UNIVERSITY OF CALGARY

# Modeling Fracture Formation on Growing Surfaces

by

Pavol Federl

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE

DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

CALGARY, ALBERTA

SEPTEMBER, 2002

THE UNIVERSITY OF CALGARY

FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled “Modeling Fracture Formation on Growing Surfaces” submitted by Pavol Federl in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

---

Supervisor, Dr. Przemyslaw Prusinkiewicz, Computer Science

---

Dr. Brian Wyvill, Computer Science

---

Dr. Jon Rokne, Computer Science

---

Dr. Marcelo Epstein, Mechanical and Manufacturing Engineering

---

External Examiner, Dr. Eugene Fiume, University of Toronto

---

Date

# *Abstract*

*This thesis describes a framework for modeling fracture formation on differentially growing, bi-layered surfaces, with applications to drying mud and tree bark. Two different, physically based approaches for modeling fractures are discussed. First, an approach based on networks of masses and springs is described. Two important shortcomings of the mass-spring approach are then identified: a tendency of the fractures to align themselves to the underlying mesh, and an unclear relationship between the simulation parameters and the real material properties. The rest of the thesis is focused on the second approach to modeling fractures, based on solid mechanics and finite element method, which effectively addresses the shortcomings of the mass-spring based approach. Two types of growth are investigated: isogonic and uniform anisotropic. The growth is then incorporated within the framework of finite element methods, implemented using velocity vector fields. The growth tensor is used to verify the properties of the growth. A number of efficiency and quality enhancing techniques are also described for the finite element based approach. An adaptive mesh refinement around fracture tips is introduced, which reduces the total number of elements required. Construction of a temporary local multi-resolution mesh around the fracture tips is described, which is used to calculate stresses at fracture tips with increased accuracy but without introducing any additional elements permanently into the model. Further, a general method for efficient recalculation of an equilibrium state is presented, which can be employed after localized changes are made to the model. Finally, an adaptive time step control is described, which automatically and efficiently determines the next optimal time step during simulation.*

# *Acknowledgements*

*My first and most important thank-you goes to my mother, who supported my education in every possible way. It is impossible to imagine how difficult it would have been without her support. My next thank-you belongs to my supervisor, Przemyslaw Prusinkiewicz, for his guidance, for the numerous hours we spent together discussing my research, for his brilliant ideas and invaluable advice, and for offering his tremendous experience when helping to edit this thesis. A very special thank-you goes to my girlfriend, Jennifer Walker. I would like to thank her for the incredible job of proofreading this thesis, for patiently putting her life on hold while I busy with my research and also for her emotional support. Finally, I would like to thank Marcelo Epstein for his valuable advice in the area of solid mechanics, and to everybody in the graphics jungle group for creating an enjoyable working atmosphere.*

# *Table of Contents*

# *List of Figures*

# *List of Tables*

# CHAPTER 1 *Introduction*

In recent years, considerable attention has been devoted to computer-based modeling and visualization of various natural phenomena [3]. Instead of real-life experiments, mathematical models are used as virtual experiments. Computers can easily simulate conditions which would otherwise be very difficult or impossible to achieve, such as weightlessness, absolute zero temperatures or perfect vacuum. Computer simulation can therefore be an invaluable tool for scientists wishing to perform "what-if" type experiments. Additionally, by modeling only the essential components of experiments [53], computers are often able to generate results within a fraction of the time required in real life.

Due to their interdisciplinary character, computer based simulations of natural phenomena offer synergy between computer science and other scientific fields, such as biology. Computer graphics enhances this synergy even further, by making visualization of the simulation results possible. *Visual inspection* is an important measure of success of a theory. Upon completion of a virtual experiment, results can be easily evaluated by the human eye, and the agreement between these virtual results and reality can function as evidence supporting or disproving the theory on which the simulation was based[1]. From the computer graphics perspective, this synergy provides tangible benefits as well. For

1. This does not contradict the importance of qualitative analysis; but qualitative analysis of patterns remains a largely open problem.

example, the simulation of pattern formation models can be used to generate natural looking textures, which are important in image synthesis. Their presence is vital for achieving photorealism, because images lacking textures appear unrealistic and visually uninteresting.

Many interesting and highly complex surface patterns are produced as a result of cracking. Cracks often occur as a direct consequence of the dynamic change in a material's structure, as can be observed in aging paint, drying mud, old ceramics, or tree bark. A successful model of fracture formation on growing domains can help us to study and to better understand the processes involved in crack pattern formation in nature. It can also lead to the synthesis of realistic textures for CG purposes. In my research, I attempt to simulate such phenomena using physically based methods.

There are three basic types of textures used in computer graphics: image-based, procedural and volumetric. Image-based textures are normally obtained by digitizing an existing image, such as the photograph of a real-world pattern, and storing it as an array of pixels. The texture is then mapped onto the surfaces being rendered. In contrast, procedural textures are programs that determine the value of any pixel in the texture when queried. The algorithms used in procedural textures are often designed after careful observation of the desired real-life pattern [10]. Volumetric textures, initially introduced by Kajiya and Kay [29] and later refined by Neyret [43], can be used to render complex geometries. The basic idea is to first compute a pattern of 3D geometry in some reference volume, and then tile the pattern over a desired surface in a manner similar to applying a regular 2D texture. In the recent past, a new approach to texture synthesis has been studied in which the generated texture is synthesized from examples of real-life textures. To some extent, this method of texture synthesis from examples offers a bridge between the traditional image-based and procedural textures. From the computer graphics perspective, the area of my research can be classified as a procedural texture generation. Although procedural textures were first introduced as many as 15 years ago (by Peachey [49] and

Perlin [50]), the design of algorithms for procedural texture generation remains an open research problem. I believe the simulation of fracture formation I consider in my work presents an intriguing research path for addressing this problem for a class of textures representing models in which fracture patterns develop due to growth.

The main objective of my research is to introduce a framework for scientific simulation of fracture pattern formation occurring on surfaces experiencing positive or negative growth. The basic methodology of my approach is to integrate surface growth with the existing physically based methods for simulating material deformation. I consider two such physically based techniques: the first one is based on networks of masses and springs, and the second is based on solid mechanics combined with the finite element method (FEM).

The main contributions of my research described in this thesis are:

- Integration of growth with an existing mass-spring based model of fracture formation.
- Incorporation of growth into the framework of finite element methods. I show how to implement negative growth (shrinkage) in the material, and positive growth of the underlying background.
- Illustration of these methods using selected examples: bark and drying mud.

I also make a number of technical contributions with respect to modeling fractures using finite element methods:

- I demonstrate how modeling of fractures can be combined with dynamic subdivision to avoid unnecessary increase of the global discretization level.
- I describe an adaptive technique for local recalculation of the equilibrium solution, which diminishes the need to recalculate the solution for the

entire domain, and therefore improves the efficiency of the simulation.

- I show how adaptive time step control can be used to automatically and efficiently forward the simulation time to a point of interest, i.e. to a point where the next fracture will occur.
- I show how nodes can be repositioned when modeling fractures, which reduces the number of degenerate elements.
- I describe a method which dynamically improves the shapes of the elements around crack tips using a localized angle-based mesh smoothing technique. I also show how the persisting degenerate elements can be removed by edge-collapsing.
- Finally, I demonstrate how a temporary local multi-resolution mesh can be used to calculate the stresses in the material at fracture tips, without having to permanently increase the discretization level.

These contributions and the results are explained in full detail in the following chapters, organized as follows. In Chapter 2 I discuss the previous work relevant to my research. The mathematical models describing growth, together with my implementation of negative and positive growth are discussed in Chapter 3. Chapter 4 describes how I extended an existing mass-spring based model of fracture formation by incorporating growth, and demonstrates why mass-spring models are limited in their capacity to successfully model fracture formation on growing domains. Chapter 4 is later extended by Appendix A, to which I relegate the discussion of synthesizing self-similar patterns using a one dimensional mass-spring model. Chapter 5 describes my second approach to fracture modeling, based on solid mechanics and finite element models. There, I also document the results I have obtained using this approach. Finally, Chapter 6 provides a brief summary of the contributions of my work, and concludes the thesis with potential directions for future work.

# CHAPTER 2 *Review of previous work*

In the recent past, simulation of pattern formation has been recognized by computer graphics researchers as a viable method for generating realistic patterns and animations. The existing models of pattern formation can be divided into two classes: models assuming static space and models assuming changing space. An example of models operating on a static surface are the reaction-diffusion models, originated by Turing [67] in 1952. They generally deal with the distribution dynamics of pattern forming substances in a medium of constant size. L-system[1] models, in contrast, fall into a category of models operating on changing domains. They deal with development, but are limited to branching structures (see Prusinkiewicz and Lindenmayer [54]). In spite of the relative success of some models, e.g. reaction-diffusion models that capture shell pigmentation patterns [37], or L-system models of herbaceous plants [54], many patterns continue to elude modeling efforts. In my research, I explore the hypothesis that some of these patterns can be captured with fracture models operating on growing surfaces, such as bark on tree trunks or drying mud. In this chapter I review previous work most related to my research.

Previous work in the area of simulating fracture formation can be essentially divided into two groups based on the amount of discontinuity produced by fractures in the material. In the first group, the models use fractures to introduce large discontinuities, i.e. they focus

1. Although they are usually not perceived as models of pattern formation, they do contribute a class of morphogenetic models.

on simulating the process of an object breaking into pieces as a result of applied external forces, with no emphasis on the actual fracture pattern being formed. The second group of models use fractures to produce small discontinuities in the object's surface, and they focus on the emerging crack patterns. The models in the second group operate either on static or dynamic surfaces.

**Models of fracture formation assuming large discontinuities**

One of the earliest works in the group of models simulating large discontinuities is that of Norton et al. [44], who investigated the use of a 3D mass-spring model to produce realistic animations of a teapot falling onto a surface and shattering into pieces. The teapot was discretized into cubes, connected to each other at their nodes. The internal physical properties of each cube were simulated by a system of masses and springs. Each cube consisted of 8 point masses and 28 springs connecting each pair of nodes. The cubes would break when their internal springs reached the limit of elastic behavior, leading to the formation of fractures. Since entire cubes of material were being removed from the model, the fracture planes in the generated images were quite rugged.

An animation method for a breaking window under a spherical blast wave was developed by Neff and Fiume [42]. Their fracture model was based on the theory of rapid fracture propagation. Initially, a small micro-fracture was introduced, which subsequently propagated through the material, releasing the stored energy. By randomizing various simulation parameters, the fractures wiggled and split, producing convincing results. The impact of a blast wave on the surrounding objects was simulated by Mazarak et al. [35]. The simulated model was discretized using a connected voxel grid. The fractures in the material were modeled by breaking the connecting link between two voxels whenever the pressure generated by the blast wave at their midpoint was greater than the link's yield threshold.

Terzopoulos and Fleischer [63][64] investigated modeling of viscoelastic and plastic deformations. They presented a general technique for modeling these phenomena, at the heart of which were three energy functions used to estimate the deformation over surfaces defined by splines. The energy functions were defined using metric tensors, and the simulated models were discretized using various methods. Using a finite differencing method, one of their models simulated fracture formation and propagation in torn paper and ripped cloth. The discontinuities introduced by fractures were modeled by setting the stiffness coefficients of the material to zero between any two nodes exceeding a maximum prescribed threshold distance. The most significant disadvantage of this approach was the use of finite differencing, which requires discretization of a surface by a regular rectangular grid. The rectangular grid is unsuitable for discretizing surfaces of arbitrary topologies. The resulting fractures also exhibited an aliasing effect caused by the imposed grid. Metaxas and Terzopoulos later employed finite element methods for animating deformations [38], as FEMs are generally more robust and accurate.

O'Brien and Hodgins [45] also developed a method for fracture modeling based on continuum mechanics, with the deformation of the material expressed with strain tensors. The continuous models were discretized into tetrahedral meshes and then modeled using a finite element approach, as opposed to a mass-spring approach. Fractures were modeled by splitting the elements at nodes with high stress, in the direction of the computed fracture planes. Applications included realistic simulations of a wall being shattered by a wrecking ball, a bowl breaking upon impact with a floor, and a slab of glass shattered by a heavy weight. A more detailed description of this algorithm was later given by O'Brien in his doctoral thesis [46].

A physically motivated model of rapid simulation of fracture formation in brittle materials under impact was described by Smith, Witkin and Baraff [62]. Their model is similar to a mass-spring model, but instead of using stiff springs, the authors framed the problem in terms of distance-preserving constraints. The forces exerted by the distance-preserving

constraints were calculated using Lagrange multipliers, yielding a large sparse system of linear equations that had to be solved for each simulation time step (in this respect, their method is very similar to the finite element approach), to which end they used a conjugate gradient method. The modeled materials were discretized using tetrahedral elements, modeled as point-masses interconnected by the linear constraints. The fractures were modeled by breaking the constraints when they reached certain thresholds. The main disadvantage of their approach lies in the resulting fractures propagating exclusively along the mesh boundaries, resulting in jaggy cracks.

The mechanical models reviewed up to this point were directly applied to simulating fracture formation. Various other mechanically based models exist, which simulate deformation in materials, but lack direct application to fracture modeling. Some of these models could be adapted to fracture formation, such as the stable and efficient algorithm for animating mass-spring systems introduced by Desbrun et al. [8]. In their approach, the authors employ an integration scheme derived from implicit integration. Their algorithm is an approximated implicit predictor/corrector scheme, preserving important physical quantities such as linear and angular moments. By diminishing the need to solve a linear system of equations, this approach is capable of achieving interactive speeds of realistic animations of any mass-spring network, although the authors noted that the ‘approximation cannot be used for more accurate simulations’.

**Models of fracture formation focusing on the emerging crack patterns**

A model of crack formation and propagation due to shrinkage of a material deposited on a background surface was introduced by Skjeltorp and Meakin [61]. The deposited material was physically modeled as a mono-layer of uniformly sized micro-spheres confined between two parallel sheets of glass. The diameter of the spheres gradually decreased as they dried out, leading to the formation of cracks. The material layer was discretized by a hexagonal lattice of masses and springs. Experiments and simulations produced cracking

patterns resembling those observed in thin films of paint or ceramic-coated materials. The regular arrangement of micro-spheres, however, led to six privileged cracking directions, rotated by 60 degrees with respect to each other. The presence of this directionality was particularly visible in computer simulations (cf. Figure 2b in [61]).

The Skjeltorp and Meakin model was adapted to generate fracture patterns on tree bark by Federl and Prusinkiewicz [13], by introducing anisotropic growth of the background. This work will be reviewed in greater detail later in the thesis. Hirota et al. [24] also extended Skjeltorp and Meakin's model for image synthesis purposes. In their work, they related the formation and propagation of cracks to the notions of the elasticity theory - strain and stress. The mass-spring network was still used as a computational model, although the authors noted it was "less suitable for the precise description of the physical characteristic than the finite element method". Non-isotropic behavior of the material was simulated by adjusting the yielding criteria of springs to their initial orientations. In [25], Hirota et al. extended their previous model to volumes. Sample applications included a cubic block of clay that dried on the surface.

Gobron and Chiba [17] used solid mechanics techniques to analyze the relationship between stresses and fractures in a material. The result was a semi-physically based model, implemented as a cellular automaton operating on a 2D rectangular grid. Sample applications included simulated fracture patterns in concrete, mud, ceramics and glaze. Inspired by Gobron's and Chiba's work, Paquette [48] extended their model by adding adhesion and curling behavior, and applied the resulting model to generate images of aging paint.

Federl and Prusinkiewicz [14] introduced growth into the framework of finite element methods, and applied the resulting model to simulate formation of crack patterns in drying mud and tree bark. This work represents a large portion of my research, and therefore it has been incorporated into this thesis.

A phenomenological approach to synthesizing bark patterns has been recently proposed by Lefebvre and Neyret [31]. The bark was modeled with a series of independent circular strips perpendicular to the main axis of the tree. Each strip was modeled as a one-dimensional system of two alternating types of springs, one representing the material, the other representing the fracture. The fracture was initiated inside a strip when one of the material springs was elongated beyond its threshold. The crack was modeled by splitting this material spring and inserting a fracture spring. An existing crack was allowed to propagate to the adjacent strips. The direction of propagation was determined by considering both the state of the strip of the existing crack, as well as the state of the strip into which the fracture propagated. The resulting bark patterns were quite realistic, and included patterns synthesized for branching structures. The strength of this method lies in its speed. By modeling only a simplified model of tree bark it is possible to run the simulations interactively. Unfortunately, by considering only a simplified model of bark, the capacity of the model to be used as a scientific tool is debatable. For example, the cracks are only allowed to propagate from one strip to an adjacent strip, limiting the application of this model to types of bark in which cracks occur vertically.

# CHAPTER 3 *Modeling growth*

In my research, I consider fracture formation in materials that can be modeled by simplifying them into two layers: a *material layer* and a *background layer.* Stress, and the subsequent cracks, result from the growth or shrinkage of one layer with respect to the other. I focus on two phenomena: formation of bark patterns on tree trunks, and formation of cracks in drying mud. In the next section I describe some of the existing mathematical techniques for expressing growth. I then show how I model bark and mud with two layers in Section 3.2, and how I simulate growth using these layers in Section 3.3.

## 3.1. Mathematical description of growth

The phenomenon of growth is common to all living organisms and was therefore studied and analyzed by many biologists, resulting in the formation of its various mathematical descriptions. One of the earliest mathematical models representing growth of an organism is the concept of *allometric growth*, termed by Huxley et al. [27]. It quantified the relationship of two measurable traits of an organism using an exponential equation,

*Eq. 3.1* $$y = bx^k ,$$

where $b$ and $k$ are constants, and $x$ and $y$ represent the measures of the two traits compared. The allometric constant $k$ is the ratio of the two specific growth rates of the two traits, and is therefore a pure number:

$$k = \frac{\frac{dy}{ydt}}{\frac{dx}{xdt}}.$$

The specific growth rate $\frac{dx}{xdt}$ measures the relative change of the trait $x$ with respect to time $t$. For example, the specific growth rate equal to 0.1 represents a 10% growth per 1 unit of time.

A coordinate transformation approach for analytical description of the shape changes of various organisms was proposed by D'Arcy Thompson [65,66]. Thompson used simple geometric transformations to show how one form of an organism can be *warped* into a different form. Richards and Riley [57] used Thompson's technique to study the development of salamander larvae (Figure 3.1). Richards and Kawanagh [55,56] combined the Huxley's et al. relative growth model with Thompson's transformed coordinate method into a single analytical technique, and illustrated the usefulness of their new technique by determining the regions of similar specific growth rates on a tobacco leaf. They also noted that the maximum and minimum linear growth directions are always perpendicular to each other.

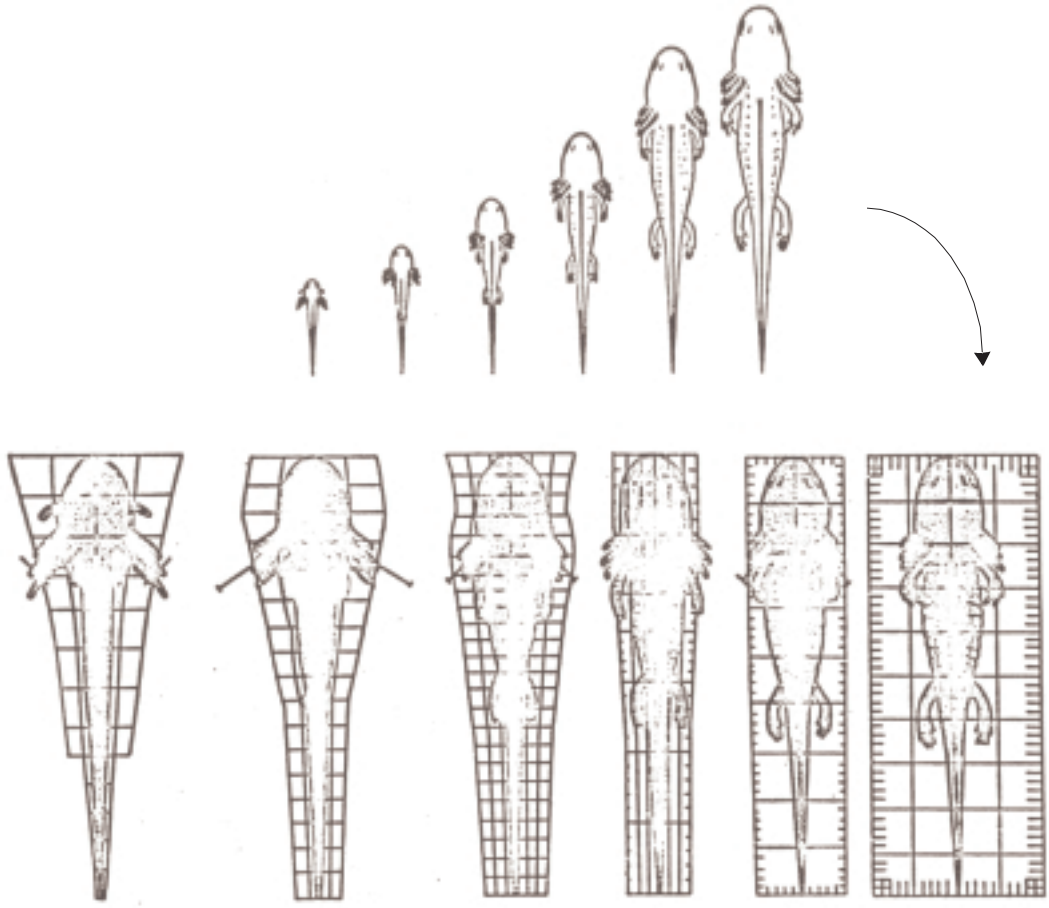


**FIGURE 3.1:** *Transformation of salamander larvae (reproduced from* [56]*).*

The relative elemental rate of growth [60,55,23], denoted as $RERG_{l(s)}$, of a segment of length $\Delta s$ in the direction of the segment, is:

*Eq. 3.2*

$$RERG_{l(s)} = \lim_{\substack{\Delta s \to 0 \\ \Delta t \to 0}} \frac{\Delta(\Delta s)}{\Delta s \Delta t} = \frac{d(ds/dt)}{ds} = \frac{dV_s}{ds},$$

where $V_s$ is the displacement velocity of the segment in the direction $s$. Erickson [11] showed how the field of displacement velocities can be used to estimate the relative

elemental growth rates. If the velocity field $V$ is known, $V_s$ can be calculated by a scalar product of $V$ and the unit vector in the direction of $s$, i.e. $V_s = V \cdot e_s$. Then the relative elemental growth rate of a segment can be written as:

$$RERG_{l(s)} = \frac{d(V \cdot e_s)}{ds} \quad \text{or} \quad RERG_{l(s)} = \nabla(V \cdot e_s) \cdot e_s .$$

The relative elemental rate of growth $RERG_{l(s)}$ is defined for every direction $s$. Since it may be non-zero in every direction, it is neither a scalar nor a vector. Silk and Erickson noticed this property of $RERG_{l(s)}$, and consequently used tensor fields to analyze the growth of a plant organ [12][60]. They treated the growth of a plant organ as a deformation in a compressible fluid, and thus expressed the growth using in terms standard to fluid dynamics, i.e. in terms of rate-of-strain tensor

$$d_{ij} = \frac{1}{2}\left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i}\right) ,$$

and vorticity tensor

$$\Omega_{ij} = \frac{1}{2}\left(\frac{\partial v_i}{\partial x_j} - \frac{\partial v_j}{\partial x_i}\right) ,$$

where $v_i$ represents the components of the velocity field $V = (v_x, v_y, v_z)$ and $x_i$ denotes the axis $i$ of the coordinate system used.

