models energy dissipation during the impact. It also acts in the direction given by $\hat{\boldsymbol{f}}^A$, but its magnitude is determined by a combination of the node velocities and the overlap volume. For a pair of tetrahedra, the collision damping force on object A, $\boldsymbol{f}^A_{\text{dmp}}$, is computed with

$$\boldsymbol{f}^A_{\text{dmp}} = -k_{\text{dmp}} \hat{\boldsymbol{f}}^A \text{vol}(\mathcal{F}) (\sum_{i=1}^{4} b^A_{[i]} \boldsymbol{v}^A_{[i]} - \sum_{i=1}^{4} b^B_{[i]} \boldsymbol{v}^B_{[i]}) \cdot \hat{\boldsymbol{f}}^A \tag{5.9}$$

where k_{dmp} is the dissipation coefficient, $b^A_{[i]}$ are the barycentric coordinates of the overlap center with respect to the tetrahedron from object A, $\boldsymbol{v}^A_{[i]}$ are the velocities of A's nodes, and $b^B_{[i]}$ and $\boldsymbol{v}^B_{[i]}$ are defined similarly. The force on B can be computed by swapping the A and B's in (5.9), or equivalently by just negating $\boldsymbol{f}^A_{\text{dmp}}$. If the second object, B, is not another tetrahedron, the terms of (5.9) in the parenthesis are replaced with $\sum_{i=1}^{4} b^A_{[i]} \boldsymbol{w}^A_{[i]}$ where $\boldsymbol{w}^A_{[i]}$ is the node velocity relative to B expressed in world coordinates.

The final component of the collision response force is the friction component, $\boldsymbol{f}_{\text{frc}}$. This component is computed using a simple Coulomb friction model with

$$\boldsymbol{f}^A_{\text{frc}} = -k_{\text{frc}}|\boldsymbol{f}^A_{\text{err}}|\sum_{i=1}^{4} b^A_{[i]}\bar{\boldsymbol{w}}^A_{[i]} \tag{5.10}$$

where $\bar{\boldsymbol{w}}^A_{[i]}$ is the component of the node's relative velocity, $\boldsymbol{w}^A_{[i]}$, that is perpendicular to $\hat{\boldsymbol{f}}^A$, and k_{frc} is the coefficient of friction between the two objects. To differentiate between static and dynamic friction, a different value of k_{frc} is employed if the magnitude of the result of the summation in (5.10) is near zero.

This method for computing collision forces satisfies the criteria listed at the beginning of this section. Equal but opposite forces are applied to both objects, so the net change in momentum will be zero. Because these forces are applied at the same location in space, the net change in angular momentum is also zero. As long as no tetrahedron becomes completely submerged in another object and no tetrahedron has "poked through" another object, the collision forces will direct the two objects away from each other.

However, if one of the tetrahedra does become submerged in another object then the direction of the collision force is undefined because the face normal sum in (5.4) will produce the zero vector. If a tetrahedra "pokes through" another object then the collision force may cause the objects to be pushed through each other rather than apart. Both of these situations are regarded as a type of tunneling and parameters such as the simulation timestep should be selected so that they do not occur.

The final criteria, that the collisions do not add energy to the system, is also met. The friction and damping collision forces act against velocity vectors and are therefore damping forces that dissipate energy. The error force is identical in form to a linearized gas or volume based spring, and behaves elastically so that it conserves energy. Of course, errors introduced by numerical integration could cause any of the above forces to behave unstably, but this observation is also true of the internal elastic forces.

An additional advantage of this collision method is that the resulting forces are not dependent on the resolution of the mesh used to represent the objects, but instead on the penetration volume which is a geometric quantity. Penalty methods that make use of forces generated by the penetration depth of individual nodes are resolution dependent. Additionally, certain colliding configurations do not involve node penetrations, so a node-based method is not robust.

Chapter 6

Results

This chapter presents results obtained using the methods I have developed for animating fracture. As stressed in the introduction, the goal of this research is to generate visually realistic motion as opposed to rigorously predicting a particular outcome, and because of this goal the results are primarily presented in a visual fashion. Ideally this visual presentation would be through animated sequences so that the generated motion could be viewed directly. Unfortunately, printing technology does not currently support moving images, therefore still image sequences have been used in lieu of animations. These image sequences illustrate the static appearance of the results, including the generated crack patterns, and also give a rough notion as to the dynamic appearance of the motion.

In addition to demonstrating a static and dynamic appearance that is consistent with real objects found in the physical world, the images are also intended to show the generality of the methods. The examples demonstrate several different materials under a variety of conditions.

Although a realistic visual appearance is the primary goal of this research, exactly what a "realistic visual appearance" entails is a subjective judgment. This topic will be discussed further as a possible area for future work in Chapter 7. However, in an effort to allow some form of objective assessment of the results at the current time, Section 6.3 provides a direct side-by-side comparison of high-speed video footage of situations involving breaking objects with simulation results for similar conditions.

6.1 Visual Results

The examples shown in Figures 6.1 through 6.7 illustrate some of effects that can be modeled using the methods described in this thesis. The examples have been selected to demonstrate generality and to give the reader an opportunity to subjectively access the realism of the results.

The breaking ceramic bunny shown in Figure 6.1 provides an example of the type of complex situation that this work is intended to address. Rather than a simple, controlled experiment with a single crack propagating in isolation, the figure shows a situation involving a network of many thousands of three-dimensional cracks within the solid volume of a geometrically complex object. In addition to the primary collision between the sphere and the bunny, there are many secondary collisions among the fragments of the shattered bunny.

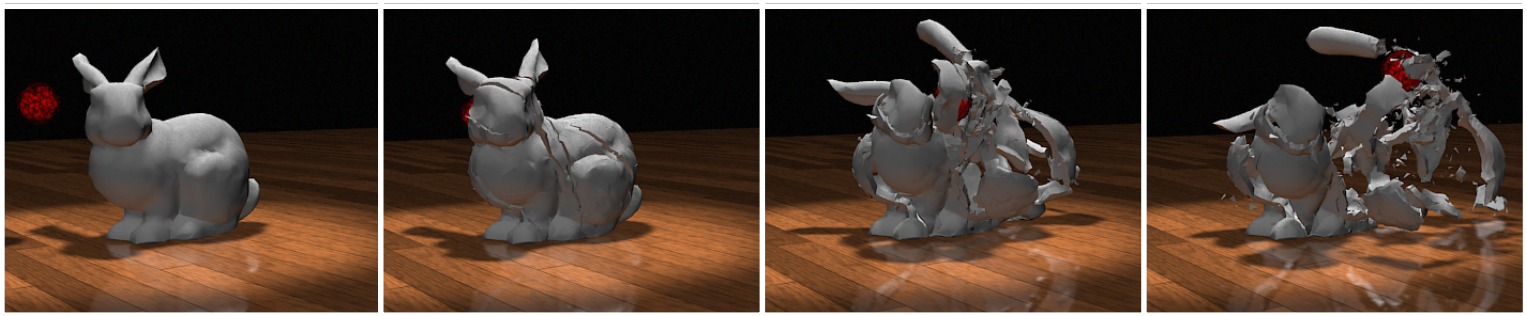


Figure 6.1: **Stanford Bunny model being shattered** – A small, spherical object moving along a circular trajectory strikes a hollow version of the Stanford Bunny model, causing it to shatter. Images are spaced 200 ms apart.

A similar situation is shown in Figure 6.2 where two similar walls have been struck by wrecking balls of different masses. Both of these figures illustrate how the methods that I have developed for animating fracture are able to produce visually realistic results by demonstrating behaviors that are characteristic of real-world events. For example, when struck by a massive wrecking ball, an adobe wall should crack and, if the force of the impact is sufficiently great, be knocked over. A more massive wrecking ball will tend to punch a hole in the wall and throw debris longer distances.

Figure 6.3 shows another situation where the simulation method produces results that are consistent with those observed in the real world. When struck by a falling weight, the slab of glass develops a characteristic pattern of radial and concentric cracks.

Figure 6.4 shows the final frames from four animations of bowls that were dropped onto a hard surface. Other than the toughness, τ, of the bowls, the four simulations are identical. The first bowl develops only a few cracks; the weakest breaks into many pieces. These images demonstrate how an individual parameter can be adjusted to achieve a range of behaviors. In this case, the effect of the parameter is relatively intuitive, however this is not always the case. Figure 6.5 shows a series of images from five animations of a ceramic tray falling onto a hard surface. Each image shows the tray after it has struck the surface but before the pieces have

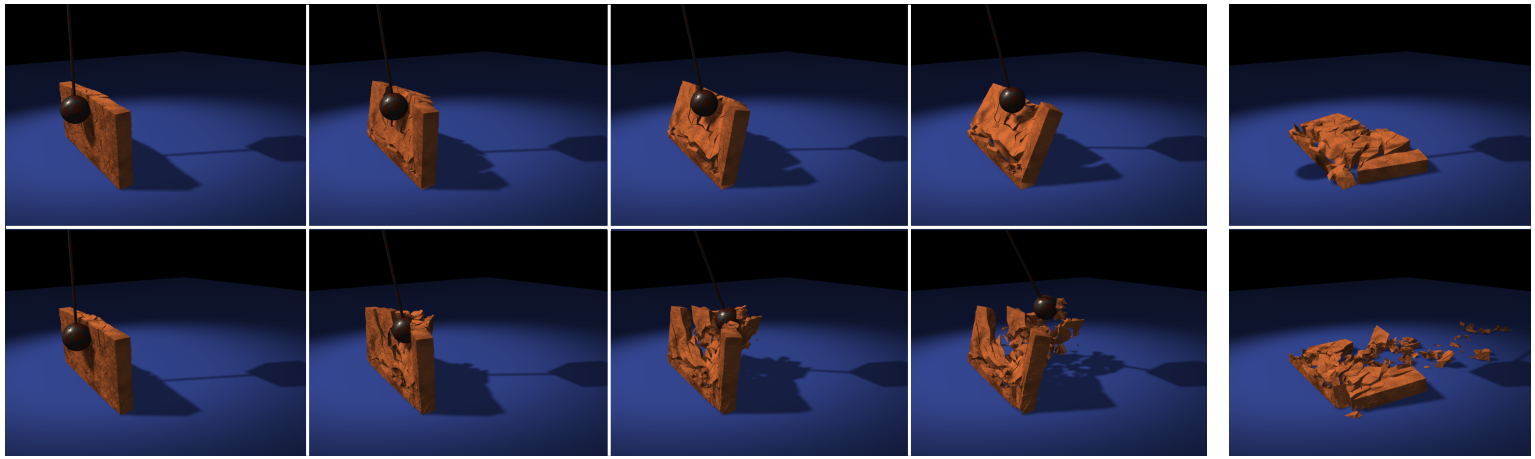


Figure 6.2: **Adobe walls struck by wrecking balls** – Two adobe walls that are struck by wrecking balls. Both walls are attached to the ground. The ball in the second row has 50× the mass of the first. Images are spaced 133.3 ms apart in the first row and 66.6 ms in the second. The rightmost images show the final configurations.