Hejnowicz and Romberger [23] analyzed the definition of the relative elemental rate of growth of a line segment, and arrived at the formulation of *growth tensor.* Their growth

tensor $T_{ij}$ is a second rank tensor, identical to a tensor which can be obtained by summing up the rate-of-strain and vorticity tensors computed by Silk and Erickson, i.e.:

*Eq. 3.3*

$$T_{ij} = d_{ij} + \Omega_{ij} = \frac{\partial v_i}{\partial x_j}.$$

Although the end-result of Hejnowicz's and Romberger's work is similar to that of Silk and Erickson, Hejnowicz and Romberger pointed out that the growth '*tensor concept is evoked by the very nature of growth, not merely by its analogy to deformation or strain*'. Another important aspect of Hejnowicz's and Romberger's work is their definition of principal directions of growth, in which the plant organ achieves the maximum/minimum rates of growth. They defined these principal directions of growth as the eigenvectors of the growth tensor.

In the next section I describe the assumed simplified models of tree bark and mud. Then I show how to use velocity vector fields to implement anisotropic planar growth of the background layer. I also show how the growth tensor can be used to determine such velocity vector fields. In the case of cylindrical and spherical surface growth, I use the growth tensor to mathematically demonstrate the isogonic and anisotropic properties of such growth.

## 3.2. The assumed models of bark and mud

Tree bark consists of dead conductive tissue, phloem, which is expanded by the radial growth of cambium inside the trunk [59]. As a result of this expansion, the bark stretches until it reaches its limit of deformation and breaks. In my simulation I use a simplified, two-layer model of a growing tree trunk, with the cross-section shown in Figure 3.2 on the

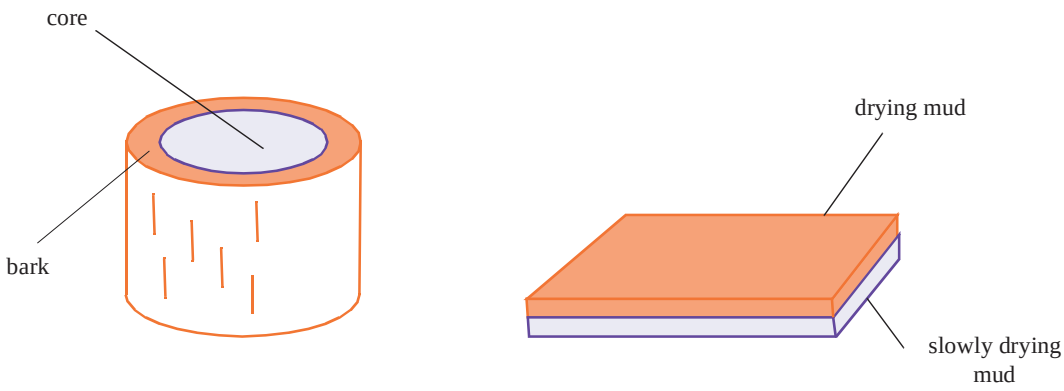



**FIGURE 3.2:** *The assumed two-layered models of bark and drying mud.*

left. The inside core grows radially and does not break. I model the cylindrical core as the *background layer*. The outer layer, which I model as the *material layer*, represents the bark.

As water evaporates from mud, the mud shrinks and thus changes in shape. Since water evaporates faster from those layers closest to the surface, layers change shape at different rates with respect to each other. This non-uniform shrinkage (or *unequal growth*) of the various layers leads to the formation of stress, and consequently to the formation of cracks. Although the presence of distinguishable layers is not as evident as it was in the case of tree bark, I still model drying mud with two layers - by employing an 'extreme' discretization: The background layer is assumed to be static, representing either mud that dries very slowly or the surface on which the drying mud rests. The material layer represents the drying mud and is attached to the background layer. The assumed model of drying mud is illustrated in Figure 3.2 on the right.

## 3.3. Implementing growth and shrinkage

In my research I consider surfaces experiencing differential growth, caused by two connected layers changing shape at different rates. In general, I consider three cases of differential growth: shrinkage of a material layer with respect to a static background; growth of an underlying background surface with respect to a static material layer; or the combination of these two phenomena. For example, in the case of drying mud, the top layer dries faster (shrinking) than the lower layers. In the case of tree bark, on the other hand, the outer dead-tissue layer does not grow, but the inner living wood layer continuously expands [59]. Furthermore, the dead bark can also shrink due to drying.

The effects of growth of the underlying background on the material layer, and how it can lead to increased stresses is illustrated in Figure 3.3. The top of Figure 3.3 shows a mass-spring system undergoing planar background growth. This method of discretizing a material attached to a background using a mass-spring system was introduced by Skjeltorp and Meakin [61]. The material layer is represented by a system of masses interconnected with material springs. The connection to the underlying background is modeled by anchor springs, which attach each mass to the background at an anchor point. In contrast to the material springs, the rest lengths of the anchor springs are equal to zero. In essence, the anchor springs model the mechanical property of the material under sheering deformation, and by varying the stiffness of the anchor springs, materials of different thicknesses can be modeled. As the background expands, the positions of the anchor points are adjusted to capture such growth. Since the masses are attached to the background surface via anchor springs, the growth of the background results in forces being exerted on the masses. To balance these forces, the material springs elongate as well, leading to increased tensions. At the bottom of Figure 3.3, a single finite element in the shape of a wedge is illustrated. The wedge is attached to the growing background by the bottom three nodes. As the

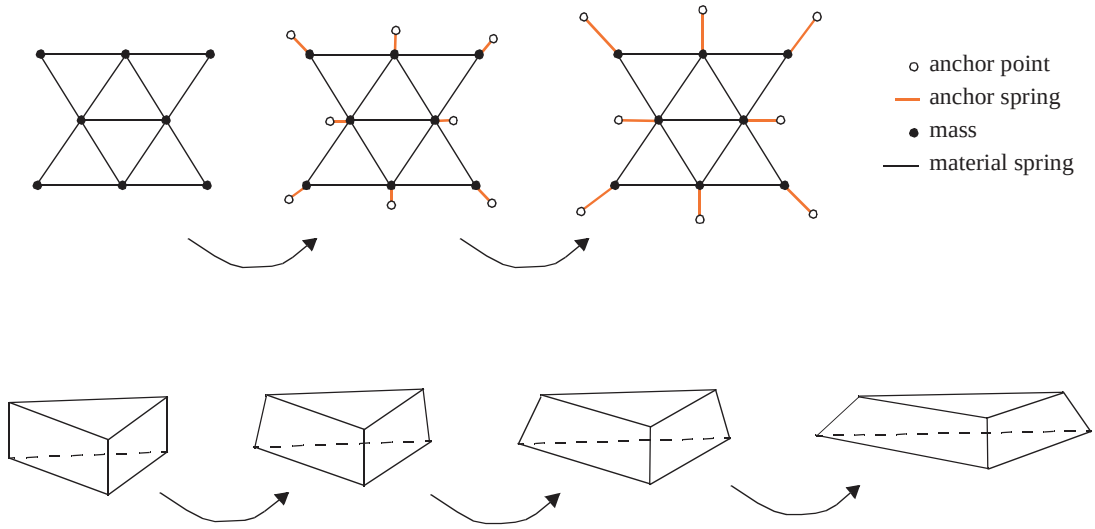



**FIGURE 3.3:** *The effects of a growing background on the material layer.*

background expands, the bottom nodes are forced apart, leading to increased internal stresses within the element.

Shrinkage of a material layer with respect to a static background surface leads to the formation of increased stresses in an analogous manner, as illustrated in Figure 3.4. The top of Figure 3.4 shows a material layer shrinking, which is implemented by reducing the rest lengths of the material springs. The material springs then begin to apply forces on the masses. Since the masses are attached to the background via anchor springs, the material springs are prevented from achieving their rest lengths, leading to increased tensions. The bottom of Figure 3.4 illustrates the shrinkage of the material layer, modeled by adjusting the reference shapes of the finite element wedges. Although the reference shape of the wedge is made smaller in each simulation step, the bottom of the wedge is still attached to the static background, and therefore cannot assume its reference state. As a result, the element undergoing shrinkage is deformed, leading to the presence of stresses.

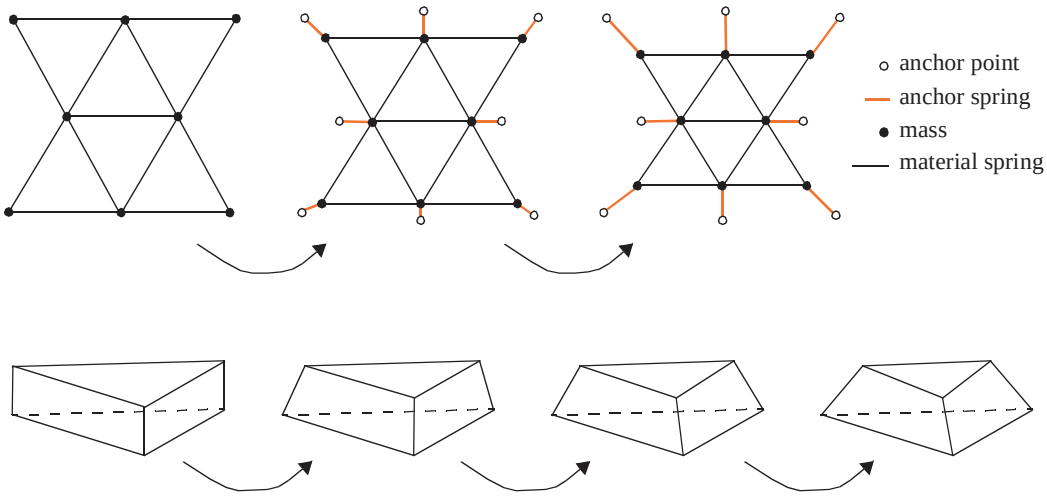



**FIGURE 3.4:** *Shrinkage of the material layer with respect to a static background.*

### 3.3.1. Growth of the background layer

The material layer in which the cracks are formed is attached to the background layer. In the case of the mass-spring implementation of my fracture simulation, the attachment of the material layer to the background layer is modeled using anchor springs. In the finite element approach, the connection between the material and the background layers is modeled using fixed nodes. To simplify the discussion of my growth implementation, I will commonly refer to both anchor points and fixed nodes as *attachment points*.

I simulate the growth of the underlying background layer by adjusting the positions of the attachment points. At each simulation time step, the trajectory of each attachment point on the background is defined by its initial position $P(0) = (x, y, z)$ and a velocity vector $\vec{v}$. The velocity vector describes the direction and the amount of displacement per unit of

time of a point on the growing background. The position $P_k(t)$ of the attachment point $k$ at time $t$ is calculated using the general formula:

*Eq. 3.4*

$$P_k(t) = P_k(0) + t\vec{v}_k \ .$$

The initial positions of the attachment points are determined at the time of discretization. The velocity vectors are set to reflect different types of growth. I investigated two types of surface growth: *isogonic* and *uniform anisotropic* [59]. Under isogonic growth, the rate of growth is the same in all directions and at all locations in the surface. For example, a circle under isogonic growth will change in area, but it will remain a circle. Also, under isogonic growth, all circles of the same initial radius will be transformed into circles of the same radius, independent of their location[1]. *Uniform anisotropic* growth occurs when growth is the same at all locations, but changes with respect to direction. The overall shape of the surface under anisotropic growth is elongated by different amounts in different directions. For example, two identical circles will transform into the same ellipses, independent of their locations. Isogonic growth is therefore a special case of the uniform anisotropic growth.

Although originally designed as a growth analysis tool, growth tensors can be used for modeling purposes as well [41]. I use the mathematics of growth tensors to implement growth of the background layer. Given a growth tensor corresponding to a desired type of growth, I determine what the analogous velocity vector field should be. Let us consider uniform anisotropic growth in the XY-plane, where a circle of initial radius equal to unity is transformed into an ellipsoid with radii $r_1$ and $r_2$. The angle $\alpha$ represents the maximum direction of elongation, measured from the Y-axis, as illustrated in Figure 3.5. Since planar growth has two mutually perpendicular principal directions of growth [23], it

1. This is different from isotropic growth, under which two identical circles transform into circles, but not necessarily of the same size.

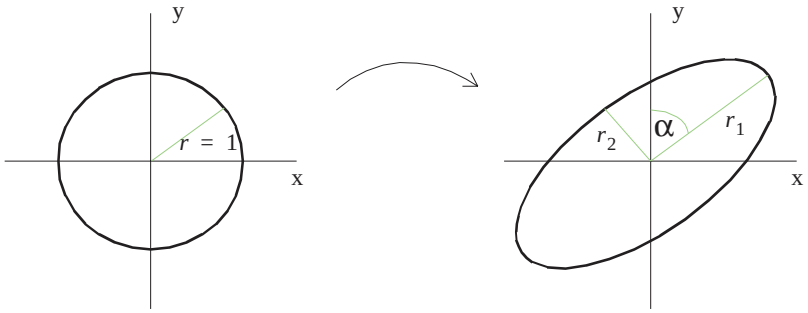



**FIGURE 3.5:** *Anisotropic planar growth: a circle transforms into an ellipse.*

is represented by a growth tensor which has eigenvalues $r_1$, $r_2$ and $0$, and the corresponding eigenvectors $(\sin\alpha, \cos\alpha, 0)$, $(\sin(\alpha + \pi/2), \cos(\alpha + \pi/2), 0)$ and $(0, 0, 1)$, respectively. The last eigenvalue being zero denotes no growth along the Z-axis. The growth tensor satisfying these properties has the form:

$$T_G = \begin{bmatrix} r_1\sin^2\alpha + r_2\cos^2\alpha & (r_1 - r_2)\sin\alpha\cos\alpha & 0 \\ (r_1 - r_2)\sin\alpha\cos\alpha & r_2\sin^2\alpha + r_1\cos^2\alpha & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

From this growth tensor, the corresponding velocity vector field can be determined. This is achieved by substituting the above $T_G$ into Eq. 3.3 and solving for $v = (v_x, v_y, v_z)$. The substitution yields a set of 9 partial differential equations:

$$\begin{bmatrix} \frac{\partial v_x}{\partial x} & \frac{\partial v_x}{\partial y} & \frac{\partial v_x}{\partial z} \\ \frac{\partial v_y}{\partial x} & \frac{\partial v_y}{\partial xy} & \frac{\partial v_y}{\partial z} \\ \frac{\partial v_z}{\partial x} & \frac{\partial v_z}{\partial y} & \frac{\partial v_z}{\partial z} \end{bmatrix} = \begin{bmatrix} r_1\sin^2\alpha + r_2\cos^2\alpha & (r_1 - r_2)\sin\alpha\cos\alpha & 0 \\ (r_1 - r_2)\sin\alpha\cos\alpha & r_2\sin^2\alpha + r_1\cos^2\alpha & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

The solution to these equations is the desired velocity vector field:

$$\begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = \begin{bmatrix} \frac{1}{2}((r_1 + r_2)x + (r_1 - r_2)(y\sin 2\alpha - x\cos 2\alpha)) + C_1 \\ \frac{1}{2}((r_1 + r2)y + (r_1 - r_2)(y\cos 2\alpha + x\sin 2\alpha)) + C_2 \\ C_3 \end{bmatrix},$$

where $C_1$, $C_2$ and $C_3$ are arbitrary constants. These three constants identify which point on the plane is not displaced during growth. For example, by setting $C_1 = C_2 = C_3 = 0$, such a point would be the point of origin $(0, 0, 0)$ (a circle centered at the point of origin would transform into an ellipse whose center would remain at the origin). By setting $r_1 = r_2$, the above velocity vector field would represent isogonic growth. Figure 3.6 illustrates the resulting velocity vector fields for various parameter values of $\alpha$, $r_1$ and $r_2$.

I also consider growth on the cylindrical surfaces of tree trunks. As a tree grows, its trunk increases in size radially, but not in height [59]. Thus, if the main axis of the cylinder coincides with the Z-axis, the velocity vector field will be

*Eq. 3.5*

$$\begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = \begin{bmatrix} rx \\ ry \\ 0 \end{bmatrix},$$

where $r$ denotes the rate of growth of the circular cross-section of the cylinder. A graphical representation of this cylindrical velocity vector field is shown in Figure 3.7 on the left. Due to the fact that the trajectory of each point on the surface of the cylinder is

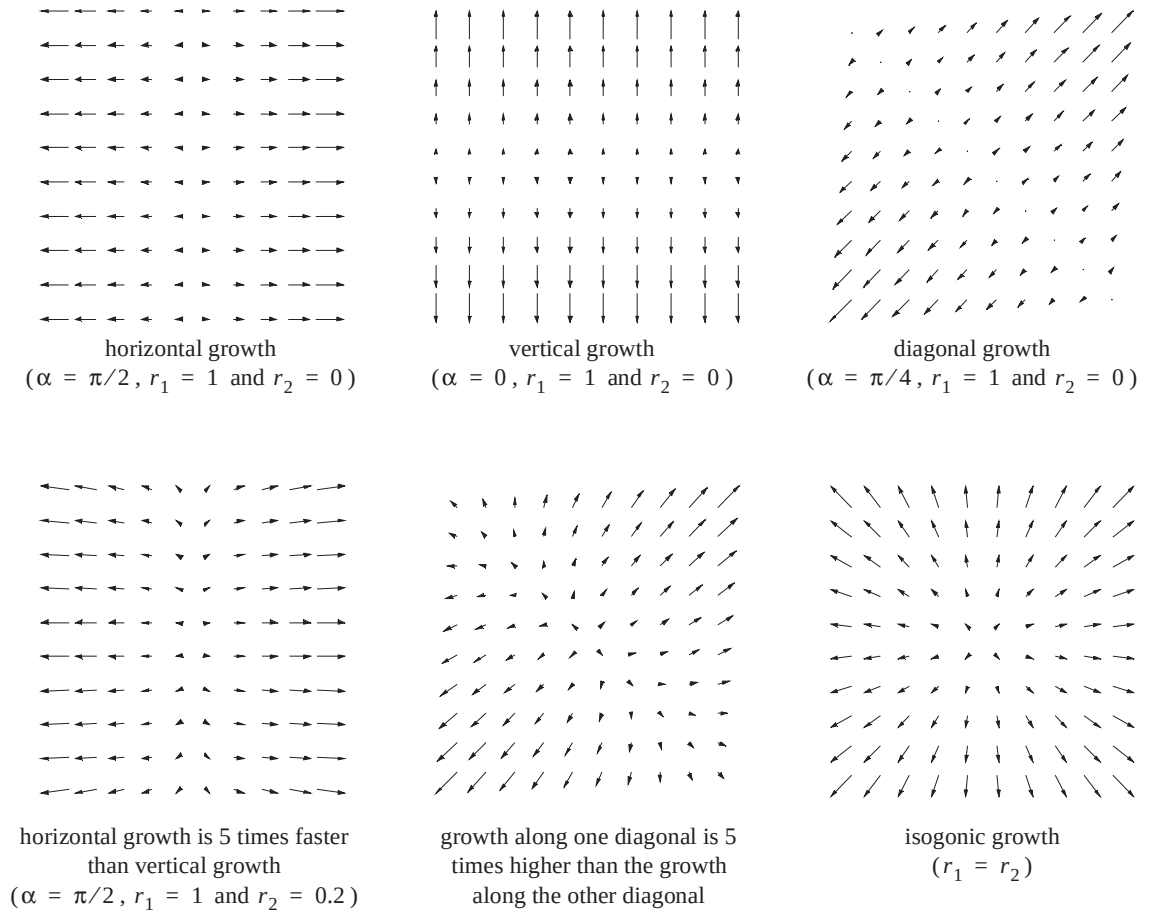



**FIGURE 3.6:** *Velocity vector fields corresponding to anisotropic planar growth.*

perpendicular to the Z-axis, this velocity vector field is identical to that representing isogonic planar growth in the XY-plane. However, the surface of the cylinder under this assumed growth expands anisotropically. For example, a circle drawn on the surface will be transformed into an ellipse. Mathematically, this can be shown by analyzing the growth of the surface at some point P using the growth tensor.

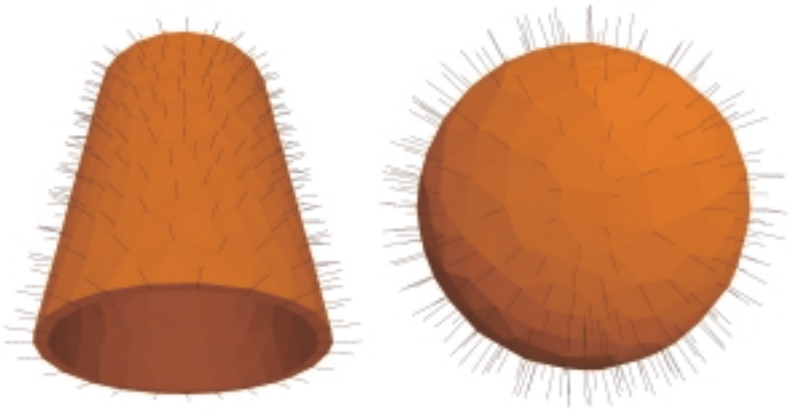


**FIGURE 3.7:** *Cylindrical and spherical velocity vector fields.*

Let point P be defined in cylindrical coordinates as $(\alpha, r, h)$. We want to show that an infinitesimal patch of surface at this point will grow at different rates in different directions. Let us define a local 2D coordinate system $\{\hat{e}_1, \hat{e}_2\}$ of the patch, as illustrated in Figure 3.8. Vector $\hat{e}_1$ is perpendicular to both the Z-axis and to the surface normal at P and vector $\hat{e}_2$ is parallel to the Z-axis, i.e. $\hat{e}_1 = (-\cos\alpha, \sin\alpha, 0)$, $\hat{e}_2 = (0, 0, 1)$. The growth tensor corresponding to the cylindrical growth implemented using the velocity vector field defined in Eq. 3.5 is:

$$T_G = \begin{bmatrix} r & 0 & 0 \\ 0 & r & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

The amount and direction of growth along $\hat{e}_1$ is calculated as $T_G \cdot \hat{e}_1 = (-r\cos\alpha, r\sin\alpha, 0)$, which is non-zero if the rate of growth $r$ is non-zero. On the other hand, the amount of growth in the direction along $\hat{e}_2$ is $T_G \cdot \hat{e}_2 = (0, 0, 0)$, indicating zero growth. Thus, under this assumed cylindrical growth, the surface of a

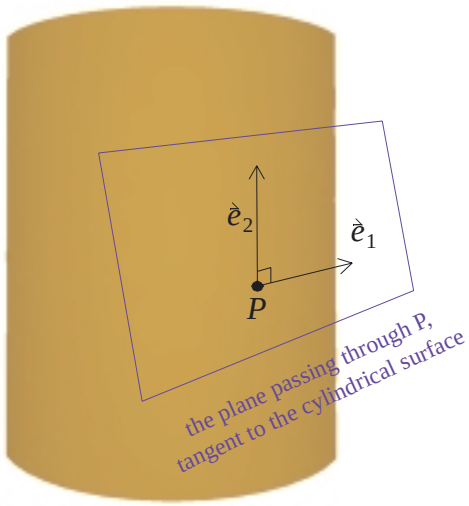



**FIGURE 3.8:** *Local coordinate system of an infinitesimal patch of a cylindrical surface.*

cylinder expands at different rates in different directions. The velocity vector field defined by Eq. 3.5 therefore represents anisotropic growth of the surface.

I also consider growth of a uniformly expanding sphere. To this end, I define the velocity vector field in which the individual velocity vectors are in the directions of the surface normals:

$$\begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = \begin{bmatrix} rx \\ ry \\ rz \end{bmatrix},$$

where $r$ is again the rate of expansion of the sphere. A spherical velocity vector field is illustrated in Figure 3.7 on the right. This velocity vector field corresponds to isogonal

surface growth, which can be again demonstrated mathematically using the growth tensor. The corresponding growth tensor is

$$T_G = \begin{bmatrix} r & 0 & 0 \\ 0 & r & 0 \\ 0 & 0 & r \end{bmatrix}.$$

We want to show that an infinitesimal patch of the spherical surface grows at the same rate in all directions. Let us consider an arbitrary 2D coordinate system of some patch, defined with two unit vectors $\{\grave{e}_1, \grave{e}_2\}$ parallel to the surface at some point P on the sphere. The amount and the direction of growth along $\grave{e}_1$ is $T_G \cdot \grave{e}_1 = r\grave{e}_1$ and along $\grave{e}_2$ it is $T_G \cdot \grave{e}_2 = r\grave{e}_2$, indicating that any two vectors would be transformed by elongation of the same magnitude. This means that a circle drawn on the sphere expanding as described would transform into a circle, and also that the described growth is isogonal.

### 3.3.2. Shrinkage of the material layer

Shrinkage of the material layer is implemented by adjusting the reference shapes of the elements used to discretize the material layer. In the case of the mass-spring models, the rest lengths of the springs are modified, and in the case of the finite element approach, the reference shapes of the wedge elements are adjusted.

### Modeling shrinkage in the mass-spring model

Implementation of the shrinking material layer is straightforward for the mass-spring model. I use the mechanism suggested by Skjeltorp and Meakin, i.e. the rest length $L_i(t)$ of a material spring $i$ at time $t$ is calculated as:

*Eq. 3.6*

$$L_i(t) = L_i(0) \cdot (1 - rt) \ ,$$

where $L_i(0)$ denotes the initial rest length of the material spring and $r$ represents the rate of shrinking. For example, $r = 0.25$ represents a shrinkage rate in which the material springs would be shortened by one fourth of their original rest lengths per one unit of time.

### Modeling shrinkage in the finite element model

It is possible for a material to shrink unevenly, as is the case for drying mud, where the top layer dries faster than the bottom layer. Such uneven rates of shrinkage were not directly considered in my mass-spring model[1], but they were considered in the finite element approach. Consequently, the implementation of shrinkage is slightly more involved in the finite element approach.

In my models, I make the assumption that the shrinkage of the material layer is negative isogonic growth, i.e. that the material shrinks by the same amount in all directions. Two user-defined functions specify the amount of shrinkage over time, one at the top of the material layer, and the other at the bottom. For example, in the case of drying mud, the shrinkage functions might have shapes as shown in Figure 3.9 at the top. The illustrated

1. In the mass-spring model I consider uneven shrinkage only in the post-processing step, used for rendering purposes (Section 4.5).

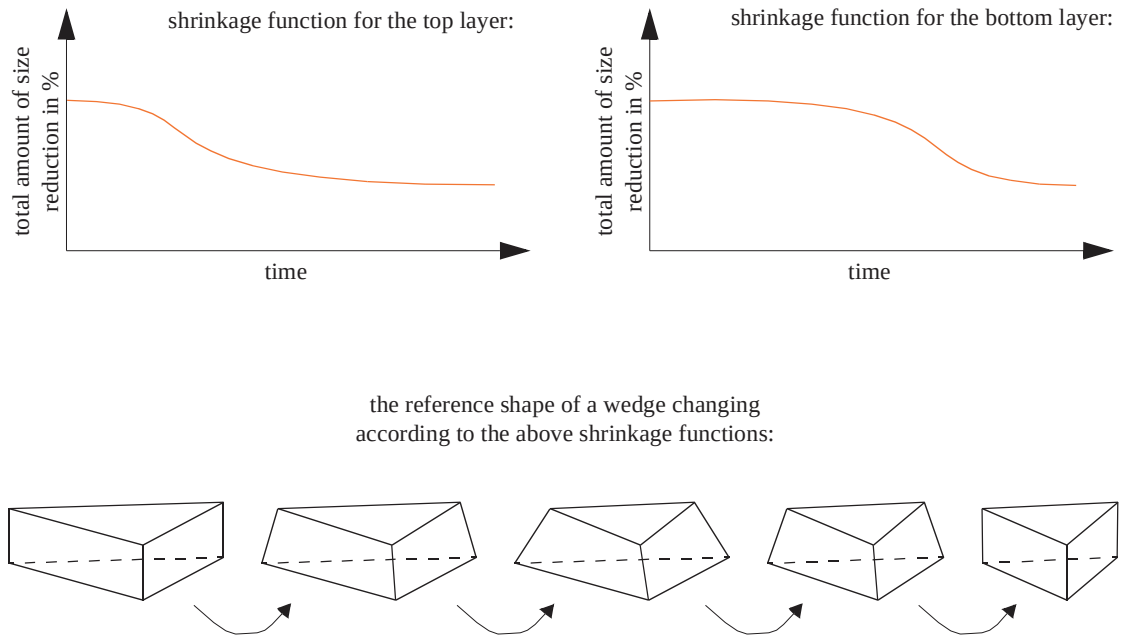



**FIGURE 3.9:** *An example of uneven shrinkage.*

shrinkage functions represent a material layer which immediately starts to shrink at the top, and continues doing so until it is completely dry and cannot shrink any more. The material at the bottom does not shrink until later in the simulation. In the end, both the top and bottom have equally shrunk. The shape of a finite element wedge under this scenario would change as illustrated in Figure 3.9 at the bottom.

Shrinkage of the material layer is modeled by recomputing the reference shapes of the wedge elements and is implemented by the following steps. Each wedge element is defined by its original bottom face triangle and its height. The reference shape of the wedge is computed from these two attributes, taking into consideration the amount of shrinkage at a particular simulation time, as determined by the shrinkage functions. The shapes of the shrunk top and bottom faces are computed independently, by scaling the original bottom face triangle by the amounts specified by the two shrinkage functions.

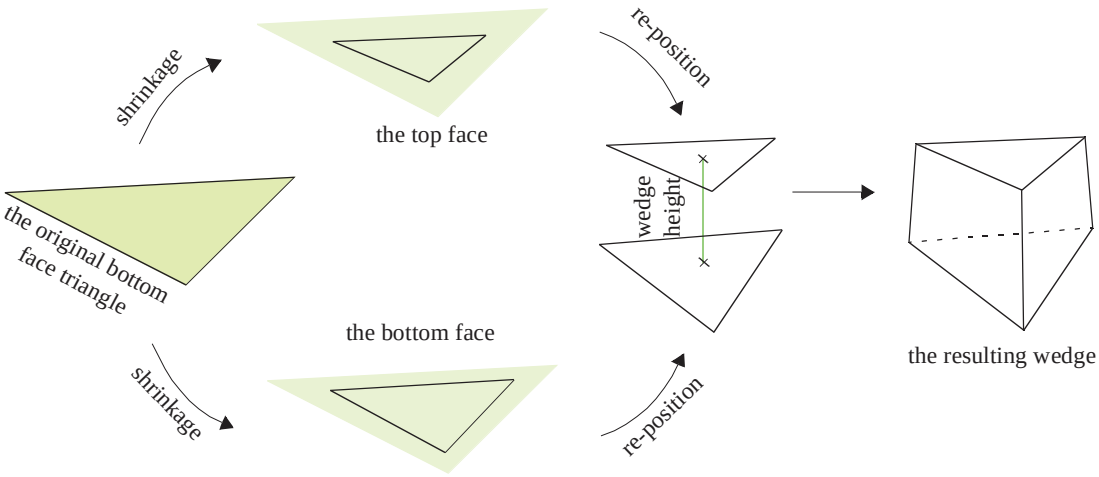



**FIGURE 3.10:** *Construction of the reference shape for the wedge element.*

This produces two triangles, one describing the shape of the top face of the wedge, the other describing the bottom face. The resulting triangles are then positioned so their centroids form a vector, equal to the normal of the original bottom face triangle multiplied by the shrunk height of the element. The amount of shrinkage applied to the height of the wedge is calculated as the average of the shrinkage amounts applied to the top and bottom faces. The triangles are then connected, forming a wedge element. If the shrinkage functions specify zero shrinkage, the above steps amount to the construction of a wedge element by extruding a triangle along its normal. The graphical representation of the steps for constructing the reference shape is shown in Figure 3.10.

In the next two chapters I proceed to show how the described implementation of growth was used in my models to generate fracture patterns on surfaces. First I describe how I extended an existing mass-spring based model of fracture formation by incorporating growth. Then, I demonstrate the application of growth in the framework of finite element methods.

# CHAPTER 4 *The mass-spring approach*

Mass-spring based models are commonly used in computer graphics to simulate various deformations in objects. Their wide-spread use can be attributed to being conceptually simple and straight-forward to implement. The work presented in this chapter was motivated by the success of Skjeltorp and Meakin [61], who used a simple mass-spring system to synthesize convincing fracture patterns in a material composed of a single layer of shrinking microspheres. I have extended Skjeltorp and Meakin model by incorporating growth of the background and applied the extended model to cylindrical surfaces. The goal of these extensions was to investigate whether Skjeltorp's and Meakin's model could be adapted to simulating fracture patterns in tree barks. For the completeness of the presentation, I give a brief overview of the original Skjeltorp and Meakin model first. Then I present my extensions and discuss the results.

## 4.1. Skjeltorp and Meakin model

Skjeltorp's and Meakin's model [61] simulates the behavior of a shrinking microsphere monolayer suspended between two parallel sheets of glass. The spheres are closely

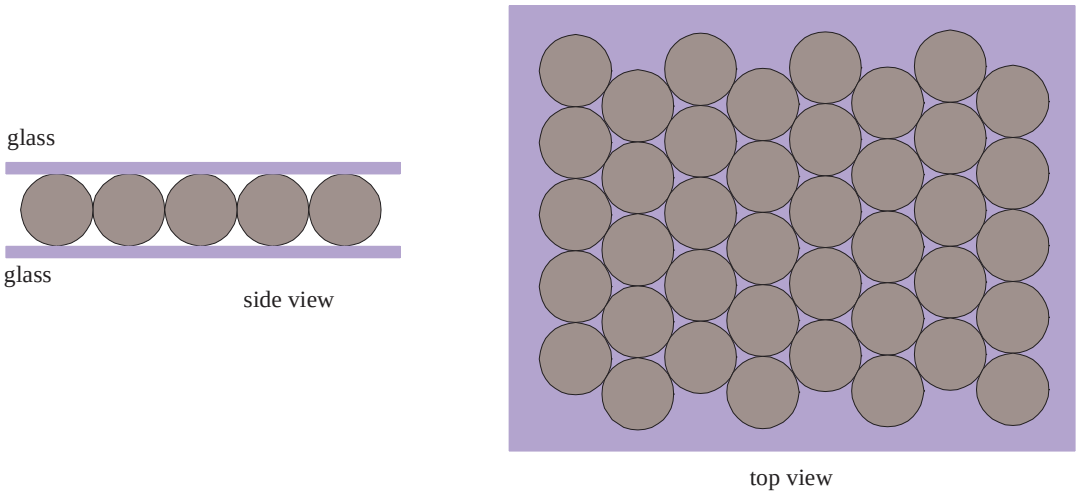



**FIGURE 4.1:** *Monolayer of shrinking micro-spheres.*

packed, as illustrated in Figure 4.1. The material layer (the monolayer of the spheres) is discretized into regularly shaped hexagonal surface segments, approximating the microspheres. These surface segments are modeled as single *point masses*, while the bonds between them are modeled as *material springs* (Figure 4.2). The connections between the spheres and the glass are modeled as *anchor springs*, approximating static as well as kinetic friction. Each anchor spring connects one point mass to the background, attached at an *anchor point.* The properties of the anchor springs are different from the properties of the material springs: they have different stiffness coefficients and their rest lengths are equal to zero. Additionally, anchor springs are allowed to slide when they reach a maximum allowed length. The behavior of all springs in the model is governed by Hooke's spring law, which states that the force exerted by a spring is directly proportional to the amount by which it is stretched.

The shrinkage of the microspheres is modeled by reducing the rest lengths of the material springs. Since each of the point masses are attached to the static background, the shrinkage results in increasing strains in the material layer. Some of the material springs

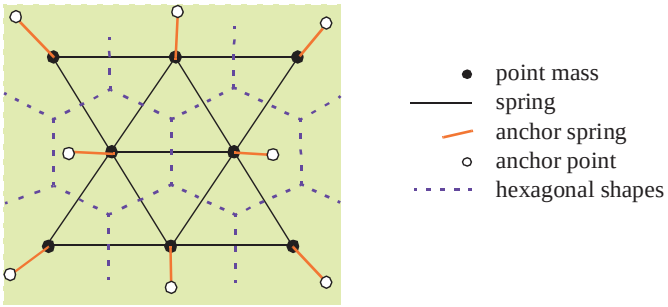



**FIGURE 4.2:** *The mass-spring model.*

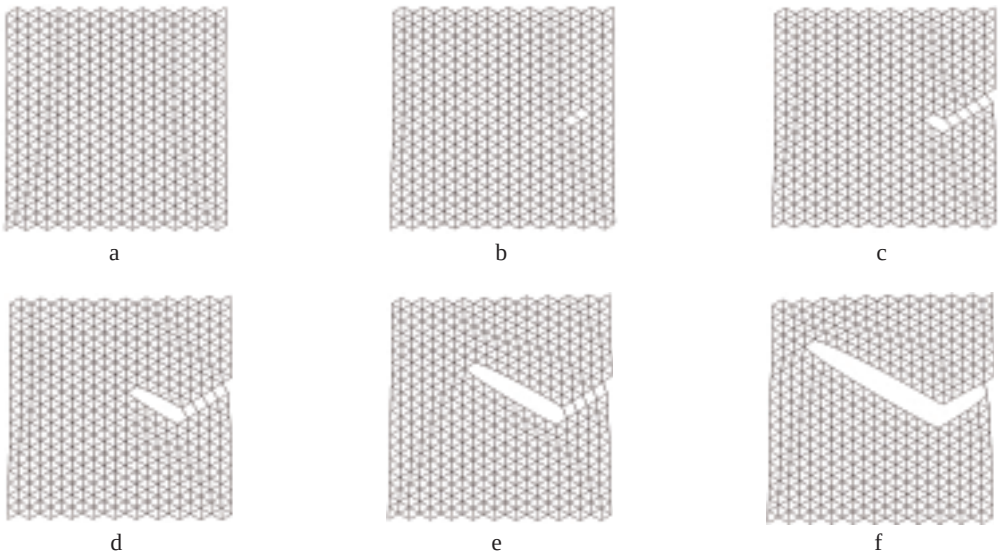



**FIGURE 4.3:** *Fracture propagation simulated using a mass-spring model.*

eventually reach a threshold length, at which point they break. Such a break causes further instability in its immediate vicinity, by decreasing the tensions on some neighboring springs, and increasing it on others. The fracture then propagates, eventually leading to the emergence of a crack pattern (see Figure 4.3).

In Skjeltorp's and Meakin's model, the internal stresses in the material layer and the subsequent fracture formation are a direct consequence of the material layer changing shape (by shrinking) with respect to the static background (the glass supporting the monolayer of spheres). This model is well suited for some patterns, such as ceramics, drying mud or aging paint, where the top layer shrinks while the background remains static. However, some fracture patterns in nature are influenced by growth of the underlying medium. With tree bark, for example, the material layer representing bark may be considered inactive, while the background layer representing the actively growing wood expands. Consequently, I extended Skjeltorp's and Meakin's model by allowing the background layer to grow and applied the extended model to cylindrical surfaces. These extensions are described in detail in the following sections.

## 4.2. Modeling growth

As discussed in the previous chapter, I model the growth of the background surface by updating the coordinates of the anchor points at each simulation time step. The primary difference between the original and the extended model is the cause of stress in the surface. As the background layer grows, the attached material layer is pulled along, creating internal stresses in the material (Figure 4.4). This is different from the original model where stress was induced solely by shrinkage of the material layer.

Figure 4.5 on the left shows the result of simulating isogonic planar growth of a background with respect to a static material layer. Compare this to the result shown in the same figure on the right, which was obtained by allowing the material layer to shrink with respect to a static background. The two patterns are identical, showing (as expected) that isogonic growth of the background has the same effect on the resulting pattern as the shrinkage of the material. It is also observed that fractures have a strong tendency to align

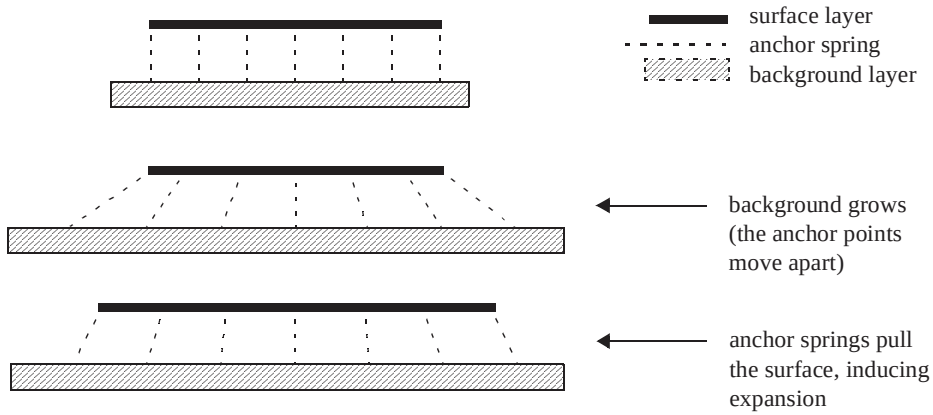
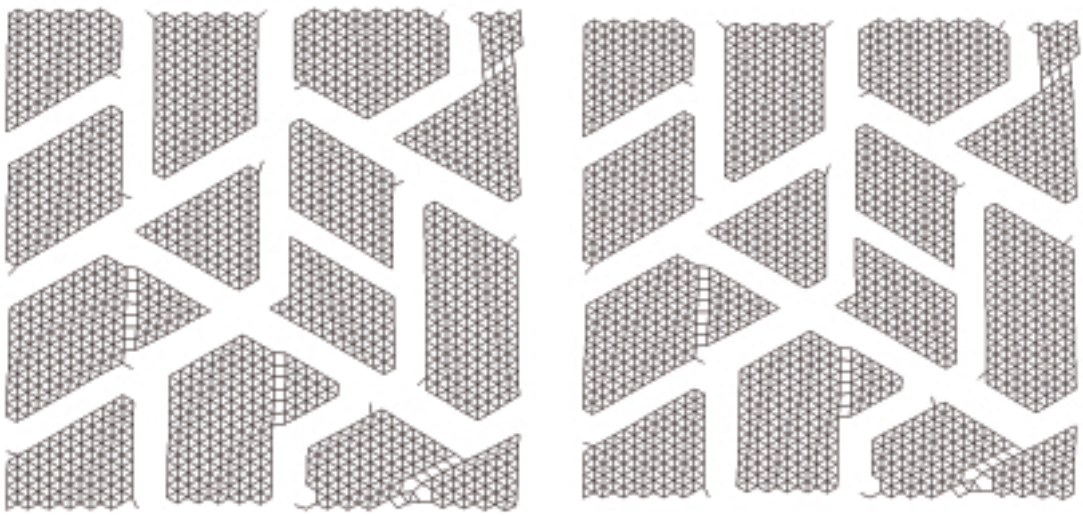



**FIGURE 4.4:** *Growing background induces surface expansion and stress.*



isogonic planar growth of the background

isogonic shrinkage of the material layer

**FIGURE 4.5:** *Synthesized fracture patterns using isogonic growth.*

themselves to the underlying mesh of springs. As a result, fractures form almost exclusively at 60-degree angles with respect to each other, and consequently, there are no horizontal fractures present.