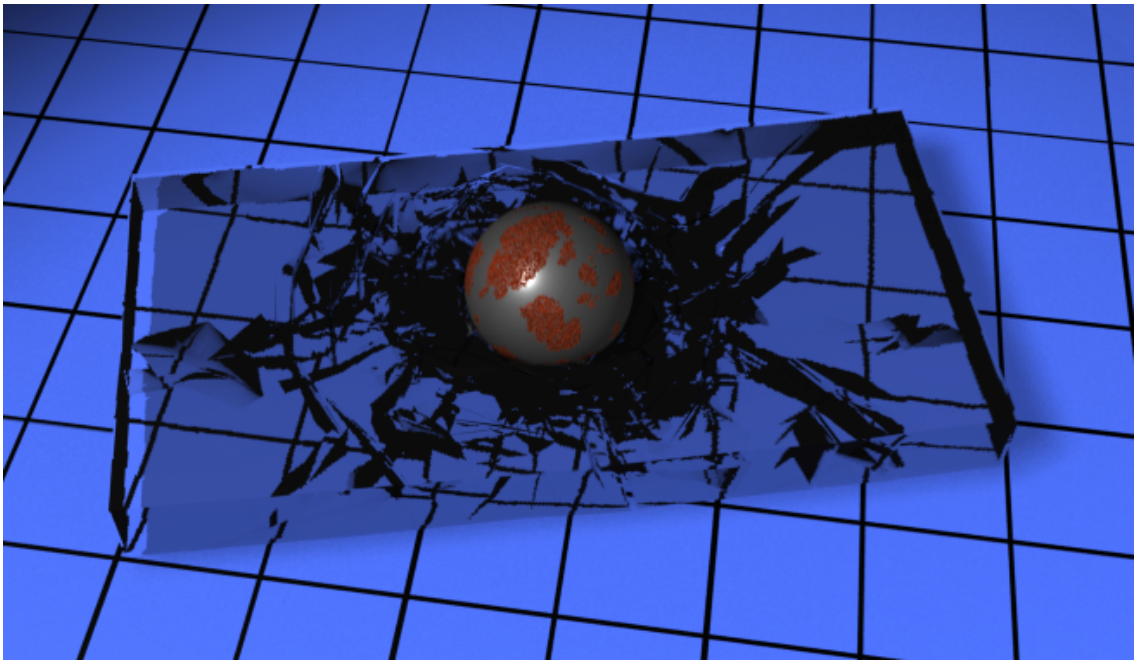


Figure 6.3: **Slab of glass that has been shattered by a heavy weight** – The pattern of radial and concentric cracks in the glass is common to real-world situations like the one shown.

scattered. The damping parameters, ϕ and ψ, have been set to successively lower values, resulting in progressively more violent impacts.

In addition to exploring the effects created as individual parameters, or a pair of related parameters, are adjusted, groups of parameters may be selected to yield a particular behavior. Figure 6.6 demonstrates some of the variety that can be achieved by showing the results of projecting a ball through sheets of different materials that are suspended in a fixed, rigid frame.

Because fracture initiation and propagation are computed from the stresses that arise in the material as it deforms, the methods I have developed can be used to

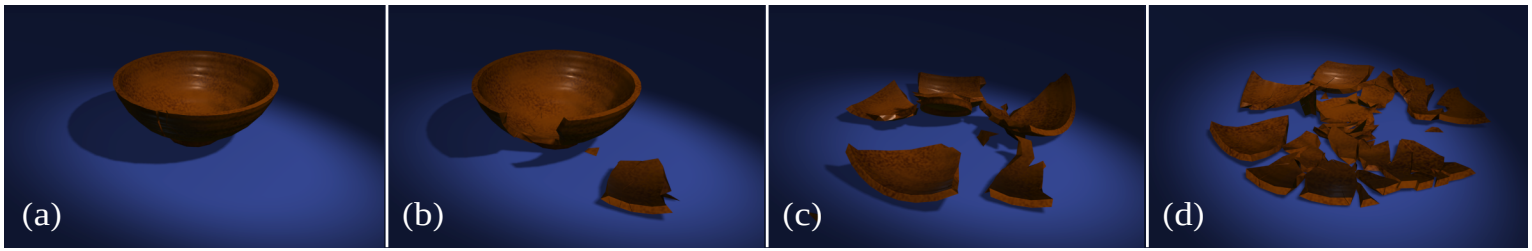


Figure 6.4: **Bowls with successively lower toughness parameters** – Each of the bowls was dropped from the same height. Other than their toughness, τ, the bowls have the same material properties.

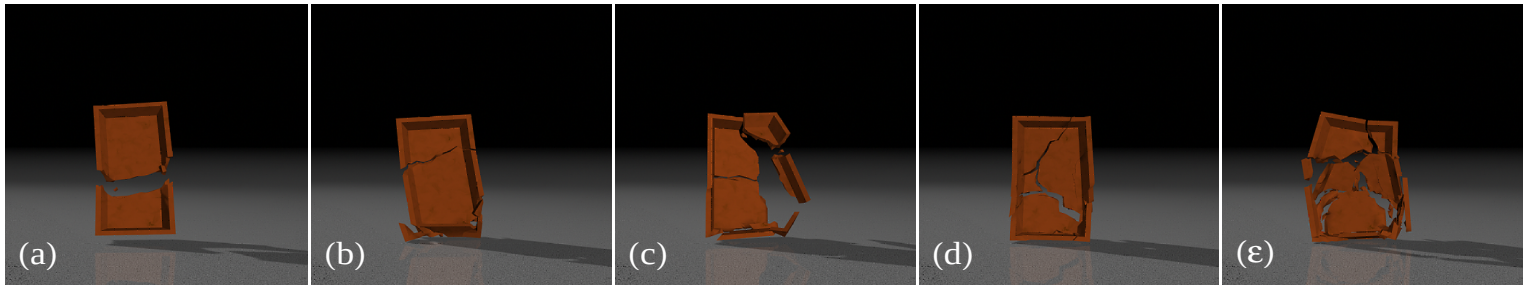


Figure 6.5: **Trays with successively lower damping parameters** – Each of the trays was dropped from the same height. As the damping parameters, ϕ and ψ, are set to lower values, the impact becomes more violent. In (e) the toughness, τ has also been reduced. The images shown have been selected to show the tray after impact and after the pieces have separated enough to show the crack pattern.

model material behavior in a wide range of situations. Some of the ones that I have demonstrated include falling on a surface, being struck by another object, and being pulled apart. Figure 6.7 shows a window being blown out by an explosion and it demonstrates another type of situation that can be modeled.

These results were not generated in real-time. The simulations were run off-line and the generated output was then rendered using a commercial rendering program[1], Pixar's RenderMan. Table 6.1 lists the amount of time that each simulation required, on average, to produce the data for one second of simulated motion.

[1]Except Figure 6.7, which was rendered using a ray-tracer described in [72].

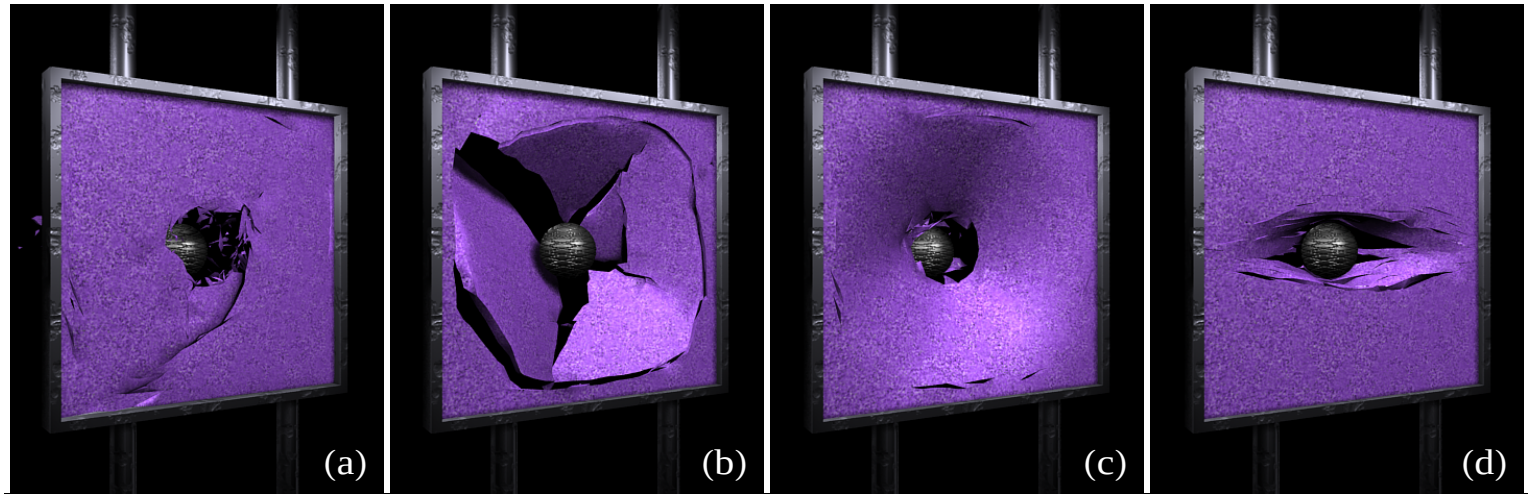


Figure 6.6: **Example range of material behaviors** – The edges of the material are fixed to a rigid frame as a projectile is hurled at the material. **(a)** A stiff, brittle material. **(b)** A light (low density), slightly flexible, but still brittle material. **(c)** A stiff material that deforms to a certain point and then begins to yield plastically. **(d)** Same material as shown in (c) but with anisotropic toughness.