I also simulated anisotropic growth of the background, as this type of growth is characteristic of tree bark. By mapping the cylindrical surface of a tree trunk onto a plane, its growth can be simulated by planar growth in a single direction. The resulting pattern can then be mapped back onto the cylinder. Figure 4.6, on the left, shows the spring patterns obtained by applying anisotropic growth to the background. The top left pattern was obtained by applying growth to the background only in the horizontal direction, and as expected, the resulting fractures were predominantly vertical. The bottom left pattern was simulated by applying vertical growth to the background. Contrary to expectation, the resulting fractures are not predominantly horizontal. Again, it is the hexagonal mesh of springs representing the material layer that prevents the development of horizontal fractures.

By randomly perturbing some of the simulation parameters, the tendency of the fractures to align themselves to the imposed mesh is weakened. This is demonstrated in the patterns shown in Figure 4.6 on the right, generated by perturbing the break lengths of the material springs as described by Skjeltorp and Meakin.

To show that the extended mass-spring model can be applied to non-planar surfaces, I have generated a fracture pattern by allowing the simulation to operate directly on a cylindrical surface. The resulting pattern, shown in Figure 4.7 on the left, was obtained by attaching the anchor springs to a growing cylindrical background layer.

## 4.3. Multi-layer model

The single material layer model only allows for cracks of uniform depth. To model fractures of varying depths, I extended the model by introducing multiple material layers.

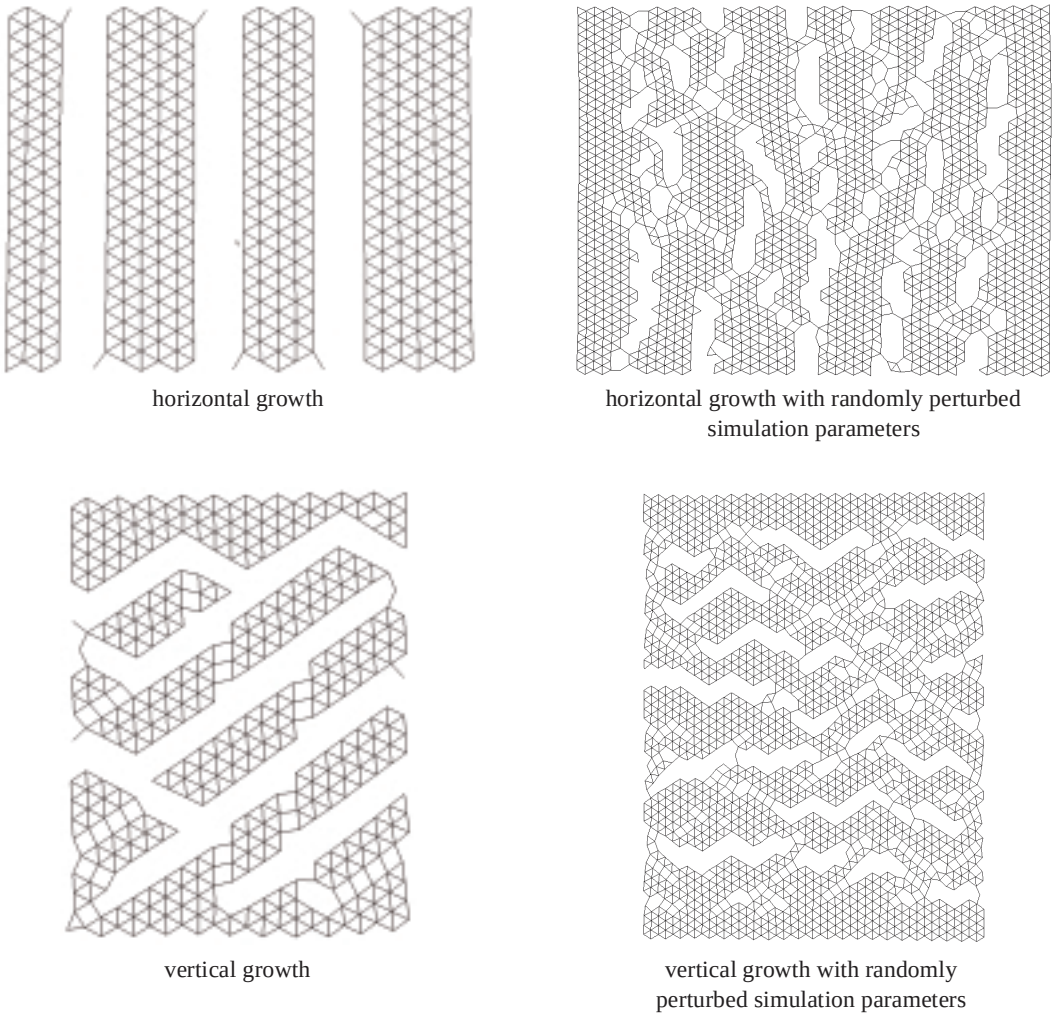



**FIGURE 4.6:** *Synthesized fracture patterns using anisotropic growth.*

The bottom-most material layer was attached to the background through anchor springs. Each of the remaining material layers were attached to each other through inter-layer springs, perpendicular to the surface. The rest lengths of these inter-layer springs represented the thicknesses of the layers. Growth was applied in a manner identical to the two-layer model, by adjusting the positions of the anchor points. A sample result generated by the multi-layer extension is shown in Figure 4.7 on the right.

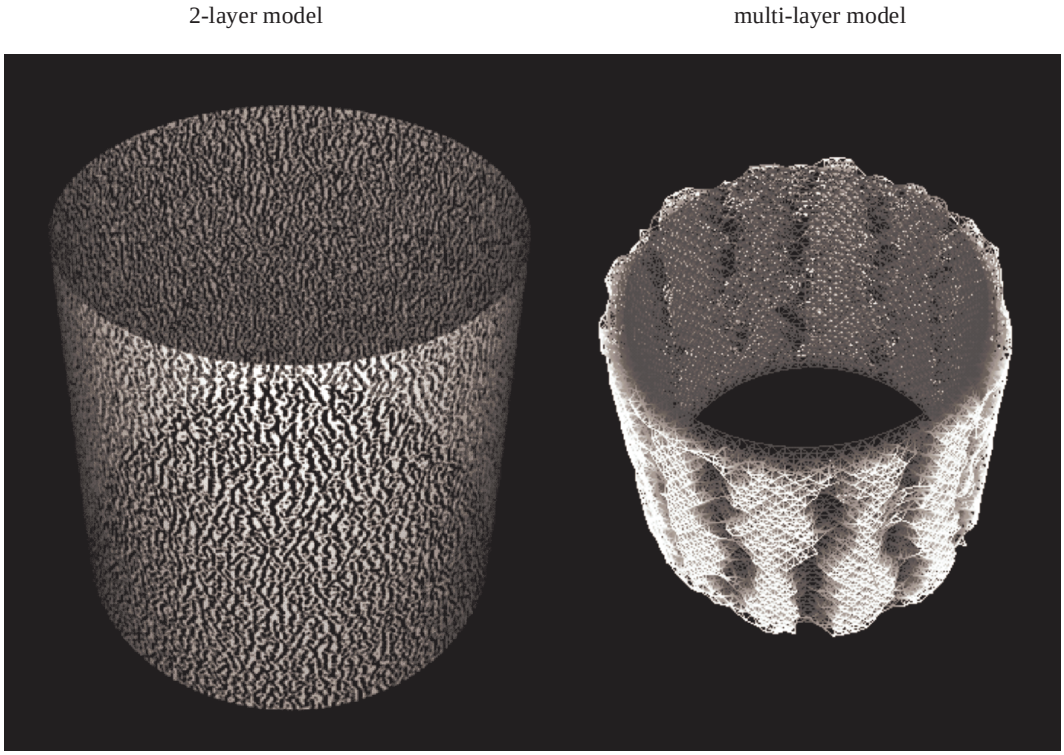



**FIGURE 4.7:** *Fracture patterns obtained on cylindrical surfaces.*

## 4.4. Non-uniform discretization

Because the fracture propagation is modeled by eliminating springs from the model, the fracture can advance only in the direction perpendicular to the spring being removed. Therefore, at the resolution level of a single spring, the underlying mesh will affect the direction of fracture propagation regardless of the type of the mesh used. If the entire mesh of springs is uniform, the local effect of springs on the directionality of the fracture can have a global impact on the overall fracture pattern, as was demonstrated earlier (Figures 4.5 and 4.6).

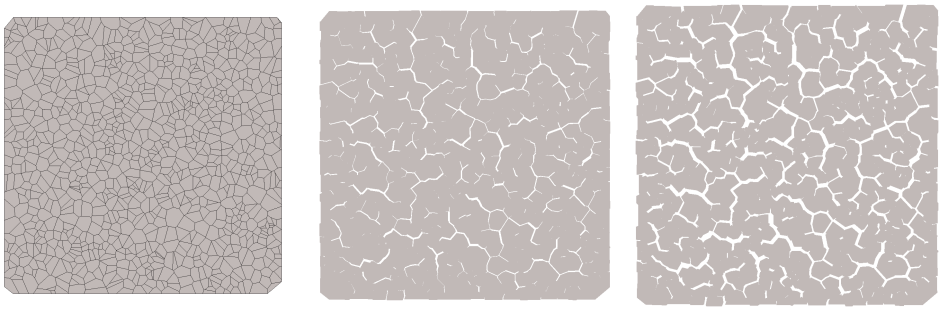



**FIGURE 4.8:** *Fracture pattern synthesized using a randomized surface subdivision.*

To minimize this global effect of the imposed mesh on the directionality of the cracks, I experimented with a pseudo-random surface subdivision. First, a set of points was randomly dispersed over the surface area. The positions of the points were adjusted using a particle repelling method, to prevent two points from being too close to each other (the particle repelling algorithm will be fully described in Section 5.2.1). The resulting points represented the centers of the surface segments and therefore determined the initial positions of the masses in the mass-spring model. The shape of each surface segment was determined by constructing its *Voronoi region* [51] and the mesh of springs connecting the segments was established by constructing the *Delaunay triangulation* - a dual of the Voronoi graph. The mass of each segment was directly proportional to its area. The stiffness coefficient of an anchor spring was proportional to the mass of the segment it connected to the background. The stiffness coefficients of the material springs were set to be directly proportional to the length of the Voronoi edge shared by two surface segments, and inversely proportional to the length of the corresponding Delaunay edge. This method of calculating the values of stiffness coefficients is consistent with the method used by Hirota et al. [24].

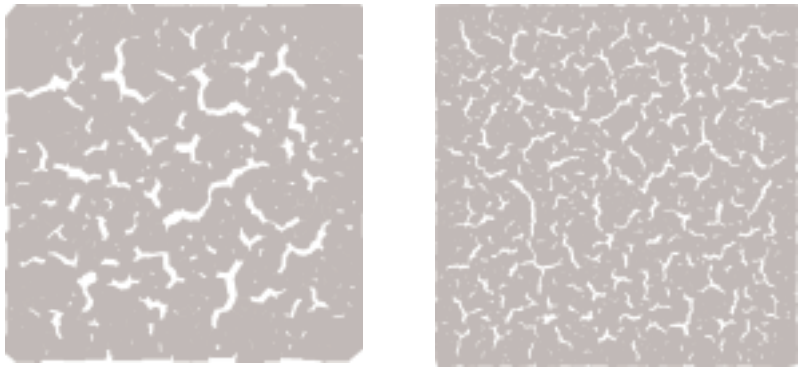



**FIGURE 4.9:** *Generated patterns affected by the level of discretization.*

Figure 4.8 shows the formation of a simulated crack pattern obtained using the described non-uniform surface discretization. The pattern no longer exhibits the global directionality of the fracture pattern observed when using a regular mesh. In this respect, the introduction of the random subdivision is successful. However, the overall appearance of the generated patterns is affected by the level of discretization. The results shown in Figure 4.9 were obtained by using two different discretization levels. The patterns are noticeably different, as the density of the cracks in the pattern on the right is much higher than in the pattern on the left. Also, the generated cracks are much thinner when using a finer discretization. This behavior is undesirable, as the generated pattern should not depend on the level of discretization. I attributed this inconsistency between the generated patterns to the mass-spring model incorrectly representing the continuous properties of the material layer.

## 4.5. Rendering

From the visualization perspective, the inconvenience of using the mass-spring model presented is that the shapes of the individual surface segments in the resulting pattern are not computed. In order to improve the visual presentation of the results, the deformed shapes of the surface segments should be considered. One approach is to incorporate the calculation of the shapes into the simulation. Since this would negatively impact the performance of the simulation, I decided to calculate the shapes of the resulting surface segments in a post-processing step. To this end, I use sub-networks of masses and springs to model the deformations of an individual surface element, similar to the method used by Norton et al. [44]. Since the surface elements are connected to each other, the resulting collection of such sub-networks is used to calculate the deformation of the entire surface layer. To further increase the realism of the results, the post-processing step also considers the 3D aspect of the model - by accounting for the height of the material layer.

The mass-spring sub-network representing each surface element is constructed as follows. First, the 3D shape of each material element is determined by extruding its Voronoi region along the surface normal. The mass-spring sub-networks are then constructed for each material element, by interconnecting all of its nodes with springs. This process is graphically illustrated at the top of Figure 4.10. The rest lengths of the springs are adjusted to reflect the shrinkage of the material layer.

Once the mass-spring systems are constructed for each material element, the walls of adjacent elements are merged as illustrated in the middle of Figure 4.10. The merging is done by considering each pair of elements initially connected by a material spring. If the material spring is not broken, the shared wall of the elements will be completely merged.

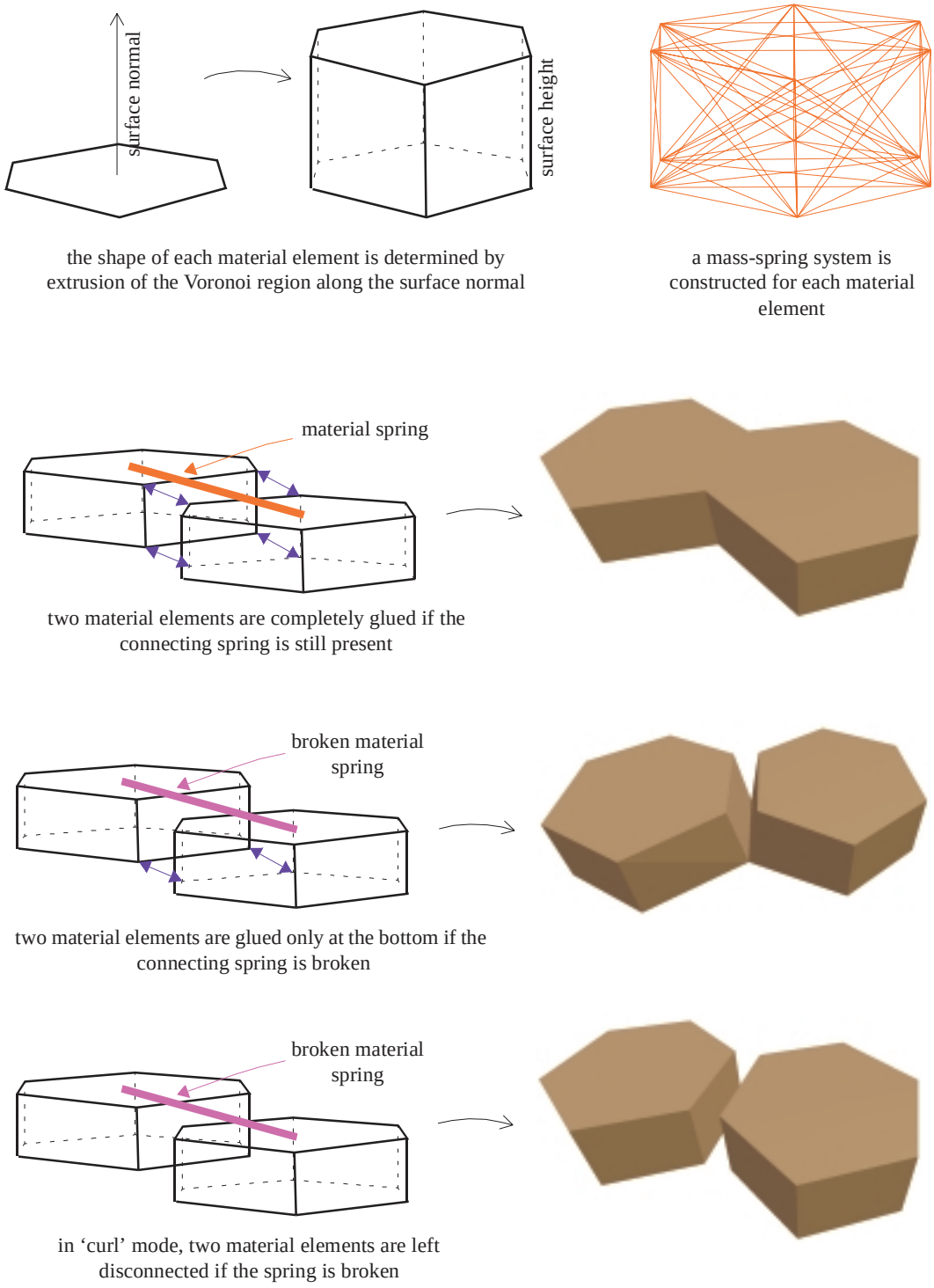


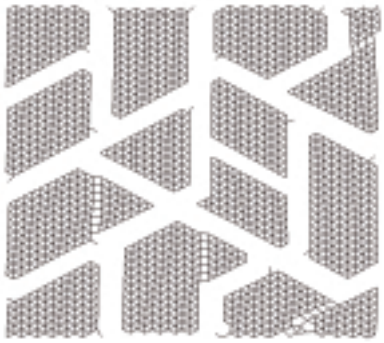

**FIGURE 4.10:** *Graphical illustration of the post-processing step.*

If the material spring is broken, only the lower portion of the wall be joined. After the walls are merged, the deformed shape of the surface layer is obtained by fixing the bottom masses of each element and relaxing the mass-spring system. The result can be then rendered in 3D, as shown in Figure 4.11.

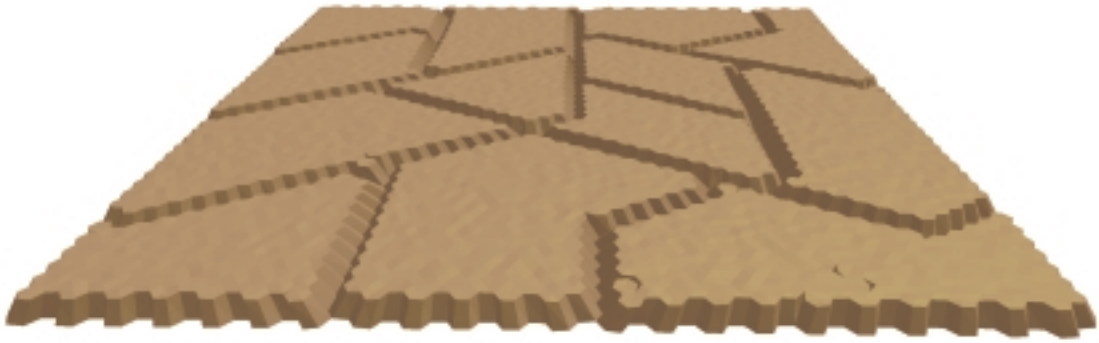
Formation of cracks on differentially growing surfaces is often accompanied by a peeling effect. Once a patch of the material surface has been detached from the surrounding material, its edges may also detach from the underlying background, and then curl upward. This curling phenomenon can be observed in some types of tree bark, and occasionally in drying mud. To simulate such curling phenomenon, I assume that the material layer has completely peeled from the background layer. This is simulated by allowing the masses at the bottom of the elements to move freely, and by not merging walls between elements whose interconnecting material spring is broken. The simulated curling effect is demonstrated in the pattern shown at the bottom of Figure 4.11.

## 4.6. Self-similar patterns

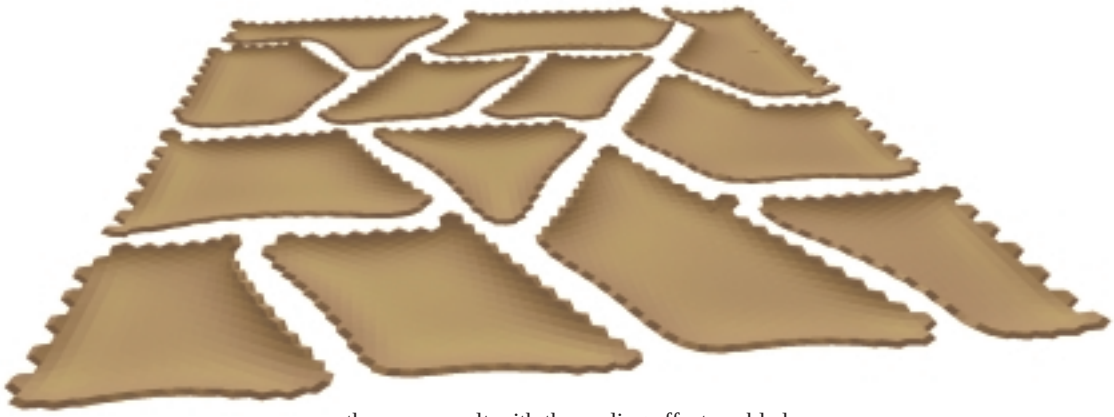
While experimenting with various simulation parameters, I attempted to obtain a model capable of generating a self-similar pattern. Such a recursive pattern should consist of smaller parts that are similar to the overall pattern. To better understand the processes involved when fracture formation is induced by the application of growth, I investigated the issue of self-similar patterns using a one dimensional mass-spring system. Although my work on the 1D mass-spring system was successful, I have not extended its results to surfaces. I have therefore decided to include this work in Appendix A, rather than in this chapter.



the result rendered as a wire-frame model



the result rendered in 3D, using the post-processing step



the same result with the curling effect enabled

**FIGURE 4.11:** *Different methods for rendering the same result.*

The self-similarity issue has been recently addressed and solved in the domain or R-D models by Crampin et al. [7], who successfully generated an infinitely repeating pattern under exponential growth. Since it can be argued that both mass-spring models and R-D models are based on short-range activation and long-range inhibition, the solution to the posed problem of self-similarity should be similar in both domains.

## 4.7. Conclusions

The goal of the work presented in this chapter was to investigate the possibility of adapting Skjeltorp's and Meakin's mass-spring model to simulating fracture formation on growing surfaces, such as tree bark. To this end, I extended their model by incorporating growth of the background and then applied the result to cylindrical surfaces. A minor extension involved introduction of multiple material layers, which allowed simulation of fractures of varying depths. I also presented a method for rendering the results in a visually pleasing fashion, achieved by calculating the shape of the deformed material layer in a post-processing step.

The results obtained by the extended mass-spring model revealed a tendency of fractures to align themselves with the underlying mesh, which became particularly evident when simulating anisotropic growth of the background. I attempted to reduce this directionality of fractures by implementing a non-uniform discretization method. Although the non-uniform discretization resulted in fracture patterns with reduced global directionality, an undesired relationship between the generated pattern and the level of discretization was observed. I attributed the failure of the model to scale to different discretizations to the inability of the mass-spring model to correctly represent the continuum of the material layer. Consequently, I continued my research of modeling crack formation using the more robust technique of finite element methods, which I describe in the next chapter.

# CHAPTER 5 *The finite element approach*

In this section I present an approach for simulating fracture formation in two-layered materials, based on the method of finite elements. The main advantage of the finite element approach over the mass-spring approach is that it produces more correct results for the same level of discretization, e.g. by considering the poission ratio. Additionally, it also allows for dynamic refinement of the discretization. Although an increased accuracy can be attained in mass-spring models by using a finer discretization, it would lead to a substantial increase in computation time. The only disadvantage of the finite element method is that it is more difficult to implement.

The finite element method is a technique originally developed in solid mechanics [5][6][18][39], to approximate the continuous properties of solid structures. FEMs are typically used to numerically calculate stresses and strains in structures under loads. Since their development, finite element methods have been successfully used in many other areas of sciences and engineering [73], for example in fluid dynamics [19], heat conduction [32] and recently in plastic surgery [30].

Generally, FEMs are used when an analytical solution to a given continuous problem is not feasible, and a numerical solution is required. In this sense, FEMs are very similar to

the finite differencing schemes. However, instead of a regular, rectilinear grid used for finite differencing, an arbitrary mesh can be used for finite element methods. The ability to discretize the domain of the problem into elements of many different shapes an sizes is very important, since the domains of many practical applications are not rectangular.

Once the domain is subdivided into finite elements, the goal of the FEM is to numerically calculate the solution to the problem over the discretized domain. This is achieved by first establishing the relationships (or equations) between the values at the nodes of each element. The derivations of the relationships between the values at the nodes in a single element is the essence of finite element methods. Once these relationships are established, the resulting equations for all elements are simultaneously solved to obtain the solution for the entire system of elements.

## 5.1. Overview of the fracture algorithm

The overall structure of the algorithm I implemented for the fracture formation simulation is the standard way for implementing a general FEM [74]. The first step is to discretize the domain of the problem using finite elements. This is achieved by approximating the surfaces on which the fractures develop using triangular wedges. The relationships between the nodes of the elements are then calculated for each element. These equations are then combined into a single set of equations, representing the relationships between all of the nodes in the system of elements.

The algorithm enters the main simulation loop by calculating the equilibrium state of the model, using a relaxation step. After relaxation, the elements are examined to detect whether internal stress anywhere in the material layer has reached a threshold of failure (*threshold stress*). If such a spot is found, a fracture in the material is formed and the simulation returns to the beginning of the loop. If no spot is found where the threshold

```
discretize the surface
for each element calculate the local relationships between its nodes
calculate the global relationship between all nodes in the system
while simulation not done
        relax the model
        if threshold stress exceeded then
                introduce a fracture
        else
                increment simulation time
                apply growth and/or shrinkage
output results
```

**ALGORITHM 1:** *The overall simulation algorithm.*

stress is exceeded, the growth and/or shrinkage is applied, the simulation clock is incremented and another iteration is started. The simulation ends when the simulation clock reaches a desired value. The described algorithm can be represented with the pseudo-code in Algorithm 1. The individual steps of the algorithm will be now explained in full detail.

## 5.2. Surface discretization

Triangular meshes are the simplest and most commonly used meshes in CG for representing surfaces. There are many known techniques for manipulating triangular meshes, such as adaptive refinement or mesh quality control. Because of their wide-spread use, I apply triangular meshes to discretize the modeled surfaces into finite elements.

I implemented the mesh generation process in three steps. A number of vertices are first distributed over the background layer of the modeled surface. These vertices represent the points where the material layer is connected to the background layer. Next, a triangulation of these points is constructed, approximating the surface of the background layer with triangles. Finally, through extrusion of the triangles along the surface normals, the final

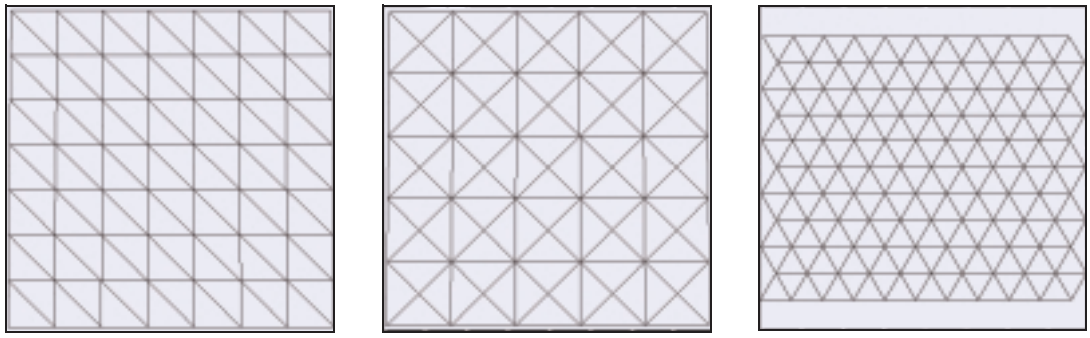


**FIGURE 5.1:** *Examples of uniform discretizations of a planar surface.*

mesh of finite elements is generated. The resulting shape of each element is a 6-node wedge, also called a prism [30], shown in Figure 5.5 on page 55. A detailed description of these steps for planar surfaces follows. For cylindrical and spherical surfaces these steps are modified in a straightforward manner, which is briefly described.

### 5.2.1. Distribution of points on a surface

A uniform distribution of vertices on a planar surface results in regular triangular meshes, such as those illustrated in Figure 5.1. Because the orientations of finite elements can have small but undesired effects on the calculation of stress, and therefore on the directions of fractures, I avoid uniform surface discretizations. A completely random distribution of points, however, leads to generation of meshes which may contain degenerate triangles (see Figure 5.2), and leads to disproportional wedge elements. Degenerate elements are undesirable for use with FEMs, because they may incorrectly approximate the continuum [75]. Consequently, I implemented a ‘semi-random’ distribution of points on surfaces, where the orientations of the resulting triangles are randomly perturbed, but their shapes remain roughly uniform.

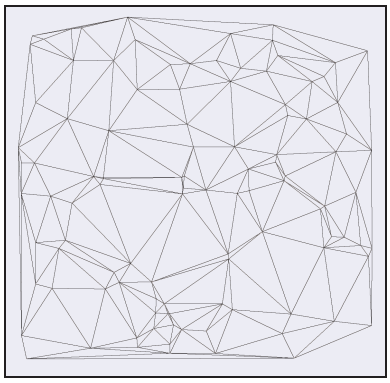


**FIGURE 5.2:** *Triangulation of randomly distributed points on a plane.*

When using random point distribution, the degenerate triangles are formed during triangulation because some vertices are too close to each other. To obtain nicer triangles, a distribution of points is needed, in which the minimum distance between points is above some threshold. To this end, I use a particle repelling technique, similar to that suggested by Witkin and Heckbert in [69], where each point repels every other point in the system. After an initially random configuration of points is established, points are iteratively repelled from each other until an equilibrium is achieved. To avoid a completely uniform configuration of points, I modified the Witkin's and Heckbert's algorithm so that the repelling force field of each point is randomly perturbed. The pseudo-code of this algorithm is straightforward (Algorithm 2).

The repelling force between two points increases as the distance decreases, and vanishes as the distance increases. Once two vertices are more than a certain distance apart, the

```
generate random points on a surface
repeat until an equilibrium is obtained:
        for each point p do:
                p -> total_force = 0
                for each point q that is not p do:
                        p -> total_force += force_between(p, q)
        for each point p do:
                p -> position += scale * p -> total_force
```

**ALGORITHM 2:** *Simple particle repelling algorithm.*

resulting repelling force is zero. The repelling force acts along the vector determined by the line between two vertices. The magnitude of the force is calculated as:

*Eq. 5.1*

$$forcemag = \begin{cases} \left(1 - \frac{d}{rsum}\right)^2 & d \le rsum \\ 0 & d > rsum \end{cases}$$

where $d$ is the distance between two vertices and $rsum$ is the combined radius of the two force fields. The repelling force is zero between any two points more than $rsum$ distance apart. The force field radii of each point must be chosen with some care, so that the average value of $rsum$ is not too large. If it is too large, the points will eventually assume a distribution that is very close to uniform (usually a hexagonal lattice). By experimentation, I found the best results are produced when $rsum$ is kept below the value of $3.2\sqrt{A/n\pi}$, where $A$ is the total area of the planar surface over which the points are distributed.

The time complexity of the above algorithm is $O(n^2)$, because the forces are calculated between every pair of points, making it inefficient for a large number of points. A significant improvement in the running time can be achieved by taking the advantage of the fact that the force function is zero between any two points more than $d_{max}$ distance

```
generate_random_points_on_a_surface()
for each point p do:
        assign a square to p
repeat until equilibrium is achieved:
        for each point p do:
                p -> toal_force = 0
                for each point q in the adjacent 9 squares & p != q do:
                        p -> total_force += force_between(p, q)
        for each node p do:
                p -> position += scale * p -> total_force;
                move p to a new square if needed
```

**ALGORITHM 3:** *Efficient particle repelling algorithm using square grid.*

apart. $d_{max}$ is calculated as the sum of the two largest force field radii. Specifically, when calculating the total force acting on a point, only the neighbors less than $d_{max}$ away must be considered. To this end, I subdivide the entire bounding box area into squares of size $d_{max}$ x $d_{max}$. Each square then keeps track of all points inside it. When the total force acting on a vertex $v$ has to be calculated, the algorithm needs to check only the vertices in the nine squares immediately adjacent to the square in which $v$ is located. Heckbert also suggested a similar performance improvement to his earlier version of the particle repelling algorithm in [22]. The pseudocode for the algorithm I implemented is given Algorithm 3, which is very similar to the technique used by Wyvill et. al [71].

The average running time of this algorithm depends on the average number of points in each square. Assuming a random distribution of points, the average number of points in each square will be constant, and therefore the average running time of the inner ‘for’ loop will be constant as well. With efficient data structures, the overhead of keeping track of points inside the squares is constant, and the average running time of the entire algorithm is then $O(n)$. Practical experiments confirmed the linear running time of this relaxation algorithm.

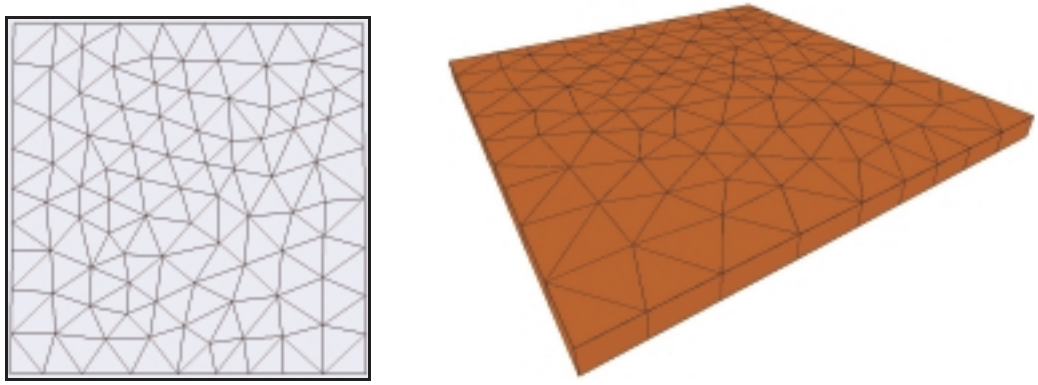
The behavior of the points crossing the boundaries of the bounding box can be implemented in one of two ways. The point encroaching the boundary is either snapped to the closest boundary point, or it is moved to the opposite boundary edge (simulating a torus manifold).

To obtain a random distribution of points on a cylinder, the surface of the cylinder is mapped onto a plane. The points are then distributed over the flat surface as described above. The resulting vertices are then mapped back onto the cylinder. To distribute vertices on a sphere I also employ a particle repelling algorithm, operating directly on spherical surfaces. Since the repelling force could actually move the points off the sphere, they are mapped back onto the sphere after every iteration.

A different approach for obtaining a pseudo-random distribution of points was suggested by McCool and Fiume [36]. In this approach, new random points are iteratively added to a set of existing points, but only if no other points are within some radius. To guarantee that the algorithm terminates, the radius is reduced by some small amount at each step. Once the desired number of points is obtained, Lloyd's relaxation [33] is applied to eliminate points from being too close to each other.

### 5.2.2. Triangulation

Once the desired distribution of points on a surface has been generated, a Delaunay triangulation is constructed [51]. I chose Delaunay triangulation because it results in a mesh where the minimum angle of any triangle is maximized. Many algorithms for obtaining Delaunay triangulations exist, differing in both the complexity of their implementation and the worst case running time. I use the approach in which the problem of triangulation is transformed into a problem of finding a convex hull, which is both



wire-frame model - top view 3D perspective view

**FIGURE 5.3:** *A planar material layer subdivided using wedge elements.*

simple to implement and achieves the best possible running time (which for Delaunay triangulation is $O(n\log(n))$ [51].

The triangulation on a planar surface is obtained by first projecting the set of points onto a paraboloid using the coordinate transformation:

$$x_{new} = x_{old}\ , y_{new} = y_{old} \text{ and } z_{new} = x_{old}^2 + y_{old}^2\ .$$

A convex hull of the projected points is then constructed, yielding a set of triangles. The triangles facing downward are then mapped back onto the planar surface, resulting in the Delaunay triangulation. Triangulation of the cylindrical surface is achieved by mapping the surface of the cylinder onto a plane. The triangulation is then executed as described above. Finally, the resulting triangulation is transferred back onto a cylinder. Examples of a cylindrical triangulation is shown in Figure 5.4 on the left. Since the surface of a sphere is its own convex hull, it is trivial to find the Delaunay triangulation of points on a spherical surface. The Delaunay triangulation is simply the 3D convex hull of the points.

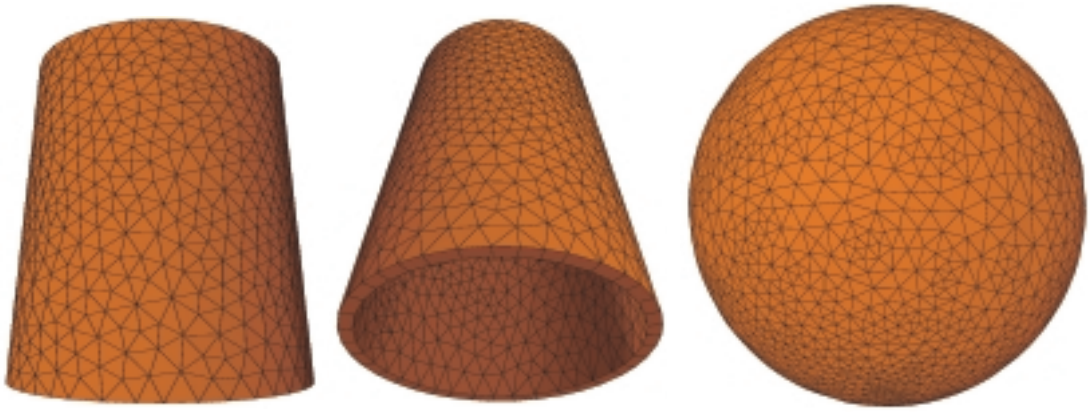


**FIGURE 5.4:** *Delaunay triangulation on cylindrical and spherical surfaces.*

An example of a Delaunay triangulation of points on a spherical surface is shown in Figure 5.4 on the right.

Once the triangulation of a surface is complete, the reference shapes of the individual finite elements are computed as was outlined in Chapter 3.

## 5.3. The element stiffness matrix

In this section I derive the mathematical equations by which a wedge element approximates the continuum it represents. These equations provide a method by which it will be possible to calculate the forces exerted by an element at its nodes for a given set of nodal displacements. Calculation of such forces is important in order to determine the stable state of the overall system of elements, or its deformation. I also show how these equations can be used to calculate strains and stresses inside the element as well as at its nodes. For the mechanical properties of the materials modeled in my research I assume linear elasticity, i.e. the stress generated inside the material is linearly proportional to the

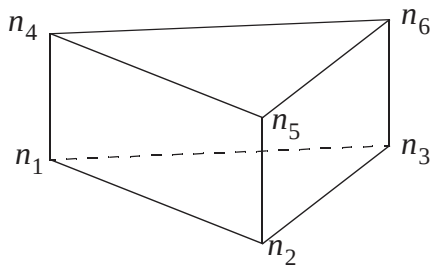



**FIGURE 5.5:** *A wedge element.*

strain. Although the information provided in this section is standard to FEMs [74], I am including it in my thesis for the completeness of the presentation.

A displacement of any node (nodal displacement) is represented by a vector in $\Re^3$. Equivalently, using the FEM terminology, every node has 3 degrees of freedom (DOF). Since there are 6 nodes in a wedge element, each element has 18 DOF.

Under the assumed linear elasticity model, the forces generated by an element at its nodes are linearly related to the nodal displacements. Using matrix notation, this relationship can be compactly written as:

*Eq. 5.2*
$$[F_e] = [K_e][q_e] \ ,$$

which represents 18 linear equations. The matrix $[F_e]$ is an 18x1 column vector denoting a set of 6 force vectors exerted by the element at the nodes:

*Eq. 5.3*
$$[F_e] = [f_x^1, f_y^1, f_z^1, f_x^2, f_y^2, f_z^2, ..., f_x^6, f_y^6, f_z^6]^T \ ,$$

*Eq. 5.4*
where $[f_x^i, f_y^i, f_z^i]$ is the force vector at node $i$.

The matrix $[q_e]$ is an 18x1 column vector representing the nodal displacement with respect to the element's undeformed state:

*Eq. 5.5*

$$[q] = [q_x^1, q_y^1, q_z^1, q_x^2, q_y^2, q_z^2, ..., q_x^6, q_y^6, q_z^6],$$

where each $\left[q_x^i, q_y^i, q_z^i\right]$ denotes the displacement of node $i$. The nodal displacements define the deformation of the element's shape.

Finally, $[K_e]$ is an 18x18 element stiffness matrix. The element stiffness matrix contains all of the coefficients needed to express the linear relationship between $[q_e]$ and $[F_e]$. The derivation of $[K_E]$ is described next.

### 5.3.1. Isoparametric coordinates

The use of Cartesian coordinates can become quite cumbersome when dealing with non-rectangular elements, such as the wedge element used in my work. An alternative coordinate system, which is more suited to the non-rectangular shape of the element, can be used instead. Such coordinate systems are often called natural or isoparametric coordinates. The isoparametric coordinates $(\xi, \eta, \zeta)$ I have chosen for the 6-node wedge element are shown in Figure 5.6.

### 5.3.2. Shape functions

Finite elements approximate the continuum they represent by interpolating the values at their nodes throughout the element. For the 6-node wedge element I used multi-linear interpolation, although higher-order interpolations can also be used by adding extra nodes

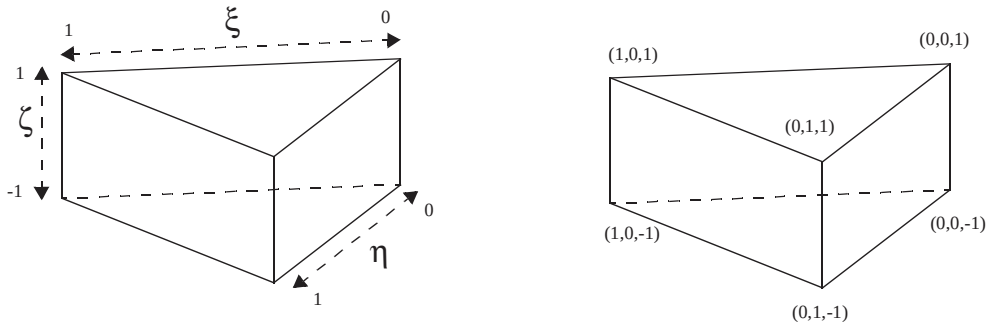



**FIGURE 5.6:** *Isoparametric coordinates of the wedge element.*

to the element. The interpolation inside finite elements is achieved using shape functions [74].