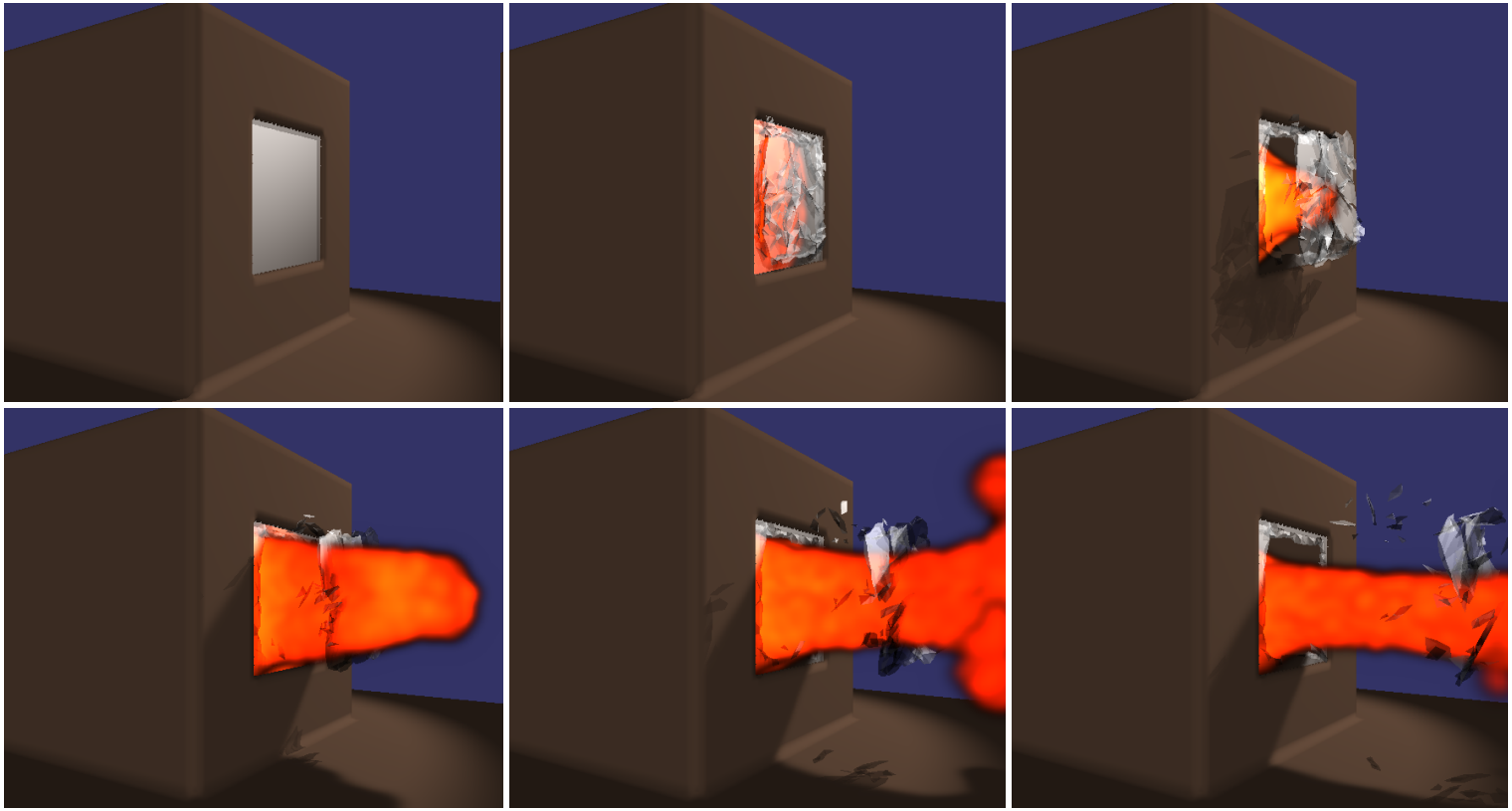


Figure 6.7: **Glass window shattered by a blast wave** – The data for the explosion was generated using a computational fluid dynamics model described in [72]. The blast wave overpressure pressure is approximately 3 atm when it reaches the window. The images show the scene at 0 ms, 13 ms, 40 ms, 67 ms, 107 ms, and 160 ms.

Table 6.1: **Simulation Times for Examples**

Example	Figure	Simulation Time	
Wall #1	6.2	399	(Min/s)
Wall #2	6.2	1098	(Min/s)
Bowl #1	6.4.a	109	(Min/s)
Bowl #2	6.4.b	115	(Min/s)
Bowl #3	6.4.c	127	(Min/s)
Bowl #4	6.4.d	156	(Min/s)
Tray #1	6.5.a	1433	(Min/s)
Tray #2	6.5.b	1561	(Min/s)
Tray #3	6.5.c	1374	(Min/s)
Tray #4	6.5.d	998	(Min/s)
Tray #5	6.5.e	1182	(Min/s)
Bunny	6.1	17280	(Min/s)
Glass	6.3	273	(Min/s)
Explosion	6.7	544	(Min/s)
Comp. Bowl	6.16	347	(Min/s)
Comp. Tray	6.17	1371	(Min/s)
The End	7.1	4665	(Min/s)

The times listed reflect the total number of minutes required to compute one second of simulated data, including graphics and file I/O. Times were measured on an SGI O2 with a 195 MHz MIPS R10K processor.

6.2 Material Library

The previous section showed that the fracture model is able to generate a wide range of effects, including many different material behaviors. Unfortunately, determining what parameters will produce a desired material behavior is not always easy. Although some parameters may have an intuitive effect on the material's behavior, many of them do not and the effect of sets of parameters changing together can be counterintuitive. This problem is made more difficult because a single parameter may influence several components of a material's behavior. For example, the internal damping parameters influence how high an object will bounce but, as shown in Figure 6.5, they also influence how the object will break.

One possible way of dealing with this problem is through the creation of material libraries. Similar to the libraries that have been created to aid in assigning material surface properties for shading, a library of physical materials would facilitate selection of simulation parameters. The entries in the library would be built either by experimentally determining the parameters of real-world materials, by looking them up in the appropriate reference, or arbitrarily by the library author. Another alternative would be to determine parameters using an optimization procedure that matched a simulation's output to a recorded event. An optimization process could

potentially yield a parameter set that was very different than those of the actual material but nonetheless generated similar behavior.

Once the library was built, users could either select a particular set of parameters by the type of material, for example "terra-cotta," or they could peruse example simulations and select a set of parameters to achieve a desired behavior. Once selected, the parameters could be adjusted to fine tune the material's behavior.

The remainder of this section presents a small example material library, both to provide an illustration of what is meant by a material library and to further demonstrate the range of behaviors that can be created using the model. Because this library is for illustrative purposes, it is fairly limited. In practice a library could contain several hundred materials along with many variations of each. In addition to a table listing the parameter values for each material, a figure exhibiting the behavior of the material is provided. Because this library is for illustrative purposes, only one example behavior has been provided for each material.

Figures 6.8 through 6.15 each show a sequence of images exhibiting the behavior of the material as it is struck by a heavy projectile. The top row of each figure shows a view from the front, and the bottom from the rear. The irregular time sampling of the images has been selected to best highlight the behavior of the material.

Tables 6.2 through 6.9 list the parameters used for each material. The density values for each of the materials were taken from *Marks' Standard Handbook for Mechanical Engineers* [5]. Where available, the Lamé constants were obtained from other references, however the values used in the simulations were reduced by a factor of $1/50$ to prevent instability. Where necessary, Lamé constants were computed from the material's modulus of elasticity and Poission's ratio using standard conversion formulae [19]. Other parameters were determined by trial and error to achieve a result consistent with the desired material. To facilitate comparison, Table 6.10 lists the parameters for all of the materials together.

Table 6.2: **Simulation parameters for "Glass" material**

Parameter	Symbol	Value	Units
Lamé Constants	λ	0.419	GPa
	μ	0.578	GPa
Damping Constants	ϕ	1.04	KPa·s
	ψ	1.44	KPa·s
Plasticity Thresholds	k_1	—	
	k_2	—	
Density	ρ	2595	kg/m^3
Toughness	τ	6.01	KN
Residual Propagation	α	0.99	

Density value from [5] for common glass.
Lamé constants $1/50^{\text{th}}$ those from [48] for soda-lime glass.

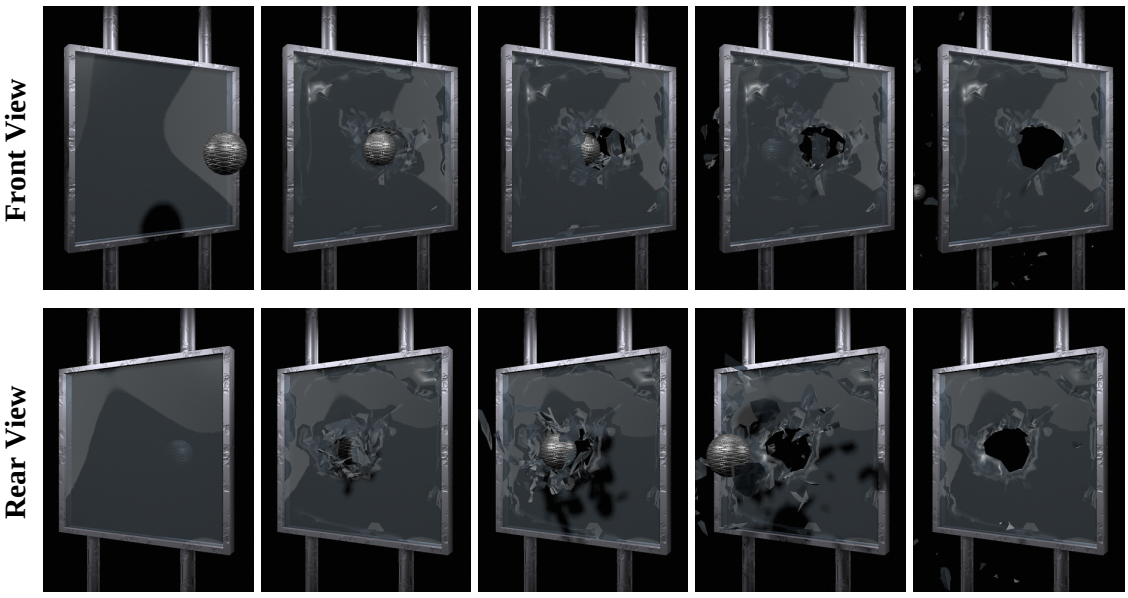


Figure 6.8: **Sample of "Glass" material**
The images show the scene at 40 ms, 106 ms, 126 ms, 173 ms, and 440 ms.

Table 6.3: **Simulation parameters for "Iron" material**

Parameter	Symbol	Value	Units
Lamé Constants	λ	0.759	GPa
	μ	1.474	GPa
Damping Constants	ϕ	18.98	KPa·s
	ψ	36.85	KPa·s
Plasticity Thresholds	k_1	0.002	
	k_2	0.211	
Density	ρ	7500	kg/m^3
Toughness	τ	24.82	KN
Residual Propagation	α	—	

Density value from [5] for wrought iron.
Lamé constants $1/50^{\text{th}}$ those from [48] for malleable iron.

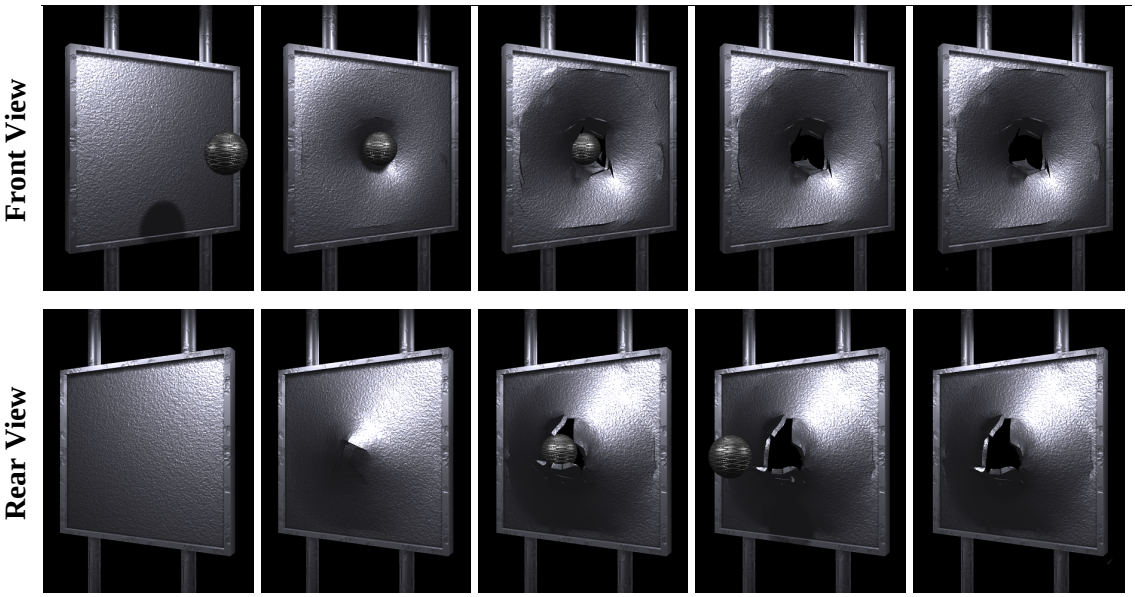


Figure 6.9: **Sample of "Iron" material**
The images show the scene at 40 ms, 106 ms, 126 ms, 173 ms, and 545 ms.

Table 6.4: **Simulation parameters for "Lead" material**

Parameter	Symbol	Value	Units
Lamé Constants	λ	—	GPa
	μ	0.593	GPa
Damping Constants	ϕ	—	KPa·s
	ψ	14.84	KPa·s
Plasticity Thresholds	k_1	0.001	
	k_2	0.991	
Density	ρ	11370	kg/m^3
Toughness	τ	11.88	KN
Residual Propagation	α	—	

Density value from [5] for lead.

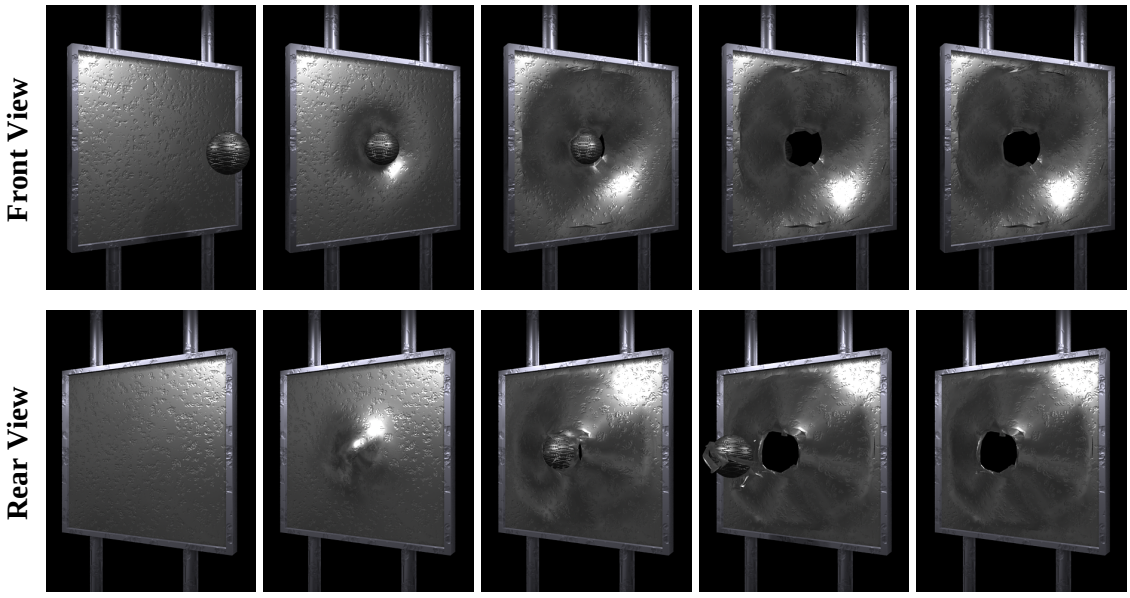


Figure 6.10: **Sample of "Lead" material**
The images show the scene at 40 ms, 106 ms, 126 ms, 173 ms, and 220 ms.

Table 6.5: **Simulation parameters for "Ceramic" material**

Parameter	Symbol	Value	Units
Lamé Constants	λ	0.320	GPa
	μ	0.484	GPa
Damping Constants	ϕ	4.03	KPa·s
	ψ	6.05	KPa·s
Plasticity Thresholds	k_1	—	
	k_2	—	
Density	ρ	2051	kg/m^3
Toughness	τ	2.09	KN
Residual Propagation	α	0.5	

Density value from [5] for hard brick.

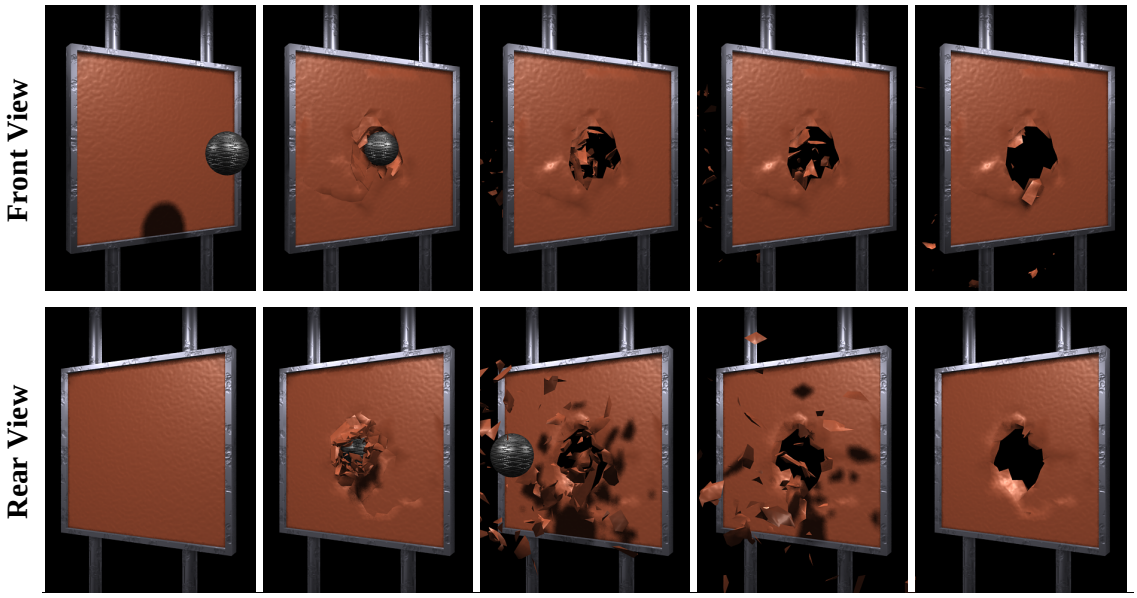


Figure 6.11: **Sample of "Ceramic" material**
The images show the scene at 40 ms, 106 ms, 173 ms, 240 ms, and 660 ms.

Table 6.6: **Simulation parameters for "Polystyrene" material**

Parameter	Symbol	Value	Units
Lamé Constants	λ	0.0014	GPa
	μ	0.0009	GPa
Damping Constants	ϕ	0.037	KPa·s
	ψ	0.025	KPa·s
Plasticity Thresholds	k_1	—	
	k_2	—	
Density	ρ	46.45	kg/m^3
Toughness	τ	0.14	KN
Residual Propagation	α	0.99	

Density value from [5] for foamed polystyrene.

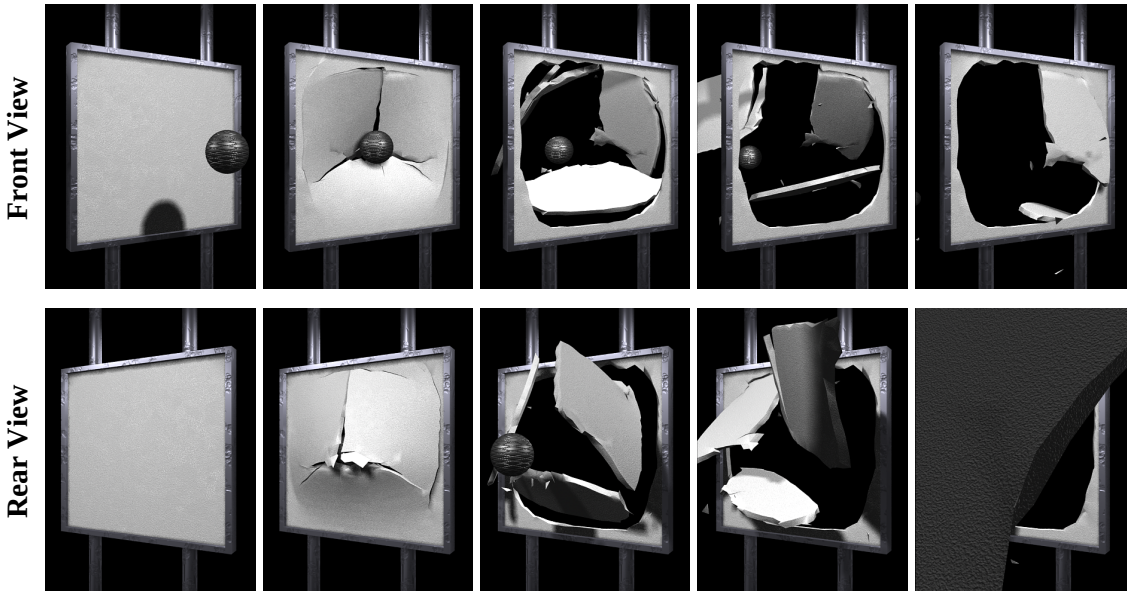


Figure 6.12: **Sample of "Polystyrene" material**
The images show the scene at 40 ms, 106 ms, 173 ms, 240 ms, and 480 ms.

Table 6.7: **Simulation parameters for "Soft Vinyl" material**

Parameter	Symbol	Value	Units
Lamé Constants	λ	0.0378	GPa
	μ	0.0252	GPa
Damping Constants	ϕ	0.945	KPa·s
	ψ	0.630	KPa·s
Plasticity Thresholds	k_1	0.0002	
	k_2	0.98	
Density	ρ	1580	kg/m^3
Toughness	τ	3.35	KN
Residual Propagation	α	0.99	

Density value from [48] for poly vinyl chloride.