Given a set of some known values $v_1, v_2, ..., v_6$ at the nodes of an element, a value $v$ anywhere inside the element, at a point specified by isoparametric coordinates $(\xi, \eta, \zeta)$, is calculated as:

*Eq. 5.6*
$$v(\xi, \eta, \zeta) = \sum_{i=1}^{6} v_i N_i(\xi, \eta, \zeta) \ .$$

The functions $N_i(\xi, \eta, \zeta)$ are called shape functions. For multi-linear interpolation, they are linear polynomials with respect to each of the variables $\xi, \eta, \zeta$ and yield values between 0 and 1 anywhere inside the element. To preserve inter-element continuity at the nodes, the following conditions must also hold:

*Eq. 5.7*
$$N_i(\xi_j, \eta_j, \zeta_j) = 0 \text{ for } i \neq j \ , \text{ and}$$

*Eq. 5.8*
$$N_i(\xi_j, \eta_j, \zeta_j) = 1 \text{ for } i = j \ ,$$

where $\xi_j, \eta_j, \zeta_j$ denote the isoparametric coordinates of the $j$-th node. The shape functions satisfying these conditions for the wedge element are:

*Eq. 5.9* $$N_1(\xi, \eta, \zeta) = \frac{1}{2}\xi(1-\zeta) ,$$

*Eq. 5.10* $$N_2(\xi, \eta, \zeta) = \frac{1}{2}\eta(1-\zeta) ,$$

*Eq. 5.11* $$N_3(\xi, \eta, \zeta) = \frac{1}{2}(1-\xi-\eta)(1-\zeta) ,$$

*Eq. 5.12* $$N_4(\xi, \eta, \zeta) = \frac{1}{2}\xi(1+\zeta) ,$$

*Eq. 5.13* $$N_5(\xi, \eta, \zeta) = \frac{1}{2}\eta(1+\zeta) \text{ and}$$

*Eq. 5.14* $$N_6(\xi, \eta, \zeta) = \frac{1}{2}(1-\xi-\eta)(1+\zeta) .$$

The values which need to be interpolated over the volume of the element are vectors in $\Re^3$. The Eq. 5.6 can still be used for this purpose, replacing the scalar $v$ by a vector $[v]$. However, the shape functions are often assembled into a single shape function matrix $[N(\xi, \eta, \zeta)]$:

*Eq. 5.15* $$[N(\xi, \eta, \zeta)] = \begin{bmatrix} N_1 & 0 & 0 & N_2 & 0 & 0 & & N_6 & 0 & 0 \\ 0 & N_1 & 0 & 0 & N_2 & 0 & \ldots & 0 & N_6 & 0 \\ 0 & 0 & N_1 & 0 & 0 & N_2 & & 0 & 0 & N_6 \end{bmatrix} .$$

This allows the Eq. 5.6 to be written in a more compact, matrix form:

*Eq. 5.16* $$[v(\xi, \eta, \zeta)] = [N(\xi, \eta, \zeta)][V]$$

where $[V]$ a 1x18 matrix, collectively denoting the 6 vectors to be interpolated. The shape function matrix is very handy as it simplifies the equations in many situations. For example, it can be used to convert isoparametric coordinates $(\xi, \eta, \zeta)$ to Cartesian coordinates $(x, y, z)$ through the equation:

*Eq. 5.17*
$$[x(\xi, \eta, \zeta), y(\xi, \eta, \zeta), z(\xi, \eta, \zeta)]^T = [N(\xi, \eta, \zeta)][Q_e] \ ,$$

where $[Q_e]$ is an 18x1 matrix containing the Cartesian coordinates for all 6 nodes when the element is in the undeformed shape. The matrix $[Q_e]$ is defined as:

*Eq. 5.18*
$$[Q_e] = [x_1, y_1, z_1, x_2, y_2, z_2, \ \dots \ , x_6, y_6, z_6]^T \ .$$

Shape functions are used to interpolate the displacement vector $(u, v, w)$ at an isoparametric coordinate $(\xi, \eta, \zeta)$. Let $u(\xi, \eta, \zeta)$ denote the displacement in the X direction, $v(\xi, \eta, \zeta)$ denote the displacement in the Y direction, and $w(\xi, \eta, \zeta)$ denote the displacement in the Z direction. These displacements are linearly interpolated over the element, using the shape functions:

*Eq. 5.19*
$$u(\xi, \eta, \zeta) = N_1(\xi, \eta, \zeta)q_x^1 + N_2(\xi, \eta, \zeta)q_x^2 + \dots + N_6(\xi, \eta, \zeta)q_x^6 \ ,$$

*Eq. 5.20*
$$v(\xi, \eta, \zeta) = N_1(\xi, \eta, \zeta)q_y^1 + N_2(\xi, \eta, \zeta)q_y^2 + \dots + N_6(\xi, \eta, \zeta)q_y^6 \ ,$$

*Eq. 5.21*
$$w(\xi, \eta, \zeta) = N_1(\xi, \eta, \zeta)q_z^1 + N_2(\xi, \eta, \zeta)q_z^2 + \dots + N_6(\xi, \eta, \zeta)q_z^6 \ .$$

Equivalently, the above can be expressed compactly using the matrix notation:

*Eq. 5.22*
$$\begin{bmatrix} u(\xi, \eta, \zeta) \\ v(\xi, \eta, \zeta) \\ w(\xi, \eta, \zeta) \end{bmatrix} = [N(\xi, \eta, \zeta)][q_e]^T$$

Since the derivatives of the shape functions will be needed later for some calculations, I define here a matrix $[N_{\xi\eta\varepsilon}(\xi, \eta, \zeta)]$ containing all of the derivatives of the shape functions with respect to the isoparametric coordinates:

*Eq. 5.23*

$$[N_{\xi\eta\varepsilon}(\xi, \eta, \zeta)] = \begin{bmatrix} \frac{\partial N_1}{\partial \varepsilon} & \frac{\partial N_2}{\partial \varepsilon} & \frac{\partial N_3}{\partial \varepsilon} & \frac{\partial N_4}{\partial \varepsilon} & \frac{\partial N_5}{\partial \varepsilon} & \frac{\partial N_6}{\partial \varepsilon} \\ \frac{\partial N_1}{\partial \eta} & \frac{\partial N_2}{\partial \eta} & \frac{\partial N_3}{\partial \eta} & \frac{\partial N_4}{\partial \eta} & \frac{\partial N_5}{\partial \eta} & \frac{\partial N_6}{\partial \eta} \\ \frac{\partial N_1}{\partial \xi} & \frac{\partial N_2}{\partial \xi} & \frac{\partial N_3}{\partial \xi} & \frac{\partial N_4}{\partial \xi} & \frac{\partial N_5}{\partial \xi} & \frac{\partial N_6}{\partial \xi} \end{bmatrix} .$$

For the wedge element, using the linear interpolation functions, the matrix $[N_{\xi\eta\varepsilon}(\xi, \eta, \zeta)]$ is:

*Eq. 5.24*

$$[N_{\xi\eta\varepsilon}(\xi, \eta, \zeta)] = \frac{1}{2}\begin{bmatrix} 1-\zeta & 0 & -(1-\zeta) & 1+\zeta & 0 & -(1+\zeta) \\ 0 & 1-\zeta & -(1-\zeta) & 0 & 1+\zeta & -(1+\zeta) \\ -\xi & -\eta & -(1-\xi-\eta) & \xi & \eta & 1-\xi-\eta \end{bmatrix}.$$

A higher degree interpolation of nodal values over an element could be used as well, in which case the matrix $[N_{\xi\eta\varepsilon}(\xi, \eta, \zeta)]$ would have to reflect the corresponding derivatives. An alternative approach to refining the interpolation of nodal values over an element was very recently suggested by Grinspun et. al. [21], which I will briefly review in Chapter 6.

### 5.3.3. Derivation of the element stiffness matrix

The process for obtaining the element stiffness matrix $[K_E]$ will now be described. The general formula for calculating the elemental stiffness matrix $[K_E]$ is [75]:

*Eq. 5.25*

$$[K_E] = \iiint_{x\,y\,z} [B]^T [D][B] dz dy dx ,$$

which is a volume integral of a set of functions written in the matrix notation. The definition of the matrices $[B]$ and $[D]$ will be explained later. The integral in the above

formula is specified in the Cartesian coordinate system. Unless the element is rectangular, this integral is difficult to evaluate. To simplify the task, a change of coordinates is commonly used [74]. Instead of using the Cartesian coordinate system, the integral is re-written in the element's isoparametric coordinates. Using the substitution $dxdydz = |det[J]|d\zeta d\eta d\xi$ we can now rewrite Eq. 5.25 as:

*Eq. 5.26*
$$[K_E] = \int_{\xi=0}^{1}\int_{\eta=0}^{1-\xi}\int_{\zeta=-1}^{1}[B]^T[D][B]|det[J]|d\xi d\eta d\zeta$$

where $[J] = [J(\xi, \eta, \zeta)]$ is the Jacobian matrix at coordinates $(\xi, \eta, \zeta)$:

*Eq. 5.27*
$$[J(\xi, \eta, \zeta)] = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \\ \frac{\partial x}{\partial \zeta} & \frac{\partial y}{\partial \zeta} & \frac{\partial z}{\partial \zeta} \end{bmatrix} \text{ evaluated at } (\xi, \eta, \zeta) \ .$$

In order to evaluate $[J]$ as defined above, we need to know the Cartesian coordinates of a point $(\xi, \eta, \zeta)$. To this end we can use Eq. 5.17. Since $[Q]$ is constant, we can write:

*Eq. 5.28*
$$[J(\xi, \eta, \zeta)] = [N_{\varepsilon\eta\xi}(\xi, \eta, \zeta)][Q]^T \ ,$$

where $[Q]$ is a 3x6 matrix containing the nodal coordinates of an element:

*Eq. 5.29*
$$[Q] = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \\ y_1 & y_2 & y_3 & y_4 & y_5 & y_6 \\ z_1 & z_2 & z_3 & z_4 & z_5 & z_6 \end{bmatrix}$$

The Jacobian matrix $[J]$ relates the first order derivatives in Cartesian coordinates to first order derivatives in isoparametric coordinates. For example, the corresponding first order

derivatives of the shape functions in both coordinate systems are related through the following equations:

*Eq. 5.30*

$$\begin{bmatrix} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \\ \frac{\partial N_i}{\partial \zeta} \end{bmatrix} = [J] \begin{bmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \\ \frac{\partial N_i}{\partial z} \end{bmatrix} \quad \text{and inversely,} \quad \begin{bmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \\ \frac{\partial N_i}{\partial z} \end{bmatrix} = [J]^{-1} \begin{bmatrix} \frac{\partial N_i}{\delta \xi} \\ \frac{\partial N_i}{\partial \eta} \\ \frac{\partial N_i}{\partial \zeta} \end{bmatrix} .$$

**Material stiffness matrix [D]**

Matrix $[D]$ in Eq. 5.26 is called a material stiffness matrix, and it relates the stress tensor $[\sigma] = [\sigma_x, \sigma_y, \sigma_z, \tau_{xy}, \tau_{yz}, \tau_{zx}]$ to strain tensor $[\varepsilon] = [\varepsilon_x, \varepsilon_y, \varepsilon_z, \gamma_{xy}, \gamma_{yz}, \gamma_{zx}]$ by the following equation, assuming linear elasticity:

*Eq. 5.31*

$$[\sigma] = [D][\varepsilon]$$

In my models I assume that the material layer is composed of homogeneous material. The material stiffness matrix is therefore constant throughout an entire element, as it only depends on the values of the Young modulus $E$ and Poisson ratio $\upsilon$ [16]:

*Eq. 5.32*

$$[D] = \frac{E}{(1+\upsilon)(1-2\upsilon)} \begin{bmatrix} 1-\upsilon & \upsilon & \upsilon & 0 & 0 & 0 \\ \upsilon & 1-\upsilon & \upsilon & 0 & 0 & 0 \\ \upsilon & \upsilon & 1-\upsilon & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1-2\upsilon}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1-2\upsilon}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1-2\upsilon}{2} \end{bmatrix} .$$

**Matrix [B]**

Matrix $[B] = [B(\xi, \eta, \zeta)]$ is a 6x18 matrix of coefficients, describing the linear relationship between the displacement vector $[q]$ and strain $[\varepsilon]$ at a point $(\xi, \eta, \zeta)$. The implicit definition of $[B(\xi, \eta, \zeta)]$ is given by the equation:

*Eq. 5.33*
$$[\varepsilon(\xi, \eta, \zeta)]^T = [B(\xi, \eta, \zeta)][q]^T$$

where $[\varepsilon(\xi, \eta, \zeta)]$ represents the strain at point $(\xi, \eta, \zeta)$. The components of the strain are defined as [16]:

*Eq. 5.34*
$$\varepsilon_x = \frac{\partial u}{\partial x}, \varepsilon_y = \frac{\partial v}{\partial y}, \varepsilon_z = \frac{\partial w}{\partial z},$$

$$\gamma_{yz} = \frac{\partial v}{\partial z} + \frac{\partial w}{\partial y}, \gamma_{xz} = \frac{\partial u}{\partial z} + \frac{\partial w}{\partial x}, \gamma_{xy} = \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}.$$

To calculate the first component $\varepsilon_x$ of the strain tensor, we use the definition of $u$ from Eq. 5.19:

$$u = N_1 q_x^1 + N_2 q_x^2 + N_3 q_x^3 + N_4 q_x^4 + N_5 q_x^5 + N_6 q_x^6 \ .$$

Since the displacements $q_x^i$ are constant, we can write:

$$\varepsilon_x = \frac{\partial u}{\partial x} = \frac{\partial N_1}{\partial x} q_x^1 + \frac{\partial N_2}{\partial x} q_x^2 + \frac{\partial N_3}{\partial x} q_x^3 + \frac{\partial N_4}{\partial x} q_x^4 + \frac{\partial N_5}{\partial x} q_x^5 + \frac{\partial N_6}{\partial x} q_x^6 \ ,$$

In the matrix notation this can be written as:

*Eq. 5.35*
$$\varepsilon_x = \left[\frac{\partial N_1}{\partial x} \ \frac{\partial N_2}{\partial x} \ \frac{\partial N_3}{\partial x} \ \frac{\partial N_4}{\partial x} \ \frac{\partial N_5}{\partial x} \ \frac{\partial N_6}{\partial x}\right] [q_x^1, q_x^2, q_x^3, q_x^4, q_x^5, q_x^6]^T \ ,$$

which is equivalent to:

*Eq. 5.36*

$$\varepsilon_x = \left[\frac{\partial N_1}{\partial x}\ 0\ 0\ \frac{\partial N_2}{\partial x}\ 0\ 0\ \frac{\partial N_3}{\partial x}\ 0\ 0\ \frac{\partial N_4}{\partial x}\ 0\ 0\ \frac{\partial N_5}{\partial x}\ 0\ 0\ \frac{\partial N_6}{\partial x}\ 0\ 0\right][q]^T.$$

The above is the equation for the first row of the matrix $[B]$. Using the same approach for the other 5 components of the strain, we can obtain similar formulas for the other 5 rows. The matrix $[B]$ is then defined as:

*Eq. 5.37*

$$[B] = \begin{bmatrix} \frac{\partial N_1}{\partial x} & 0 & 0 & \frac{\partial N_2}{\partial x} & 0 & 0 & \frac{\partial N_3}{\partial x} & 0 & 0 & \frac{\partial N_4}{\partial x} & 0 & 0 & \frac{\partial N_5}{\partial x} & 0 & 0 & \frac{\partial N_6}{\partial x} & 0 & 0 \\ 0 & \frac{\partial N_1}{\partial y} & 0 & 0 & \frac{\partial N_2}{\partial y} & 0 & 0 & \frac{\partial N_3}{\partial y} & 0 & 0 & \frac{\partial N_4}{\partial y} & 0 & 0 & \frac{\partial N_5}{\partial y} & 0 & 0 & \frac{\partial N_6}{\partial y} & 0 \\ 0 & 0 & \frac{\partial N_1}{\partial z} & 0 & 0 & \frac{\partial N_2}{\partial z} & 0 & 0 & \frac{\partial N_3}{\partial z} & 0 & 0 & \frac{\partial N_4}{\partial z} & 0 & 0 & \frac{\partial N_5}{\partial z} & 0 & 0 & \frac{\partial N_6}{\partial z} \\ 0 & \frac{\partial N_1}{\partial z} & \frac{\partial N_1}{\partial y} & 0 & \frac{\partial N_2}{\partial z} & \frac{\partial N_2}{\partial y} & 0 & \frac{\partial N_3}{\partial z} & \frac{\partial N_3}{\partial y} & 0 & \frac{\partial N_4}{\partial z} & \frac{\partial N_4}{\partial y} & 0 & \frac{\partial N_5}{\partial z} & \frac{\partial N_5}{\partial y} & 0 & \frac{\partial N_6}{\partial z} & \frac{\partial N_6}{\partial y} \\ \frac{\partial N_1}{\partial z} & 0 & \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial z} & 0 & \frac{\partial N_2}{\partial x} & \frac{\partial N_3}{\partial z} & 0 & \frac{\partial N_3}{\partial x} & \frac{\partial N_4}{\partial z} & 0 & \frac{\partial N_4}{\partial x} & \frac{\partial N_5}{\partial z} & 0 & \frac{\partial N_5}{\partial x} & \frac{\partial N_6}{\partial z} & 0 & \frac{\partial N_6}{\partial x} \\ \frac{\partial N_1}{\partial y} & \frac{\partial N_1}{\partial x} & 0 & \frac{\partial N_2}{\partial y} & \frac{\partial N_2}{\partial x} & 0 & \frac{\partial N_3}{\partial y} & \frac{\partial N_3}{\partial x} & 0 & \frac{\partial N_4}{\partial y} & \frac{\partial N_4}{\partial x} & 0 & \frac{\partial N_5}{\partial y} & \frac{\partial N_5}{\partial x} & 0 & \frac{\partial N_6}{\partial y} & \frac{\partial N_6}{\partial x} & 0 \end{bmatrix}.$$

The above definition of $[B(\xi, \eta, \zeta)]$ includes shape function derivatives with respect to Cartesian coordinates. I define a matrix $[N_{xyz}(\xi, \eta, \zeta)]$ containing the first order derivatives of shape functions with respect to the Cartesian coordinates:

*Eq. 5.38*

$$[N_{xyz}(\xi, \eta, \zeta)] = \begin{bmatrix} \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial x} & \frac{\partial N_3}{\partial x} & \frac{\partial N_4}{\partial x} & \frac{\partial N_5}{\partial x} & \frac{\partial N_6}{\partial x} \\ \frac{\partial N_1}{\partial y} & \frac{\partial N_2}{\partial y} & \frac{\partial N_3}{\partial y} & \frac{\partial N_4}{\partial y} & \frac{\partial N_5}{\partial y} & \frac{\partial N_6}{\partial y} \\ \frac{\partial N_1}{\partial z} & \frac{\partial N_2}{\partial z} & \frac{\partial N_3}{\partial z} & \frac{\partial N_4}{\partial z} & \frac{\partial N_5}{\partial z} & \frac{\partial N_6}{\partial z} \end{bmatrix}.$$

Since the shape functions have been defined in isoparametric coordinates, the inverse of the Jacobian matrix is used to change between the coordinate systems:

*Eq. 5.39*

$$[N_{xyz}(\xi, \eta, \zeta)] = [J(\xi, \eta, \zeta)]^{-1} \begin{bmatrix} \frac{\partial N_1}{\partial \varepsilon} & \frac{\partial N_2}{\partial \varepsilon} & \frac{\partial N_3}{\partial \varepsilon} & \frac{\partial N_4}{\partial \varepsilon} & \frac{\partial N_5}{\partial \varepsilon} & \frac{\partial N_6}{\partial \varepsilon} \\ \frac{\partial N_1}{\partial \eta} & \frac{\partial N_2}{\partial \eta} & \frac{\partial N_3}{\partial \eta} & \frac{\partial N_4}{\partial \eta} & \frac{\partial N_5}{\partial \eta} & \frac{\partial N_6}{\partial \eta} \\ \frac{\partial N_1}{\partial \xi} & \frac{\partial N_2}{\partial \xi} & \frac{\partial N_3}{\partial \xi} & \frac{\partial N_4}{\partial \xi} & \frac{\partial N_5}{\partial \xi} & \frac{\partial N_6}{\partial \xi} \end{bmatrix} ,$$

where the derivatives are evaluated at a point $(\xi, \eta, \zeta)$. Using Eq. 5.23 the above can be written as:

*Eq. 5.40*

$$[N_{xyz}(\xi, \eta, \zeta)] = [J(\xi, \eta, \zeta)]^{-1} [N_{\varepsilon\eta\xi}(\xi, \eta, \zeta)]$$

Once the entries of $[N_{xyz}]$ are computed, they are used to construct matrix $[B]$.

**Gaussian integration**

In order to avoid symbolic calculation of the integral given by the formula in Eq. 5.26, I calculate it by using the Gaussian integration technique, which is very commonly used technique in FEM applications. Gaussian integration evaluates the inside of an integral only at a few, carefully chosen points. It can be described as approximating an integral with a weighted sum:

*Eq. 5.41*

$$\int_a^b f(x)dx \approx \sum_{j=1}^{n} w_j f(x_j) ,$$

where $n$ represents the number of Gaussian integration points used, and $j$ represents the integration point. The precision of this technique depends on the nature of function $f$ as well as the number of integration points chosen. When the Gaussian integration is applied to Eq. 5.26, we obtain:

Eq. 5.42
$$[K_E] = \sum_{i=1}^{n} w_j [B(\xi_j, \eta_j, \zeta_j)]^T [D][B(\xi_j, \eta_j, \zeta_j)] \left| det([J(\xi_j, \eta_j, \zeta_j)]) \right| .$$

To obtain the Gaussian points and weights for the 6-node wedge element, I combined the 3-point Gaussian quadrature for a line segment (representing the height of the wedge) with a 3-point Gaussian quadrature for a triangle in a standard manner [74]. The result is a 9 point Gaussian quadrature specified by Table 5.1.

| $i$ | $w_i$ | $\xi_i$ | $\eta_i$ | $\zeta_i$ |
|---|---|---|---|---|
| 1 | 0.18518518519 | 0.66666666667 | 0.16666666667 | 0.77459666924 |
| 2 | 0.18518518519 | 0.16666666667 | 0.66666666667 | 0.77459666924 |
| 3 | 0.18518518519 | 0.16666666667 | 0.16666666667 | 0.77459666924 |
| 4 | 0.2962962963 | 0.66666666667 | 0.16666666667 | 0 |
| 5 | 0.2962962963 | 0.16666666667 | 0.66666666667 | 0 |
| 6 | 0.2962962963 | 0.16666666667 | 0.16666666667 | 0 |
| 7 | 0.18518518519 | 0.66666666667 | 0.16666666667 | -0.77459666924 |
| 8 | 0.18518518519 | 0.16666666667 | 0.66666666667 | -0.77459666924 |
| 9 | 0.18518518519 | 0.16666666667 | 0.16666666667 | -0.77459666924 |

**TABLE 5.1:** *Gaussian quadrature points for the 6-node wedge.*

**Modeling elemental pre-strain**

Eq. 5.2 relates the nodal displacements of an element to the forces it exerts on the nodes, which is standard for most FEM problems. However, in the following section it will be more useful to reformulate this equation in terms of nodal coordinates, rather than in terms of their displacements. Assume that the nodal coordinates of a node $i$ are given by a vector $(p_x^i, p_y^i, p_z^i)$. The nodal displacement $(q_x^i, q_y^i, q_z^i)$ of the node $i$ is related to the nodal coordinates by:

*Eq. 5.43*
$$(q_x^i, q_y^i, q_z^i) = (p_x^i, p_y^i, p_z^i) - (x_i, y_i, z_i),$$

where $(x_i, y_i, z_i)$ is the nodal coordinate of the element in its undeformed shape. For the entire element, this is captured by:

*Eq. 5.44*
$$[q] = [P] - [Q],$$

where $[P] = \{p_x^1, p_y^1, p_z^1, ..., p_x^6, p_y^6, p_z^6\}$. Substituting Eq. 5.44 into Eq. 5.2, we obtain:

*Eq. 5.45*
$$[F] = [K_e]([P] - [Q]) = [K_e][P] - [K_e][Q].$$

Since $[Q]$ is constant, I define a matrix $[R_e]$ as:

*Eq. 5.46*
$$[R_e] = [K_e][Q],$$

and rewrite Eq. 5.45 as:

*Eq. 5.47*
$$[F] = [K_e][P] - [R_e].$$

The above is a linear relationship between deformed nodal coordinates $[P]$ and the resulting nodal forces $[F]$ exerted by an element. Eq. 5.47 is a more general form of Eq.