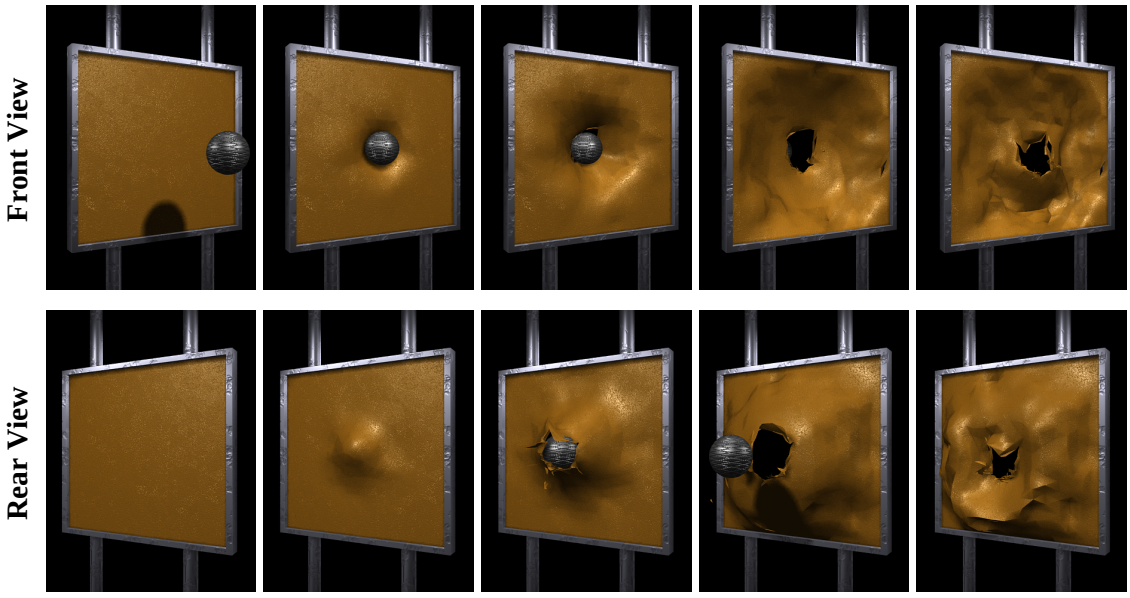


Figure 6.13: **Sample of "Soft Vinyl" material**
The images show the scene at 40 ms, 106 ms, 126 ms, 173 ms, and 220 ms.

Table 6.8: **Simulation parameters for "Hard Vinyl" material**

Parameter	Symbol	Value	Units
Lamé Constants	λ	0.0378	GPa
	μ	0.0252	GPa
Damping Constants	ϕ	0.945	KPa·s
	ψ	0.630	KPa·s
Plasticity Thresholds	k_1	0.09	
	k_2	1.49	
Density	ρ	1580	kg/m^3
Toughness	τ	3.35	KN
Residual Propagation	α	0.99	

Density value from [48] for poly vinyl chloride.

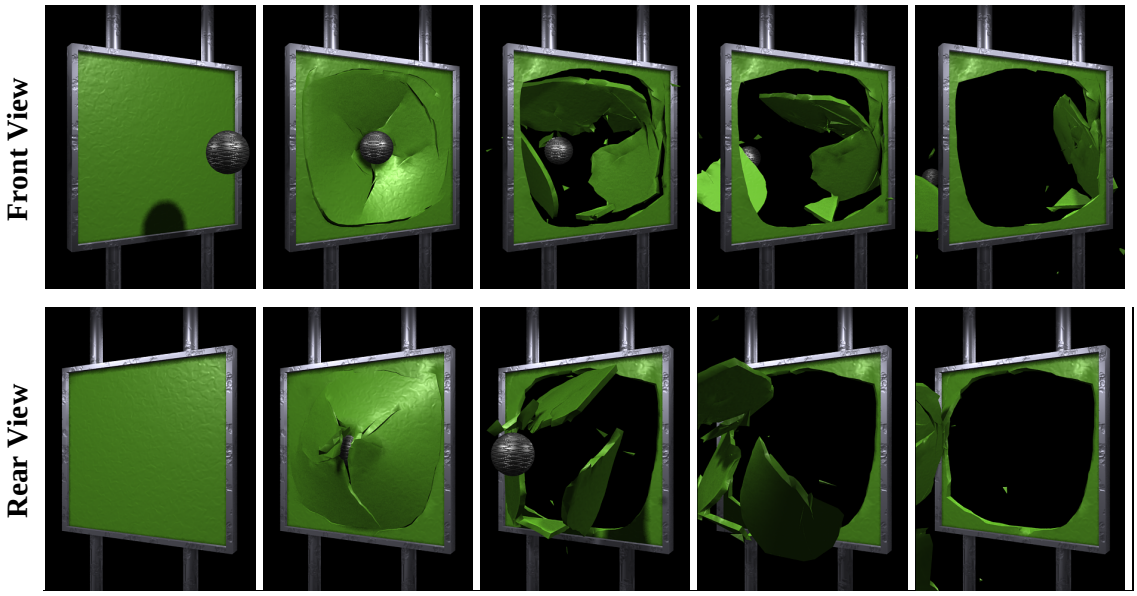


Figure 6.14: **Sample of "Hard Vinyl" material**
The images show the scene at 40 ms, 106 ms, 173 ms, 240 ms, and 366 ms.

Table 6.9: **Simulation parameters for "Rubber" material**

Parameter	Symbol	Value	Units
Lamé Constants	λ	0.0335	GPa
	μ	0.0224	GPa
Damping Constants	ϕ	2.51	KPa·s
	ψ	1.65	KPa·s
Plasticity Thresholds	k_1	0.0102	
	k_2	0.0252	
Density	ρ	2100	kg/m^3
Toughness	τ	2.10	KN
Residual Propagation	α	0.99	

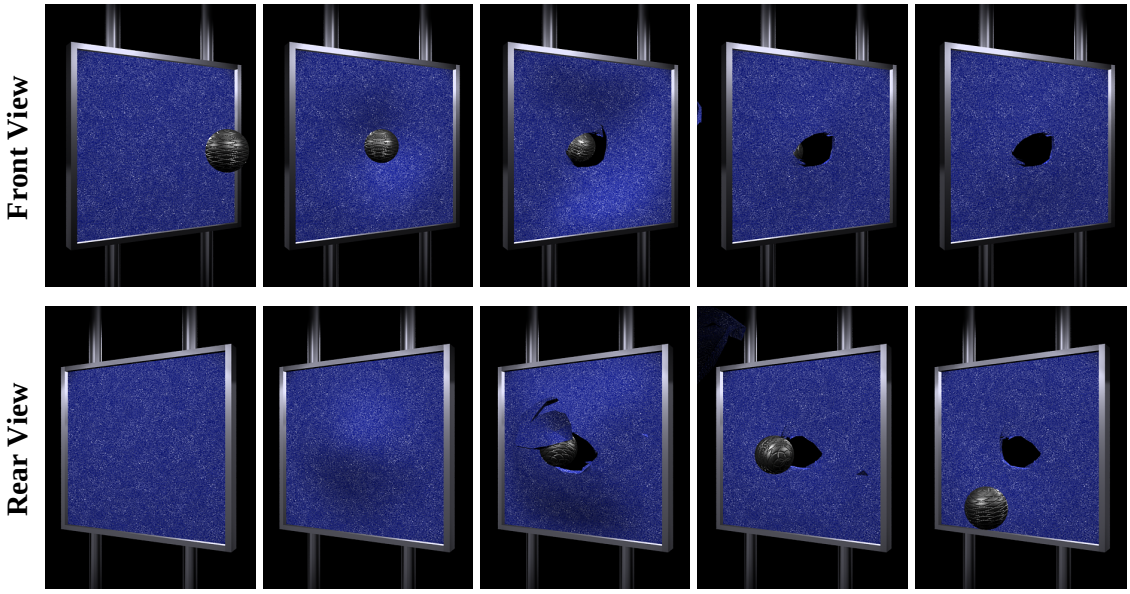


Figure 6.15: **Sample of "Rubber" material**
The images show the scene at 40 ms, 106 ms, 173 ms, 240 ms, and 393 ms.

Table 6.10: **Comparison of parameters from Material Library**

Parameter	Symbol	Glass	Iron	Lead	Ceramic	Polystyrene	Soft Vinyl	Hard Vinyl	Rubber
Lamé Constants	λ	0.419	0.759	—	0.381	0.0014	0.0378	0.0378	0.0335
	μ	0.578	1.474	0.593	0.575	0.0009	0.0252	0.0252	0.0224
Damping Constants	ϕ	1.04	18.98	—	4.79	0.037	0.945	0.945	2.51
	ψ	1.44	36.85	14.84	7.18	0.025	0.630	0.630	1.65
Plasticity Thresholds	k_1	—	0.002	0.001	—	—	0.0002	0.09	0.0102
	k_2	—	0.211	0.991	—	—	0.98	1.49	0.0252
Density	ρ	2595	7500	11370	2435	46.45	1580	1580	2100
Toughness	τ	6.01	24.82	11.88	2.48	0.14	3.35	3.35	2.10
Residual Propagation	α	0.99	—	—	0.5	0.99	0.99	0.99	0.99

6.3 Comparisons

The goal of this work is realistic animation of fracture. However assessing subjective quantities such as realism is difficult. One possible way to do so is to compare the computer-generated results with images from the real world. Figure 6.16 shows high-speed video footage of a physical bowl as it falls onto its edge compared to a simulated approximation of the scene. Although the two sets of fracture patterns are clearly different, the simulated bowl has some qualitative similarities to the real one. Both initially fail along the leading edge where they strike the ground, and subsequently develop vertical cracks before breaking into several large pieces.

A second comparison, of a real and simulated terra-cotta flower tray, is shown in Figure 6.17. Unlike the bowl comparison, the gross motion of the real and simulated trays after impact is quite different. The real tray bounces slightly and then tips backwards. The simulated tray bounces and then twists to the side before falling over backwards. The twist causes the two results to appear dissimilar. However, closer examination reveals similarities. In both, the lip of the tray breaks off along the leading edge where it strikes the ground. Additional portions of the lip break off along the sides of the tray, going slightly further up on the (viewer's) right-hand side, which strike the ground first. Both trays also crack roughly in half when they fall over and settle onto the ground.

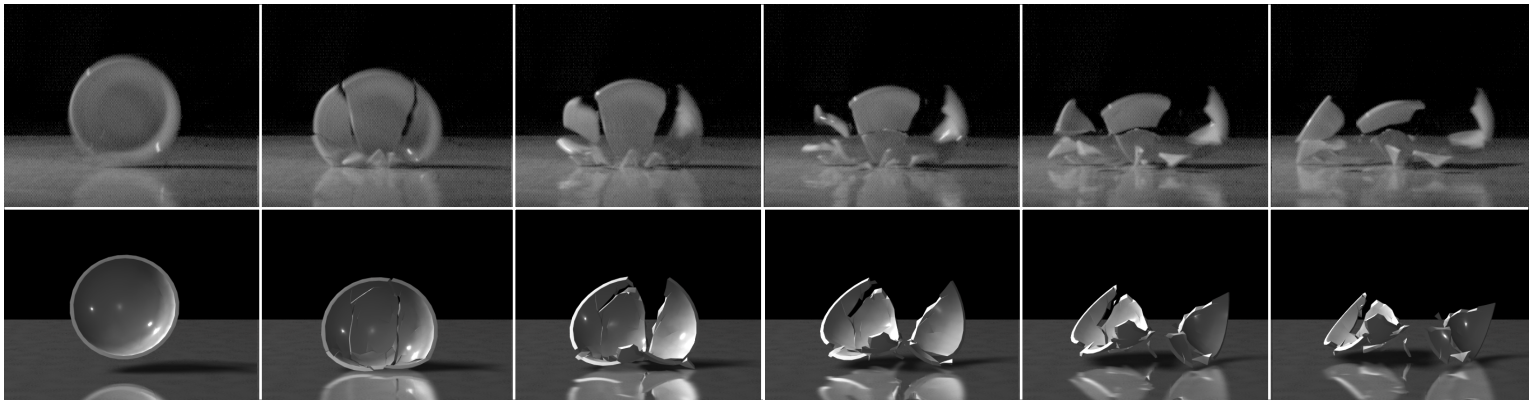


Figure 6.16: **Comparison of a real-world event and simulation** – The top row shows high-speed video images of a physical ceramic bowl dropped from approximately one meter onto a hard surface. The bottom row is the output from a simulation where I attempted to match the initial conditions of the physical bowl. Video images are 8 ms apart. Simulation images are 13 ms apart.