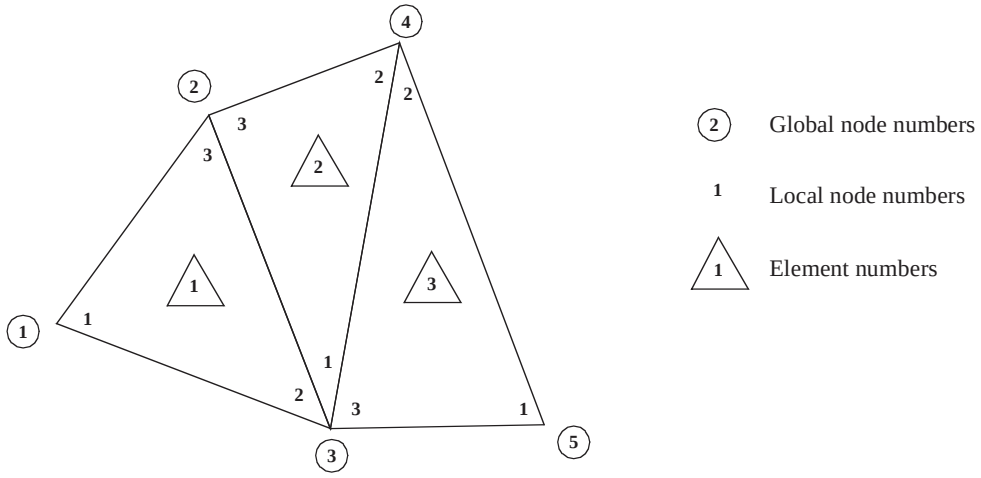



**FIGURE 5.7:** *Global and local node numbering.*

5.2, by including the quantity $[R_e]$, which is analogous to the definition of elemental pre-stress [74].

## 5.4. The global stiffness matrix

The previous section was devoted to deriving a set of linear equations relating nodal coordinates to the nodal forces an element generates on its nodes. In this section, I show the standard derivation of a similar equation for the entire system of elements. Such an equation relates the coordinates of all nodes in the system to the total force exerted by the system on the nodes.

I start by considering the total force acting on a node $i$. Such force is defined as the sum of the forces exerted on the node by all elements sharing the node. As an example, consider global node 3 in the system of triangular elements illustrated in Figure 5.7. The total force acting on global node 3 is the sum of the forces exerted on it by elements 1 and 2. The

contribution from element 1 can be obtained from the element's stiffness matrix $[F_e]$ by extracting the values $F_x^2, F_y^2, F_z^2$. Similarly, the contribution from element 2 is obtained by extracting the values $F_x^1, F_y^1, F_z^1$ from the second element's stiffness matrix $[F_e]$. The force exerted by element 1 is a linear combination of the deformed nodal coordinates of global nodes 1, 2 and 3 and undeformed nodal coordinates of local nodes 1, 2 and 3. The contribution from element 2 is a linear combination of the deformed nodal coordinates of the global nodes 2, 3 and 4, and the undeformed nodal coordinates of local nodes 1, 2 and 3. The overall result is that the total force acting on node 2 is a linear combination of the deformed nodal coordinates of global nodes 1, 2, 3 and 4 and the undeformed nodal coordinates of the local nodes for elements 1 and 2.

In general, the global equation relating the deformed nodal coordinates of the system of elements to the total nodal forces exerted by the system is of the form:

*Eq. 5.48*

$$[F_G] = [K_G][P_G] + [R_G],$$

where the column vector $[F_G]$ represents the nodal forces in the system:

*Eq. 5.49*

$$[F_G] = [F_x^1, F_y^1, F_z^1, F_x^2, F_y^2, F_z^2, ..., F_x^n, F_y^n, F_z^n].$$

The triples $F_x^i, F_y^i, F_z^i$ denote the nodal force vector at node $i$. For a system with $n$ nodes, $[F_G]$ is a column vector of size $3n$. The column vector $[P_G]$ contains the nodal coordinates of every node in the system and it is also of size $3n$:

*Eq. 5.50*

$$[P_G] = [P_x^1, P_y^1, P_z^1, ..., P_x^n, P_y^n, P_z^n],$$

where $p_x^i, p_y^i, p_z^i$ is the nodal coordinate of a global node $i$. The matrix $[K_G]$ is the global stiffness matrix, and its dimensions are $3n$ by $3n$. It is computed by combining the elemental stiffness matrices $[K_e]$. Finally, $[R_G]$ is a column vector of size 18x1, computed by combining the elemental pre-stress matrices $[R_e]$.

```
procedure calculate_Kg_and_Rg()
        allocate 3n by 3n matrix Kg and set its coefficients to 0
        allocate column vector Rg of size 3n and set its coefficients to 0
        for each element e do:
                add_Ke_to_Kg_and_Re_to_Rg( e, Kg, Rg)

procedure add_Ke_to_Kg_and_Re_to_Rg( e, Kg, Rg):
        for r = 1 to 6 do:
                for c = 1 to 6 do:
                        rg = local_to_global( r, e)
                        cg = local_to_global( g, e)
                        for i = 1 to 3 do:
                                for j = 1 to 3 do:
                                        Kg[ rg*3+i, cg*3+j] += e.Ke[r*3+i, c*3+j]
        for r = 1 to 6 do:
                rg = local_to_global( r, e)
                for i = 1 to 3 do:
                        Rg[rg*3+i] += e.Re[r*3+i]

function local_to_global( k, e)
        // given a local node number k (in the range 1..6), this function returns
        // the global number of the k-th node in element e
```

**ALGORITHM 4:** *Algorithm for calculating global stiffness matrix.*

The global stiffness matrix $[K_G]$ is calculated directly from the elemental stiffness matrices, by summing up the appropriate entries. Similarly, $[R_G]$ is calculated from each element's $[R_e]$. The algorithm for calculating $[K_G]$ and $[R_G]$ is provided in Algorithm 4. Eq. 5.48 can be used to calculate the total forces acting on each node in the system for a given set of deformed nodal coordinates $[P_G]$. It can also be used to calculate the nodal coordinates to achieve a certain force distribution, i.e. it can calculate the deformation of the model under applied loads. Finally, it allows the calculation of an equilibrium state ($[F_G] = 0$) when a deformation is introduced into the system. The deformation is modeled by fixing some of the nodal coordinates to specific values, e.g. when modeling growth the deformation is due to the bottom nodes being fixed to the growing background layer.

## 5.5. Calculating the equilibrium state

The equilibrium state is a configuration of the nodal positions in which the total nodal forces acting on all free nodes vanish. The free nodes are all the top nodes of the elements, and the fixed nodes are the bottom nodes which are attached to the background layer. The fixed nodes are the sources of initial strain when modeling growth of the background layer. Calculating an equilibrium state is equivalent to finding the nodal displacements for the free nodes to satisfy the requirement of an equilibrium state, given some initial strain. The equilibrium of the system of elements is calculated in three steps:

- inserting initial strain into Eq. 5.48 (to account for growth);
- inserting boundary conditions into the equations (setting the desired nodal forces at free nodes to 0);
- solving the equations (to find the unknown nodal coordinates of the free nodes).

The first step is to include the state of the initial strain into Eq. 5.48. To this end, the diagonal entry $(i, i)$ in $[K_G]$ is set to 1 for each known nodal coordinate value $p_G^i$, and the rest of the row $i$ is set to 0's. Further, the entry in row $i$ in $[F_G]$ is set to the value of the known nodal coordinate $w_G^i$, and the entry in row $i$ of $[R_G]$ is set to zero. This method of modeling initial strain is standard in applications of FEM [74].

The next step in calculating the equilibrium state is to insert boundary conditions into the equation. Boundary conditions reflect the fact that in the equilibrium state there should be zero net forces acting on the free nodes. To this effect all remaining rows of $[F_G]$ are set to zero.

There is an alternative method for modifying the set of global equations to include the known nodal coordinates of the fixed nodes. The known variables can be completely removed from the system (i.e. by removing the corresponding rows and columns of $[K_G]$,

$[R_G]$ and $[F_G]$), by substituting their known values into the equations of the free nodes. To illustrate the difference between these two approaches, consider the following set of 3 equations of 3 unknowns:

$$3x + 4y + z = 5$$
$$2x + y - 2z = 3$$
$$x + 2y + 2z = 1$$

The first approach for modifying these equations to include some known value, e.g. $z = 2$, would result in a set of equations:

$$3x + 4y + z = 5$$
$$2x + y - 2z = 3$$
$$z = 2$$

The second approach would modify the system by eliminating the references to the variable $z$ altogether, i.e.:

$$3x + 4y = 3$$
$$2x + y = 7$$

Since one half of the nodes are fixed, this second approach results in reducing the system of equations by 50%. Such a reduction in the number and size of equations leads to an improved performance when solving for the unknowns, although the time consumed by performing such adjustments (calculations + memory management) offsets some of these gains. I have implemented and use both approaches in my simulations. I use the first approach to recalculate the equilibrium state of the entire model, and the second approach to recalculate the equilibrium state of a sub-model.

After all known quantities are inserted into Eq. 5.48, the unknown nodal coordinates must be calculated. The resulting equation can be written in the more familiar form for describing systems of linear equations:

*Eq. 5.51* $$[A][x] = [b] \ ,$$

where $[A]$ is the coefficient matrix, equal to $[K_G]$, $[x]$ is the column vector of the unknowns, equal to $[w_G]$ and $[b]$ is the right hand side of the equation, equal to $[F_G] - [R_G]$.

Many different techniques exist for solving a set of linear equations. They can be divided into two main categories: direct and iterative methods. Direct methods solve the equations by algebraic manipulations, while iterative methods solve the equations by improving an existing solution in successive iterations. Iterative solutions do not guarantee an exact solution, although they achieve a requested precision. I implemented a number of techniques for solving a set of linear equations, which I describe next.

**Gaussian elimination**

Gaussian elimination falls into the category of direct solution techniques for solving sets of linear equations. The general idea is to eliminate the first unknown from all equations (except the first) by subtracting the corresponding multiple of the first equation from all other equations. Then the second unknown is removed in a similar manner from all equations except the first two. This process is repeated until the last equation contains only the last unknown, which provides the solution for the last unknown. This solution is then back-substituted into the second last equation to obtain the solution to the second last unknown. The back-substitution is repeated until solutions to all unknowns are retrieved. Clearly, an algorithm implementing Gaussian elimination will run in $O(n^3)$ time, where

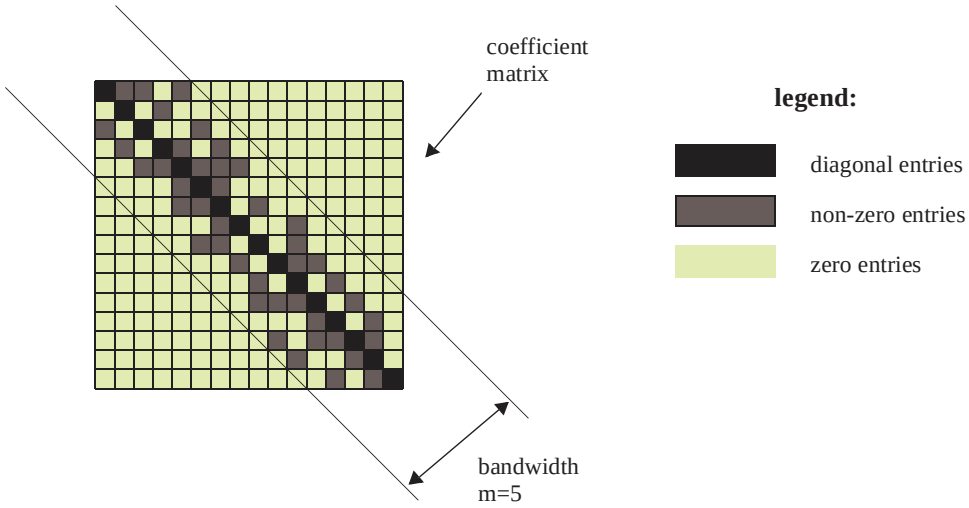



**FIGURE 5.8:** *Graphical representation of matrix bandwidth.*

$n$ is the number of equations in the system. For small $n$, the algorithm's performance is satisfactory. However, as $n$ grows into the thousands, the running time becomes unacceptable. Since there are often tens of thousands of nodes in the models I simulate, Gaussian elimination is inadequate.

**Gaussian elimination on banded systems**

The Gaussian elimination algorithm can be made more effective when the coefficient matrix of the linear system solved is banded. In a banded matrix all nonzero components tend to group around the diagonal elements. Bandwidth of a matrix is a number describing how well the given matrix is banded. It corresponds to the width of a diagonal band (or strip) which completely encompasses all non-zero elements of a matrix (see Figure 5.8 for a graphical interpretation of bandwidth). A matrix containing non-zero entries only on its diagonal would have a bandwidth equal to 1, while a matrix without any non-zero entries would have a bandwidth equal to its size.

If the bandwidth of a given matrix is $m$, then the Gaussian elimination algorithm can be modified to execute in the worst running time of $O(m^2 n)$. This is achieved by modifying a regular Gaussian elimination algorithm to perform row subtractions only in the areas where there are non-zero entries, i.e. around the main diagonal of the coefficient matrix. Since the efficiency of the algorithm is directly proportional to the bandwidth of the coefficient matrix, a significant improvement over the standard Gaussian elimination method will be seen only if $m \ll n$.

In general, the global stiffness matrix $[K_G]$ is very sparse. In a given row $i$ it contains a non-zero entry in column $j$ only if the nodes associated with degrees of freedom $i$ and $j$ are connected by an edge in the geometrical model. Two nodes are connected by an edge only if they both belong to the same element, or if they belong to two elements sharing either of the two nodes. Therefore, the bandwidth of $[K_G]$ depends on how the numbers are assigned to the nodes. To achieve optimum bandwidth, the numbering should be realized in a manner which minimizes the maximum difference between the numbers assigned to any two edge-connected nodes. Although algorithms accomplishing such node re-numbering can be found in literature on graph theory, they are slow and quite complex to implement. More importantly, even if they achieve optimum node re-numbering, the bandwidth is still too large for the total running time of the algorithm to be acceptable for large numbers of nodes.

**Gauss elimination on sparse matrices**

The models I use often have tens of thousands of elements and nodes. The size of a full coefficient matrix $[K_G]$ for $n = 10000$ nodes is $sizeof(double)(3n)^2$ bytes, or approximately 6.7 gigabytes (assuming double precision arithmetic is used and that each number requires 8 bytes of memory). At present time, it would be very difficult to find a computer system which could accommodate such a memory intensive algorithm. The

banded matrix offers an improvement, but the memory requirement is still excessive. A significant improvement in the memory requirement is achieved by implementing sparse storage for $[K_G]$.

In my implementation, I use row-compressed sparse storage of $[K_G]$. In row-compressed storage, each row of the matrix is stored as a list of columns containing non-zero entries. The entries with 0's are not stored. Given a semi-random arrangement of nodes and the resulting Delaunay triangulation, a node in the system is connected on average to 14 other nodes. This means that the global stiffness matrix $[K_G]$ will have on average only $42$ non-zero entries in each row. After insertion of the fixed nodal coordinates, this number further reduces to about $21$. As a result, storing the $K_G$ in the row-compressed format only requires approximately $21 \times 3n$ coefficients, which for $n$ = 10000 is 4.8 megabytes. Since the column number has to be stored with each coefficient, additional $21 \times 3n$ indices have to be stored. If each index is stored as a long, and each long is stored using 4 bytes, then the total memory requirement to store $[K_G]$ is approximately 7 megabytes. Clearly, the sparse matrix storage provides a significant improvement in the memory requirement.

Gaussian elimination can be properly modified to operate on such sparsely stored coefficient matrices, taking advantage of the fact that it only needs to perform row subtractions at points where there are non-zero entries. Unfortunately, the row subtraction introduces other non-zero elements to the rows and in the end, both the memory requirement and the total running time are still $O(n^2)$.

**Gauss-Seidel iterative method**

The Gauss-Seidel method was the first iterative method I investigated for use in my simulations. I found that its running time was acceptable, but in many cases it failed to converge to a solution of desired accuracy.

**Conjugate gradient method**

The method of conjugate gradients is a very commonly used technique for solving large sparse systems of linear equations [52]. Its main attractiveness is due to the fact that it only uses the sparse coefficient matrix $[A]$ for multiplication by a column vector. This can be very efficiently implemented, for example, by storing $[A]$ in the row-compressed form as described above. Only $21 \times 3n$ multiplications and additions are needed on average, based on the estimations given above. Furthermore, according to the authors, "If A is positive definite as well as symmetric, the algorithm cannot break down (in theory!).". Since $[K_G]$ is both positive definite and symmetric, I selected the conjugate-gradient over the other method described, and implemented the ordinary conjugate gradient algorithm as described in [52].

## 5.6. Fracture modeling by removing elements

Cracks form in solid materials as a result of material fatigue [1]. When a material is under load, the stresses inside the material slowly increase, until they reach a critical value - the *threshold stress* (similar to threshold stress as defined for example in [1] or [16]). At the point where the threshold stress is exceeded, an initial fracture is introduced, forming a permanent discontinuity in the material. As a consequence of this new discontinuity in the

material, stresses shift to other regions, and usually accumulate in the vicinity of the existing crack tips. This leads to crack propagation.

I examined two different approaches for modeling fractures. In this section I describe the first approach, which models fractures by removing elements in which the stress has exceeded the material's threshold stress. The second approach, based on splitting elements, will be described in Section 5.7.

### 5.6.1. Stress calculation

The first step in determining whether a fracture will be formed is to calculate the current state of stress in the material. The state of stress in solid mechanics is given by a stress tensor [16], defined as a symmetric 3x3 matrix:

*Eq. 5.52*

$$[\sigma]_{3x3} = \begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{zx} \\ \tau_{xy} & \sigma_y & \tau_{yz} \\ \tau_{zx} & \tau_{yz} & \sigma_z \end{bmatrix} .$$

In most FEM literature, an alternative representation for the stress tensor is used, defining it as a column vector containing only the unique entries from Eq. 5.52, i.e. $[\sigma] = [\sigma_x, \sigma_y, \sigma_z, \tau_{xy}, \tau_{yz}, \tau_{zx}]^T$. The stress tensor is related to the strain tensor $[\varepsilon] = [\varepsilon_x, \varepsilon_y, \varepsilon_z, \gamma_{xy}, \gamma_{yz}, \gamma_{zx}]^T$ by Eq. 5.31. The strain tensor varies throughout the material, depending on the local deformation. To calculate the strain tensor at a specific isoparametric coordinate inside an element, I use the element's matrix $[B]$ evaluated at the appropriate isoparametric coordinates, and then substitute the result into Eq. 5.33. The result is a formula for calculating the stress tensor at a given coordinate inside an element:

*Eq. 5.53*

$$[\sigma(\xi, \eta, \zeta)] = [D][B(\xi, \eta, \zeta)][q] .$$

**Selecting locations for stress evaluation**

Since the stress tensor varies throughout the element, a coordinate at which it will be evaluated must be chosen. For example, the geometric center of the element could be used, with isoparametric coordinates $(1/3, 1/3, 0)$. However, since numerical integration was used to evaluate $[K_E]$, which in turn was used to calculate the equilibrium state, the values of computed stress will not necessarily be exact at all coordinates. The highest accuracy of the calculated stresses will be attained at the Gaussian points, which were used during the numerical integration step. Therefore, evaluating the stress at one of the Gaussian points is a better choice. Since cracks observed on expanding surfaces typically originate in the top layer and then propagate deeper into the material, one of the Gaussian points near the top face of an element could be used. And since the state of stress also varies at Gaussian points, I evaluate the stresses at all three Gaussian points closest to the top face.

**Principal stresses and principal planes of stress**

The stress tensor defines the state of stress in the material, which can be different in every direction. Positive values indicate tensile stresses and negative values indicate compressive stresses. In some directions, the stress value will be smaller, in others it will be greater. A stress tensor will yield up to 3 different local minimum/maximum values for stress, called *principal stresses*. The planes whose normals are defined by the directions corresponding to principal stresses are called *principal stress planes.* Principal stresses are found by calculating eigenvalues of the stress tensor matrix, while the corresponding eigenvectors correspond to the normals of the principal stress planes [16]. The algorithms for calculating both eigenvectors and eigenvalues can be found in [52].