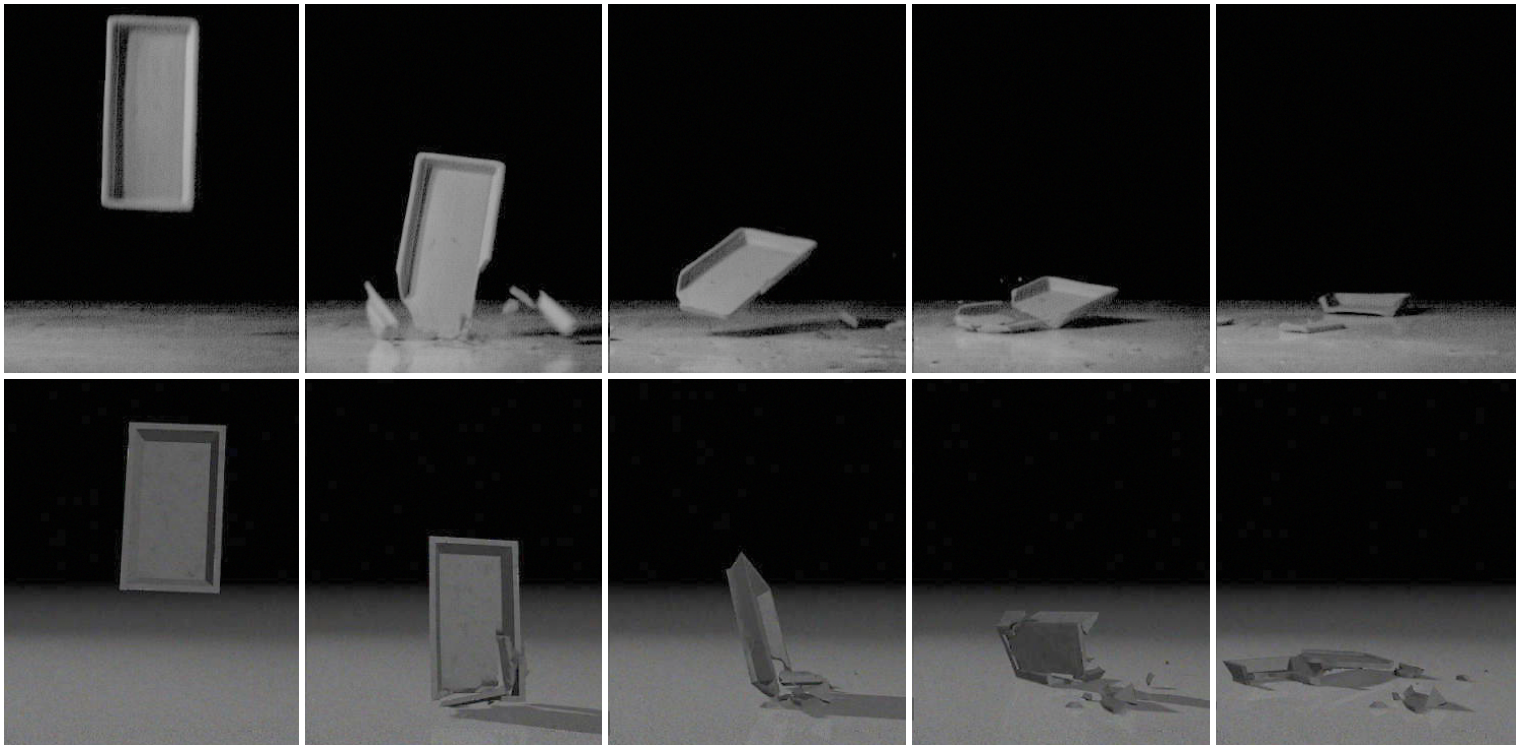


Figure 6.17: **Comparison of a real-world event and simulation** – Video images are 200 ms apart. Simulation images are 166 ms apart.

Chapter 7

Conclusions and Future Work

One of the fundamental goals of computer graphics is to develop realistic models of the world around us. Rendering and geometric modeling address some of the tasks that comprise this larger goal, but without motion the best that we can hope to accomplish is a static, unmoving shadow of the dynamic world we live in. As a result, developing methods for generating realistic synthetic motion for objects in three-dimensional animated environments is an intrinsic part of computer graphics. Designing the technology to realistically model the motion of the physical world is an enormous undertaking, and the research presented in this thesis addresses one particular piece of this task: fracture in solid objects.

The solution method I have developed possesses qualities that make it well suited for use in computer graphics applications. The dynamic re-meshing detailed in Chapter 4 is a fundamental improvement over previous techniques used in computer graphics. It allows smooth, visually realistic fracture surfaces to be generated from relatively coarse meshes, thus increasing the quality of the final results while lowering the amount of computation that would otherwise be required to produce them.

Another useful property is that my fracture algorithm determines where fractures should initiate and how they should propagate using information about how an object has been deformed, rather than employing *ad hoc* heuristics that are specific to a particular situation. As a result, the method can be applied to a wide range of physical situations. In Chapter 6, I demonstrated this quality by providing examples showing breaking objects in several different scenarios. Furthermore, the physically based nature of the fracture criteria allows it to be easily integrated with other interesting material behaviors, such as plasticity. As a result, in addition to modeling the behavior of objects under different conditions, the model is able to realistically generate the behavior of a wide range of materials using the same fracture criteria. This versatility was also demonstrated with the examples in Chapter 6.

Although I feel that my methods represent a significant advance in the graphical modeling and animation of fracture, I also recognize that ample room for improvement still remains. This fact is clearly demonstrated by the comparisons shown in Figures 6.16 and 6.17. The real materials are stiffer than the ones I was able to simulate, and as a result the real impacts have a more violent quality that is not completely captured by the simulations. The real footage also demonstrates a fineness of detail that goes well beyond what the simulation can represent; small fragments and dust are created in the real-world events while the sizes of the small-

est fragments in the simulations are limited by the re-meshing thresholds imposed by stability considerations.

The amount of computation time required by my methods is another significant limitation. Currently, simulations like the ones shown in my examples can take anywhere from a few minutes to several hours to compute each second of simulated motion. Ideally, I would like to have algorithms that could generate the motion at real-time speeds, making the methods applicable to interactive applications such as video games. Certainly, computers will become faster with time, but as they do so the complexity of the models and scenes that users expect will also increase. As a result, achieving real-time performance must be regarded as more than just a question of acquiring faster computers.

Many of the immediate directions for future work involve improving the speed of my simulation methods. Currently collision detection consumes the majority of the computation time. At first glance, this cost may appear to be an artifact of poor implementation because many efficient collision detection methods have been developed by other researchers. However, as discussed in Chapter 5, these efficient methods do not adequately address the problems that arise in detecting the self-collisions within a mesh that can deform and change its topology. In Chapter 5, I described a method I have developed that contains several incremental improvements over previous methods, but its performance still leaves ample room for further

improvement. Developing efficient and robust methods for detecting self collision remains an area for future work.

Another area for future work that relates to computation time is decoupling the graphical model used for rendering and the physical model used by the simulation. Models that are used for rendering often contain very high levels of surface detail. For example, the bunny model shown in Figure 6.1 has many small polygons that were placed there by the artist to create the impression of fur. While these fine details play a role in determining the appearance of the model, their effect on the physical behavior of the model is not noticable because they have no significant impact on the deformation of the object and their influence over where fractures will form is not likely to be noticeable to a human viewer. Despite the fact that they do not have a visible effect on the behavior of the object, their presence has a very large impact on the speed and stability of the simulation. The large number of triangles used to represent these details results in more tetrahedra whose motion must be computed, and the small size of the tetrahedra tends to decrease the numerical stability of the system so that it requires a smaller integration timestep. The net result is that a these dynamically insignificant features greatly increase the cost of the simulation. A technique that could determine which details are dynamically insignificant, remove those details, compute the motion, and then return the details to the results as they are rendered would be very useful.

The ccurrent implementation can switch between either a forward Euler integration scheme or a second order Taylor integrator. Both of these techniques are explicit integration schemes, and subject to stability limits that require very small time steps for stiff materials. Another possible way to decrease the amount of time required by the simulation might be through the use of semi-implicit[1] integration schemes rather than the explicit methods I have used. For certain types of simulation, such as cloth, other researchers have demonstrated performance that is faster by a factor of two to three orders of magnitude [9]. This speedup results from the tendency of semi-implicit methods to exhibit integration errors as artificial damping. In contrast, errors with explicit integration methods tend to add energy to the system, causing it to go unstable. The same methods used to enhance the speed of cloth simulations could also be applied to fracture simulations provided there are no adverse interactions between the large timesteps and fracture propagation. One possible type of adverse interaction could be aggravating the crack speed related artifacts discussed in Section 4.2.

[1]In the graphics literature there is some confusion about the terms explicit, implicit, and semi-implicit when applied to numerical integrations. Explicit schemes only require a method to compute the derivative (perhaps multiple derivatives) of the current state. In contract, implicit methods require inverting the function that relates state to the derivative of the state. The potential payoff with an implicit method is unconditional stability, but if the system is non-linear (which most interesting systems are) then inverting the derivative function tends to be infeasible. Semi-implicit methods approximate the inverse of the derivative function by linearizing around the current state with the Jacobian of the state derivative function. Semi-implicit methods are not unconditionally stable, but they are much more stable than an explicit method so long as integration steps remain within the region where the linearization is a good approximation. Of course, the fact that the system remains stable says little about the magnitude of the errors being incurred. See [51] for further details.

In addition to providing the direct benefit of saving time, faster simulation methods would also address the previously mentioned problem of the simulated materials being more compliant than the real-world materials. The simulated versions of materials, such as glass, are not as stiff as they should be because simulating such stiff materials with the current method would require very small timesteps. These small timesteps cause very long simulation times, making simulations involving the stiff materials too time consuming to be feasible. If the techniques described above provided a speed up of about 50× then simulated materials with stiffnesses comparable to those of real materials could be modeled at the speeds that simulations using the current materials now run at.

Faster simulation methods would not be particularly effective at addressing the problem of modeling small fragments and dust. Finite element methods are best suited for modeling objects that will demonstrate some significant amount of deformation. As a result, small fragments and dust particles that do not deform appreciably are not well suited to being modeled with a finite element model. However, they are well suited to being modeled with techniques such as particle systems, and finding a principled way to couple a particle-based model to a finite element model could provide a solution to this problem.

The issue of modeling these small fragments actually belongs to part of a larger problem: no one simulation method suits modeling all the various phenomena that

are present in the real world. For example, Lagrangian finite element models, like the one used in this research, excel in representing solids, but they do not perform well for fluids. Other simulation methods, such as particle systems and Eulerian finite differencing, model fluids well but not solids. Thus a model of any complex environment must be comprised of sub-components that utilize different simulation methods. The explosion image in Figure 6.7 illustrats a multi-component model. The simulation used to generate the images uses a finite element fracture model for the window and a fluid dynamics model for the explosions [72]. In addition to developing the component simulation methods, coupling multiple systems together raises the question of how the interfaces between dissimilar systems can be modeled. Some of my previous work addresses this issue to a limited extent [47], but it is far from a solved problem. In particular, my previous work only looked at the issue of how forces are communicated between two systems. In addition to exploring that issue further, the question of transitioning material from one model to another should be investigated. For example, in the case of small fragments described above, moving the material that comprises a fragment from the finite element system a particle system constitutes a type of coupling between the two systems that is distinctly different than a coupling generated through the exchange of forces.

Another important area for future work lies in improving the usability of physically based animation methods. Chapter 6 discussed the construction of a library of

predefined materials as a way to facilitate selecting simulation parameters. Such a library would be very useful for specifying the general properties of a material's behavior, but it would not help with subsequent fine tuning. Adapting techniques for computer-guided parameter specification, such as design galleries [39], might provide an intuitive way to allow users to adjust simulation parameters. Another possibility would be using simulation steering techniques such as those proposed by Chenney and Forsyth [14] and by Popovic and his colleagues [50].

Beyond determining material parameters, simulation steering techniques would allow users to specify a specific outcome while still having the computer fill in the details of the motion in a way that looks realistic. Simulation steering techniques have already been developed for simple systems [14, 50], but they have not yet been adapted to more complex simulation methods that would involve larger search spaces and trial times. Given the inherent difficulty in controlling physical simulations directly, simulation steering techniques will, no doubt, be an important area for future work.

Throughout this thesis, one of the goals I have focused on has been achieving "physically realistic" looking motion. However, I have not provided a concrete definition of what it means to appear realistic. Certainly, we each have an intuitive definition of what it means for a particular motion to look real, but intuitive definitions do not lend themselves to creating objective metrics and most of the results

that I have presented can only be judged subjectively. In Section 6.3, I attempted to address this issue by providing side-by-side comparison of my simulated results with real-world footage, but those comparisons still rely on subjective assessment by the viewer.

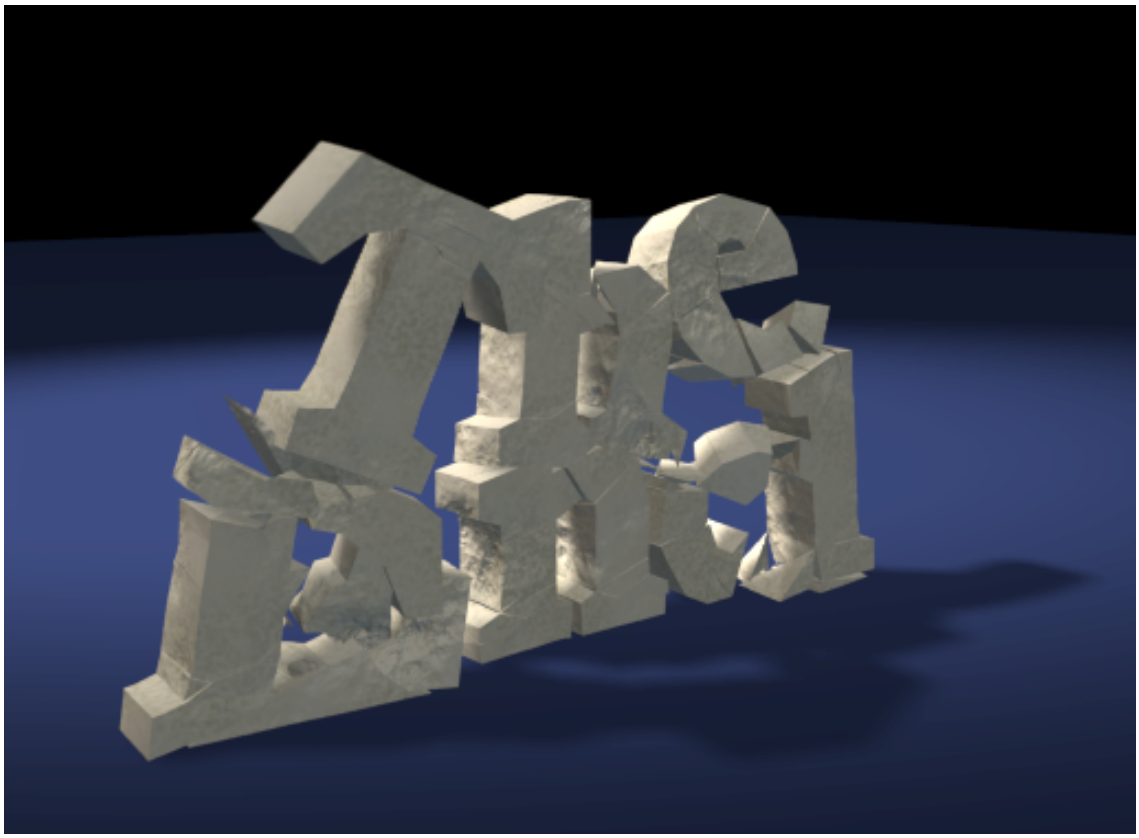
Studies of human subjects provide one possible way to make assessment by human viewers more objective. By combining the responses from a large set of observers, the effect of individual biases can be minimized. Unfortunately, designing studies that provide useful results can be difficult, particularly if one wishes to answer vague questions such as "Does this look realistic?" Furthermore, actually gathering the data for the experiments may require significant amounts of time and effort. Nonetheless, studies involving human viewers can be quite useful for answering questions about specific motions [23].

An alternative approach to user studies would be to try to compare measurable qualities from real-world events with results from a simulated version of the event. For example, one might run several trials where real bowls were broken by dropping them from a fixed height. The number, size, and distribution for the resulting fragments could then be recorded and compared to results from simulations with similar initial conditions. The success of this approach depends on whether the quantities being measured are consistent from one trial to the next. In the case of objects breaking, small changes in how the object is dropped or variation in internal

flaws can have very large effects on the resulting fracture pattern, making the validity of this assumption questionable. This approach also assumes that the qualities being measured are somehow relevant to the perceived realism of the motion.

In the fields of image processing and psychophysics, a significant amount of effort has been devoted to developing an understanding of how people perceive still, pixelated images. One of the results of this effort is models that can be used to measure the perceived difference between a pair of images [38]. In addition to providing a useful tool for evaluating the effect of image manipulation algorithms, such as lossy image compression, the models can be integrated directly into algorithms that perform other tasks. For example, Bolin and Meyer have developed an adaptive rendering method that uses a perceptual metric to select image samples [13].

Our current understanding of how people perceive three-dimensional motion is still too limited to allow the construction of a model that predicts how human viewers will perceive a particular motion. The problem is fundamentally more complex than it is for still images both because of the inherent time aspect, and because the model would probably have to account for effects due to the conversion from a three-dimensional geometric representation to a pixelated image. Although developing such a model will undoubtedly be a difficult task, the potential benefits to the field of motion generation would be significant.



Bibliography

[1] F. F. Abraham. Portrait of a crack: Rapid fracture mechanics using parallel molecular dynamics. *IEEE: Computational Science and Engineering*, pages 66–77, Apr. 1997.

[2] T. L. Anderson. *Fracture Mechanics: Fundamentals and Applications*. CRC Press, Boca Raton, second edition, 1995.

[3] S. Aoki, K. Kishimoto, H. Kondo, and M. Sakata. Elastodynamic analysis of crack by finite element method using singular element. *International Journal of Fracture*, 14(1):59–67, Feb. 1978.

[4] S. Atluri and T. Nishioka. Numerical studies in dynamic fracture mechanics. *International Journal of Fracture*, 27:245–261, 1985.

[5] E. A. Avallone and T. B. III, editors. *Marks' Standard Handbook for Mechanical Engineers*. McGraw–Hill, New York, NY, 9th edition, 1978.

[6] S. Bandi and D. Thalmann. An adaptive spatial subdivision of the object space for fast collision detection of animated rigid bodies. *Computer Graphics Forum*, 14(3):259–270, August 1995.

[7] D. Baraff. Coping with friction for non-penetrating rigid body simulation. In *SIGGRAPH 91 Conference Proceedings*, Annual Conference Series, pages 31–40. ACM SIGGRAPH, Addison Wesley, July 1991.

[8] D. Baraff. Fast contact force computation for nonpenetrating rigid bodies. In *SIGGRAPH 94 Conference Proceedings*, Annual Conference Series, pages 23–34. ACM SIGGRAPH, Addison Wesley, July 1994.

[9] D. Baraff and A. Witkin. Large steps in cloth simulation. In *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 43–54. ACM SIGGRAPH, Addison Wesley, July 1998.

[10] Z. P. Bažant, J. L. Glazik, and J. D. Achenbach. Elastodynamic fields near running cracks by finite elements. *Computers and Structures*, 8:193–198, 1978.

[11] T. Belytschko, Y. Y. Lu, and L. Gu. Element-free Galerkin method. *International Journal of Numerical Methods in Engineering*, 37:229–256, 1994.

[12] T. Belytschko, D. Organ, and Y. Krongauz. A coupled finite element–element-free Galerkin method. *Computational Mechanics*, 17:186–195, 1995.

[13] M. R. Bolin and G. W. Meyer. A perceptually based adaptive sampling algorithm. In *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 299–310. ACM SIGGRAPH, Addison Wesley, July 1998.

[14] S. Chenney and D. Forsyth. Sampling plausible solutions to multi-body constraint problems. In *SIGGRAPH 2000 Conference Proceedings*, Annual Conference Series. ACM SIGGRAPH, Addison Wesley, July 2000. To appear.

[15] J. D. Cohen, M. C. Lin, D. Manocha, and M. K. Ponamgi. I-collide: An interactive and exact collision detection system for large-scale environments. *1995 Symposium on Interactive 3D Graphics*, pages 189–196, April 1995.

[16] R. D. Cook, D. S. Malkus, and M. E. Plesha. *Concepts and Applications of Finite Element Analysis*. John Wiley & Sons, New York, third edition, 1989.

[17] N. Foster and D. Metaxas. Realistic animation of liquids. In *Proceedings of Graphics Interface '96*, pages 204–212. Canadian Human–Computer Communications Society, May 1996.

[18] N. Foster and D. Metaxas. Modeling the motion of a hot, turbulent gas. In *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 181–188. ACM SIGGRAPH, Addison Wesley, July 1997.

[19] Y. C. Fung. *Foundations of Solid Mechanics*. Prentice-Hall, Englewood Cliffs, N.J., 1965.

[20] Y. C. Fung. *A First Course in Continuum Mechanics*. Prentice-Hall, Englewood Cliffs, N.J., 1969.

[21] T. He. Fast collision detection using QuOSPO trees. *1999 ACM Symposium on Interactive 3D Graphics*, pages 55–62, April 1999.

[22] K. Hirota, Y. Tanoue, and T. Kaneko. Generation of crack patterns with a physical model. *The Visual Computer*, 14:126–137, 1998.

[23] J. K. Hodgins, J. F. O'Brien, and J. Tumblin. Perception of human motion with different geometric models. *IEEE Transactions on Visualization and Computer Graphics*, 4(4):307–316, October - December 1998.

[24] T. R. Hsu and Z. H. Zhai. A finite element algorithm for creep crack growth. *Engineering Fracture Mechanics*, 20(3):521–533, 1984.

[25] P. M. Hubbard. Approximating polyhedra with spheres for time-critical collision detection. *ACM Transactions on Graphics*, 15(3):179–210, July 1996.

[26] K. ichi Anjyo, Y. Usami, and T. Kurihara. A simple method for extracting the natural beauty of hair. In *SIGGRAPH 92 Conference Proceedings*, Annual Conference Series, pages 111–120. ACM SIGGRAPH, Addison Wesley, July 1992.

[27] A. Iones, S. Zhukov, and A. Krupkin. On optimality of OBBs for visibility tests for frustum culling, ray shooting and collision detection. In *Computer Graphics International 1998*. IEEE Computer Society, June 1998.

[28] M. Kass and G. Miller. Rapid, stable fluid dynamics for computer graphics. In *SIGGRAPH 90 Conference Proceedings*, Annual Conference Series, pages 49–57. ACM SIGGRAPH, Addison Wesley, July 1990.

[29] T. L. Kay and J. T. Kajiya. Ray tracing complex scenes. In *SIGGRAPH 86 Conference Proceedings*, Annual Conference Series, pages 269–278. ACM SIGGRAPH, Addison Wesley, Aug. 1986.

[30] J. T. Klosowski, M. Held, J. S. B. Mitchell, H. Sowizral, and K. Zikan. Efficient collision detection using bounding volume hierarchies of k–DOPs. *IEEE: Transactions on Visualization and Computer Graphics*, 4(1):21–36, January-March 1998.

[31] J. F. Knott. Numerical methods and linear elastic fracture mechanics as related to engineering structures and metallic alloys. In D. Owne and A. Luxmoore, editors, *Numerical Methods in Fracture Mechanics*, pages 1–24. Pineridge Press, 1980.

[32] K.-Y. Lee, J. D. Lee, and H. Liebowitz. Finite element analysis of the slow crack growth process in mixed mode fracture. *Engineering Fracture Mechanics*, 56(4):551–577, 1997.

[33] X. Li and J. M. Moshell. Modeling soil: Realtime dynamic models for soil slippage and manipulation. In *SIGGRAPH 93 Conference Proceedings*, Annual Conference Series, pages 361–368. ACM SIGGRAPH, Addison Wesley, Aug. 1993.

[34] S. Lien and J. T. Kajiya. A symbolic method for calculating the integral properties of arbitrary nonconvex polyhedra. *IEEE: Computer Graphics and Applications*, 4(10):35–41, 1984.

[35] M. Lin and S. Gottschalk. Collision detection between geometric models: A survey. In *The Proceedings of IMA Conference on Mathematics of Surfaces 1998*, 1998.

[36] M. C. Lin and J. F. Canny. Efficient collision detection for animation. *Third Eurographics Workshop on Animation and Simulation*, September 1992. Held in Cambridge, United Kingdom.

[37] Y. Y. Lu, T. Belytschko, and M. Tabbara. Element-free Galerkin method for wave propogation and dynamic fracture. *Computer Methods in Applied Mechanics and Engineering*, 126:131–153, 1995.

[38] J. Lubin. A visual discrimination model for imaging system design and evaluation. In E. Peli, editor, *Vision Models for Target Tracking and Recognition*, pages 245–283, New Jersey, 1995. World Scientific.

[39] J. Marks, B. Andalman, P. A. Beardsley, W. Freeman, S. Gibson, J. Hodgins, T. Kang, B. Mirtich, H. Pfister, W. Ruml, K. Ryall, J. Seims, and S. Shieber. Design galleries: A general approach to setting parameters for computer graphics and animation. In *SIGGRAPH 99 Conference Proceedings*, Annual Conference Series, pages 389–400. ACM SIGGRAPH, Addison Wesley, July 1999.

[40] O. Mazarak, C. Martins, and J. Amanatides. Animating exploding objects. In *Proceedings of Graphics Interface '99*, pages 211–218. Canadian Human–Computer Communications Society, June 1999.

[41] B. Mirtich. V-clip: Fast and robust polyhedral collision detection. *ACM Transactions on Graphics*, 17(3):177–208, July 1998.

[42] A. Needleman. Numerical modeling of crack growth under dynamic loading conditions. *Computational Mechanics*, 19:463–469, 1997.

[43] M. Neff and E. Fiume. A visual model for blast waves and fracture. In *Proceedings of Graphics Interface '99*. Canadian Human–Computer Communications Society, June 1999.

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

[45] T. Nishioka and S. Atluri. Numerical analysis of dynamic crack propogation: Generation and prediction studies. *Engineering Fracture Mechanics*, 16(3):303–332, 1982.

[46] A. Norton, G. Turk, B. Bacon, J. Gerth, and P. Sweeney. Animation of fracture by physical modeling. *The Visual Computer*, 7(4):210–217, July 1991.

[47] J. F. O'Brien, V. B. Zordan, and J. K. Hodgins. Combining active and passive simulations for secondary motion. *IEEE: Computer Graphics and Applications*, July 2000. In press.

[48] W. D. Pilkey. *Formulas for Stress, Strain, and Structural Matricies*. John Wiley and Sons, Inc., New York, NY, 1994.

[49] M. K. Ponamgi, D. Manocha, and M. C. Lin. Incremental algorithms for collision detection between polygonal models. *IEEE: Transactions on Visualization and Computer Graphics*, 3(1):51–64, January - March 1997.

[50] J. Popovic, S. M. Seitz, M. Erdmann, Z. Popovic, and A. Witkin. Interactive manipulation of rigid body simulations. In *SIGGRAPH 2000 Conference Proceedings*, Annual Conference Series. ACM SIGGRAPH, Addison Wesley, July 2000. To appear.

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

[52] C. C. Rankin, F. A. Brogan, and E. Riks. Some computational tools for the analysis of through cracks in stiffened fuselage shells. *Computational Mechanics*, 13:143–156, 1993.

[53] M. M. Rashid. A computational procedure for simulation of crack advance in arbitrary two-dimensional domains. *Computational Mechanics*, 20:133–138, 1997.

[54] B. Robertson. Antz-piration. *Computer Graphics World*, 21(10), 1998.

[55] B. Robertson. Building a better mouse. *Computer Graphics World*, 22(12), 1999.

[56] J. Schöberl. NETGEN – An advancing front 2D/3D–mesh generator based on abstract rules. *Computing and Visualization in Science*, 1:41–52, 1997. http://titan.sfb013.uni-linz.ac.at/ joachim/usenetgen/.

[57] J. Smith, A. Witkin, and D. Baraff. Fast and controllable simulation of the shattering of brittle objects. In *Proceedings of Graphics Interface 2000*. Canadian Human–Computer Communications Society, May 2000.

[58] N. Sukumar, B. Moran, T. Black, and T. Belytschko. An element-free Galerkin method for three-dimensional fracture mechanics. *Computational Mechanics*, 20:170–175, 1997.

[59] R. W. Sumner, J. F. O'Brien, and J. K. Hodgins. Animating sand, mud, and snow. *Computer Graphics Forum*, 18(1), Mar. 1999.

[60] S. Suri, P. H. amd, and J. Hughes. Analyzing bounding boxes for object intersection. *ACM Transactions on Graphics*, 18(3):257–277, July 1999.

[61] D. V. Swenson and A. R. Ingraffea. Modeling mixed-mode dynamic crack propogation using finite elements: Theory and applications. *Computational Mechanics*, 3:381–397, 1988.

[62] D. Terzopoulos. Regularization of inverse visual problems involving discontinuities. *IEEE: Transactions on Pattern Analysis and Machine Intelligence*, 8(4):413–424, July 1986.

[63] D. Terzopoulos and K. Fleischer. Deformable models. *The Visual Computer*, 4(6):306–331, Dec. 1988.

[64] D. Terzopoulos and K. Fleischer. Modeling inelastic deformation: Viscoelasticity, plasticity, fracture. In *SIGGRAPH 88 Conference Proceedings*, Annual Conference Series, pages 269–278. ACM SIGGRAPH, Addison Wesley, Aug. 1988.

[65] D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer. Elastically deformable models. In *SIGGRAPH 87 Conference Proceedings*, Annual Conference Series, pages 205–214. ACM SIGGRAPH, Addison Wesley, July 1987.

[66] J. C. Thesken and P. Gudmundson. Application of a moving variable order singular element to dynamic fracture mechanics. *International Journal of Fracture*, 52:47–65, 1991.

[67] G. van den Bergen. Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphics Tools*, 2(4):1–14, 1997.

[68] P. Volino, M. Courshesnes, and N. M. Thalmann. Versatile and efficient techniques for simulating cloth and other deformable objects. In *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 137–144. ACM SIGGRAPH, Addison Wesley, Aug. 1995.

[69] P. Volino and N. Magnenat-Thalmann. Collision and self-collision detection: Efficient and robust solutions for highly deformable surfaces. *Computer Animation and Simulation '95*, pages 55–65, September 1995.

[70] P. Witting. Computational fluid dynamics in a traditional animation environment. In *SIGGRAPH 99 Conference Proceedings*, Annual Conference Series, pages 129–136. ACM SIGGRAPH, Addison Wesley, Aug. 1999.

[71] X.-P. Xu and A. Needleman. Numerical simulations of fast crack growth in brittle solids. *Journal of the Mechanics and Physics of Solids*, 42(9):1397–1434, Sept. 1994.

[72] G. D. Yngve, J. F. O'Brien, and J. K. Hodgins. Animating explosions. In *SIGGRAPH 2000 Conference Proceedings*, Annual Conference Series. ACM SIGGRAPH, Addison Wesley, July 2000. To appear.

Graphical Modeling and Animation of Fracture

James F. O'Brien

133 Pages

Directed by Dr. Jessica K. Hodgins

This thesis addresses the problem of graphically modeling and animating the realistic behavior of materials that can undergo fracture due to deformation-induced stress. Using an approach based on linear elastic fracture mechanics and non-linear finite element analysis, three-dimensional volumes are modeled using a mesh of tetrahedral elements. By analyzing the stresses created as the mesh deforms, the simulation determines where cracks should begin and in what directions they should propagate. The system accommodates arbitrary propagation directions by dynamically retesselating the mesh. Because cracks are not limited to element boundaries, the models can form irregularly shaped features as they shatter. This technique overcomes limitations of previous methods that made it difficult to represent the shape of the fracture's surface. Results are presented to demonstrate that this method can be used to animate complex, real-world situations in a compelling, realistic fashion.