To determine whether the threshold stress has been exceeded in an element, the principal stresses of the three stress tensors are first calculated. The resulting 9 values are sorted, and the largest one is compared against the material's threshold stress. If the largest eigenvalue exceeds the threshold stress, a fracture will be formed.

Stresses in the material layer increase due to applied growth and/or shrinkage. Before any fractures are formed, stresses in the material layer tend to achieve the highest values near the center. Once a fracture is formed, stresses are highest near crack tips, and at the centre of un-cracked material patches. An illustration of the distribution of stresses is shown in Figure 5.9, where the values of maximum principal stresses are used to color the elements.

### 5.6.2. Selecting the element for removal and adaptive time-step control

Once the equilibrium state has been obtained and the maximum principal stresses in each element have been calculated, each element is checked to determine whether it exceeded the material's threshold stress. If a single element has exceeded the material's threshold stress, a fracture could be modeled by simply removing this element from the model. Unfortunately, the background may grow and/or the material layer may shrink in large enough increments for the threshold stress to be exceeded in more than one element. At first glance, this does not seem to pose any difficulties, since multiple fractures can develop simultaneously in real materials. A simple approach would then be to remove all elements which exceeded the threshold stress. This, however, would lead to the removal of large portions of the material layer, as the threshold stresses are often exceeded simultaneously in the same neighborhoods.

To prevent the threshold stress from being exceeded in multiple elements, the time step could be chosen to be sufficiently small before the simulation begins. This simple

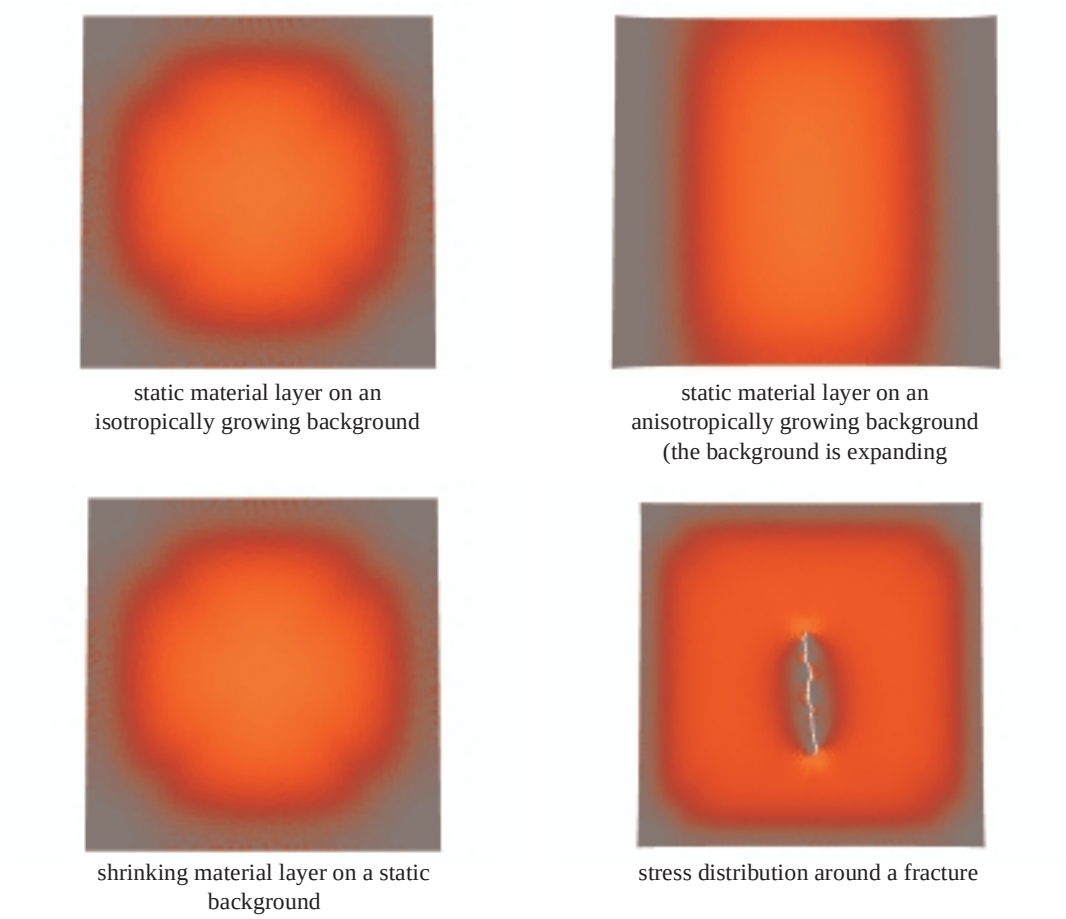



**FIGURE 5.9:** *Distribution of the maximum principal stress.*

solution, however, leads to large increases in the total running time needed to complete the simulation. The major obstacle lies in the fact that the time step would have to be chosen at the beginning of the simulation, when it would be difficult to predict how small it should be. If, in the middle of the simulation, it was discovered that the time step was too large, the simulation would have to be restarted from the beginning with a smaller time step. Furthermore, smaller time step would require more frequent applications of the growth and relaxation steps, both of which are very time consuming procedures.

A different approach is to keep the time step reasonable, but choose only one of the elements whose internal stress has exceeded the materials threshold stress. In my early implementations, I experimented by choosing the element with the highest ratio of stress to threshold stress. This choice was based on the assumption that this element was most likely the first to have exceeded the threshold stress. Although this method produced satisfactory results, it is based on an assumption which I could not prove to be a fact.

The approach I took in the current implementation uses an adaptive time step control, which I designed to quickly advance through time when no fractures develop. When time is advanced in the simulation, nothing 'interesting' happens until a new fracture is developed. The goal of the adaptive time step is to find the closest point in time when a new fracture is formed. This is achieved as a two step process. In the first step, the lower and upper bound on time is found, by advancing the time in successively larger time increments. This is repeated until a point in time is found where there is at least one fracture candidate. In the next step, a binary search is applied to find the exact time. The pseudo-code capturing this process is given in Algorithm 5.

Unfortunately, there is no guarantee that a time step can be made small enough to avoid threshold stress from being exceeded in multiple locations. After all, it is possible that the threshold stress in real life materials is exceeded simultaneously. Therefore, if the time step cannot be further reduced and there are more than one fracture candidates, the candidate with the highest stress to threshold stress ratio is selected for the next fracture. In case of a tie, one of the candidates is selected randomly.

```
procedure advance_simulation_time( t)
        t_min = current time
        dt = smallest time step allowed
        repeat:
                t_max = t_min + dt
                apply growth corresponding to time t_max
                relax the model and recalculate the stresses
                nf = number of fracture candidates
                if nf > 0 then
                        break // we have the lower and upper bounds in t_min and t_max
        repeat
                if t_max - t_min < smallest time step allowed then
                        break
                t = (t_max + t_min) / 2
                apply growth corresponding to time t
                relax the model and recalculate the stresses
                nf = number of fracture candidates
                if nf == 0 then
                        t_min = t
                else
                        t_max = t
        advance the current simulation to t_max
```

**ALGORITHM 5:** *Adaptive time step control.*

### 5.6.3. Element removal

Once the element has been chosen for deletion, it has to be removed from the model. The removal process consists of two parts. First, the geometrical representation of the model has to be adjusted to reflect the removal of the element, i.e. the associated faces and edges must be eliminated. Second, the global stiffness matrix $[K_G]$ and $[R_G]$ must be adjusted to reflect the missing element. A simple, precise, but very inefficient approach, is to simply recalculate the entire $[K_G]$ and $[R_G]$ as outlined in Section 5.3. A more efficient

```
procedure sub_Ke_from_Kg_and_Re_from_Rg( e, Kg, Rg):
    for r = 1 to 6 do:
        for c = 1 to 6 do:
            rg = local_to_global( r, e)
            cg = local_to_global( g, e)
            for i = 1 to 3 do:
                for j = 1 to 3 do:
                    Kg[ rg*3+i, cg*3+j] -= e.Ke[r*3+i, c*3+j]
    for r = 1 to 6 do:
        rg = local_to_global( r, e)
        for i = 1 to 3 do:
            Rg[rg*3+i] -= e.Re[r*3+i]
```

**ALGORITHM 6:** *Removing* $[K_e]$ *and* $[R_e]$ *from* $[K_G]$ *and* $[R_G]$ *by subtraction.*

approach is to subtract the coefficients of the removed element's stiffness matrix $[K_e]$ from the global stiffness matrix and $[R_e]$ from $[R_G]$, as is outlined in Algorithm 6.

To prevent roundoff errors, which could result from frequent subtractions, I implemented a combination of these two methods. The result is an efficient but precise algorithm for recalculation of $[K_G]$. The removal of $[K_e]$ and $[R_e]$ from $[K_G]$ and $[R_G]$, respectively, is achieved by reconstructing the affected coefficients of $[K_G]$ and $[R_G]$ using the elemental stiffness matrices of the neighboring elements. An algorithm implementing this approach is given in Algorithm 7.

Once the element is removed from the model, the equilibrium state must be recomputed. The new equilibrium state will result in other elements exceeding threshold stresses, most likely in the areas of crack tips, leading to crack propagation. When no elements exceed the material's threshold stress, the simulation advances by increasing the simulation time and by applying growth to the background and/or by applying shrinkage to the elements. The propagation of fractures simulated using this method is illustrated in Figure 5.10.

```
procedure sub_Ke_from_Kg_and_Re_from_Rg( e, Kg, Rg):
        // zero out entries of Kg and Rg affected by this element
        for rn = 0 to 5 do:
                grn = local_to_global( rn, e)
                for cn = 0 to 5 do:
                        gcn = local_to_global( cn, e)
                        for i = 0 to 2 and j = 0 to 2 do:
                                Kg[grn*3+i,gcn*3+j] = 0
                for i = 0 to 2 do:
                        Rg[grn*3+i] = 0;

        // get a list of elements that share at least one node with e
        elist = get_a_list_of_elements_connected_to_element( e)

        // for each element in the list, add its appropriate contributions to
        // the erased parts of Kg and RHSg
        for every element f in elist do:
                for rn = 0 to 5 do:
                        grn = local_to_global( f, rn)
                        if global node grn is not in element e then
                                continue
                        for cn = 0 to 5 do:
                                gcn = local_to_global( f, cn)
                                if global node gcn is not in element e then
                                        continue
                                for i = 0 to 2 and j = 0 to 2 do:
                                        Kg[grn*3+i,gcn*3+j] += f-> Ke[rn*3+i,cn*3_j]
                        Rg[grn*3+0] += f-> Re[rn*3+0]
                        Rg[grn*3+1] += f-> Re[rn*3+1]
                        Rg[grn*3+2] += f-> Re[rn*3+2]
```

**ALGORITHM 7:** *Removing* $[K_e]$ *and* $[R_e]$ *from* $[K_G]$ *and* $[R_G]$ *by reconstruction.*

Figure 5.11 illustrates the difference between fracture patterns obtained for varying levels of surface discretization. When the number of elements approximating the surface is small, the fractures are wide and the overall pattern looks unrealistic. As the discretization level is increased, the visual appearance improves.

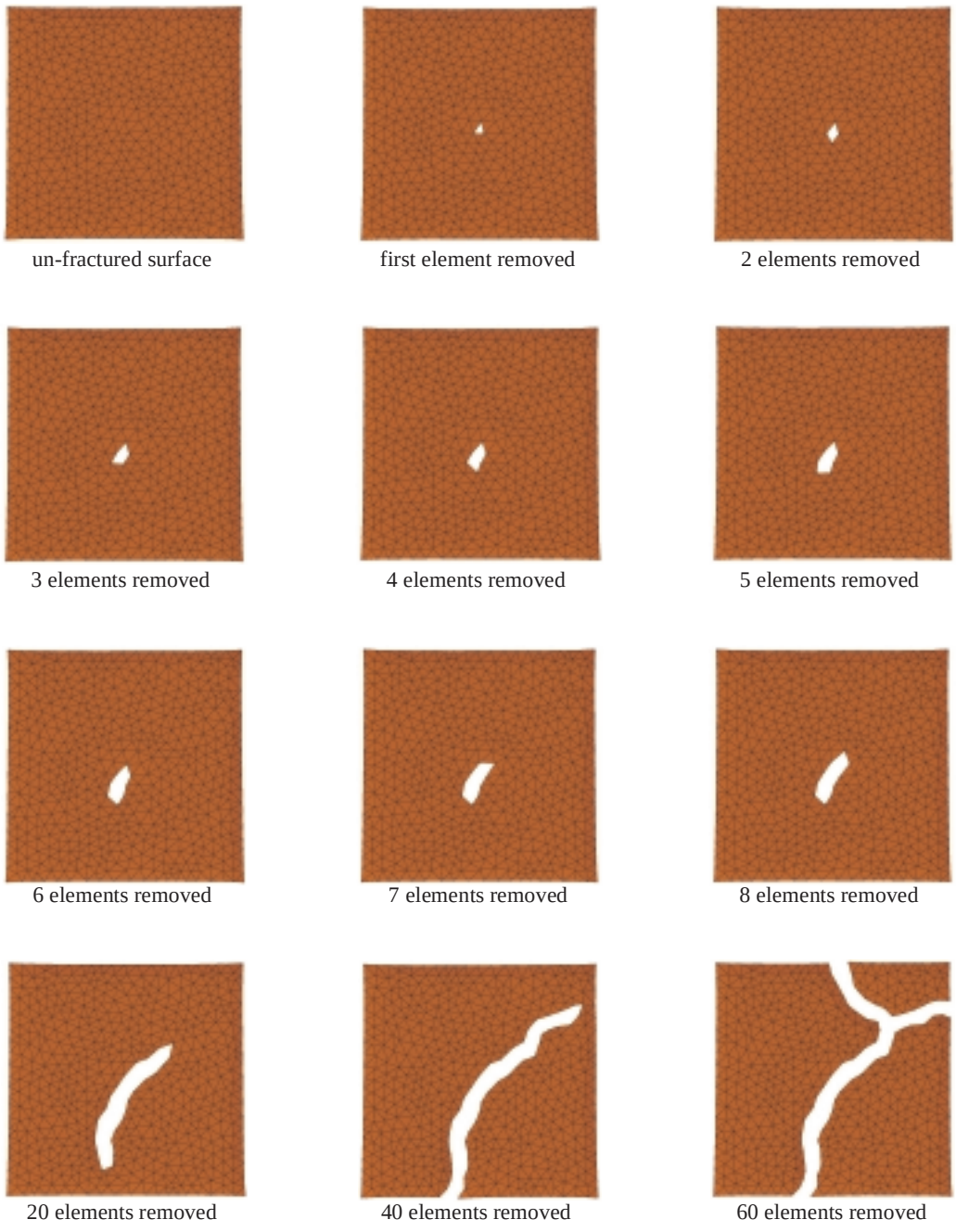



**FIGURE 5.10:** *Fracture propagation using element removal.*

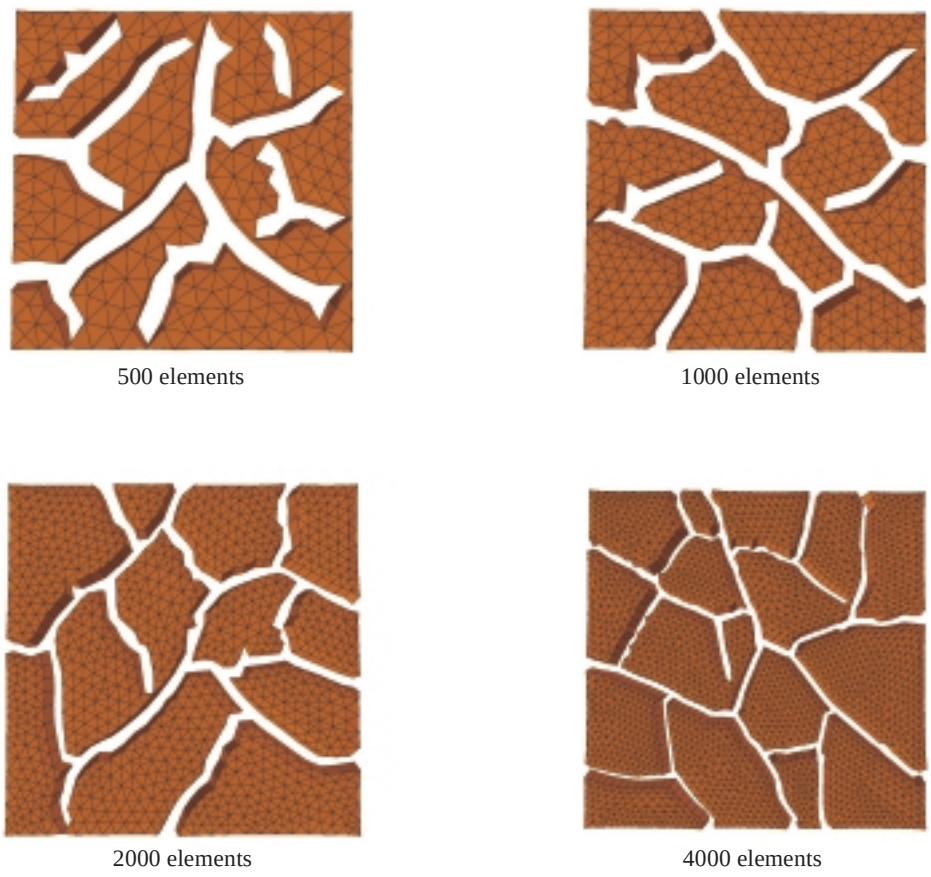



**FIGURE 5.11:** *Fracture patterns for different levels of discretization.*

### 5.6.4. Dynamic subdivision

Modeling fractures by removing elements has a visually unappealing side effect: parts of the material suddenly vanish from the model. The larger the elements, the larger the effect. In contrast, when fractures are formed in the real world, material does not magically disappear. To minimize this discrepancy between the simulation and the real world, elements should be small enough that their removal goes unnoticed by the naked eye. The

```
simulation_loop()
        while simulation not done do
                relax model
                if threshold stress exceeded then
                        identify element for removal
                        if element is small enough then
                                remove element
                        else
                                subdivide element
                else
                        increment simulation time
                        grow surface
```

**ALGORITHM 8:** *Dynamic subdivision.*

problem with this solution is that as the size of the elements decreases, their number must increase, which leads to larger memory requirements and longer simulations.

It is a waste of resources to represent the entire surface with small elements, as only a relatively small number of them are actually removed due to cracking. In areas where no fractures develop, large number of small elements would merely contribute increased computation time. It is therefore desirable to have these large areas covered by larger elements, and the small elements should only be present in areas where fractures are formed.

I implemented the above concept in the following algorithm. The surface is initially discretized by a small number of relatively large elements. An element identified for removal will be removed only if its size is smaller than some predefined threshold. If its size is larger than this threshold, it will be subdivided into smaller elements. The main simulation loop reflecting this dynamic subdivision approach is captured in the pseudocode in Algorithm 8.

To complete the description of the above algorithm, the element subdivision step must be explained. The subdivision of a single element into one or more elements is straightforward: the element is simply removed and replaced by a larger number of smaller elements. However, to preserve the connectivity of the mesh at nodes, the surrounding elements may also have to be adjusted. The problem of meshing, re-meshing and mesh subdivision is an important problem in FEM. I examined three different approaches, which will now be described.

**Subdividing an element by trisection**

Perhaps the simplest method for subdividing an element is to introduce two new nodes in the centers of the top and bottom faces and then to split the element into 3 new elements. This is illustrated in Figure 5.12. The only advantage of this approach is the simplicity of its implementation, as it does not require re-meshing of the neighboring elements. The drawback of this method lies in the rapid degeneration of the newly formed elements into narrow wedges, as illustrated in Figure 5.12. Since finite element methods do not work well with elements of deformed shapes, the subdivision method based on trisection is unusable.

**Subdividing an element into four elements**

The next approach I investigated introduces six new nodes on the edges of the element to be subdivided. Three of them are located on the top face and three on the bottom face. The locations of the new nodes are the midpoints of the top and bottom face edges, as shown in Figure 5.13 at the top. The new elements generated using this method of subdivision are

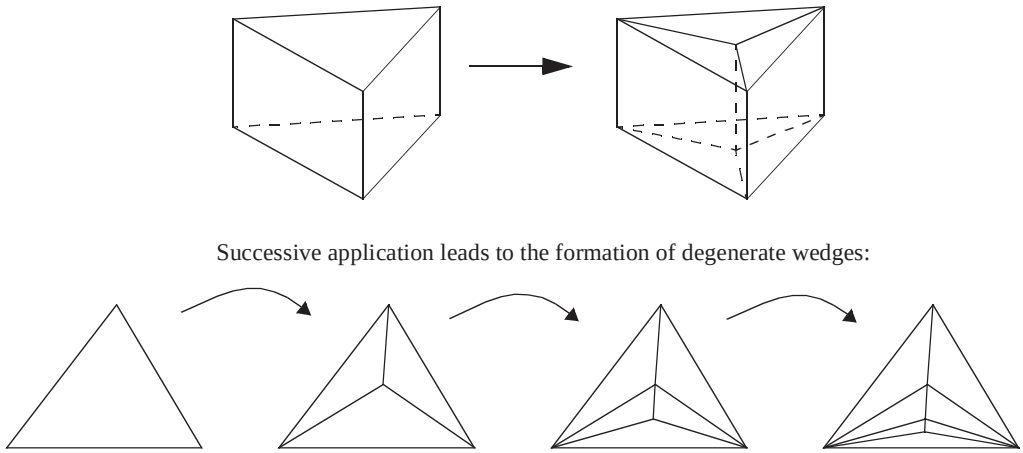



**FIGURE 5.12:** *Subdividing an element by trisection.*

guaranteed to have non-degenerate shapes, provided the original element is not itself degenerate.

Because new nodes are introduced at the side walls of the old element, the geometry of the neighboring elements also has to be adjusted. The neighboring elements are adjusted by a bisection. Even though the element being subdivided will be replaced by non-degenerate element, the same cannot be said about its neighbors. Figure 5.13 at the bottom illustrates how the mesh rapidly degenerates using this method.

**Subdividing an element by bisection**

Another approach for element subdivision is to divide the selected element into two. This is achieved by introducing two nodes in the middle of its longest top face edge and the corresponding bottom face edge, as illustrated at the top of Figure 5.14. The element on

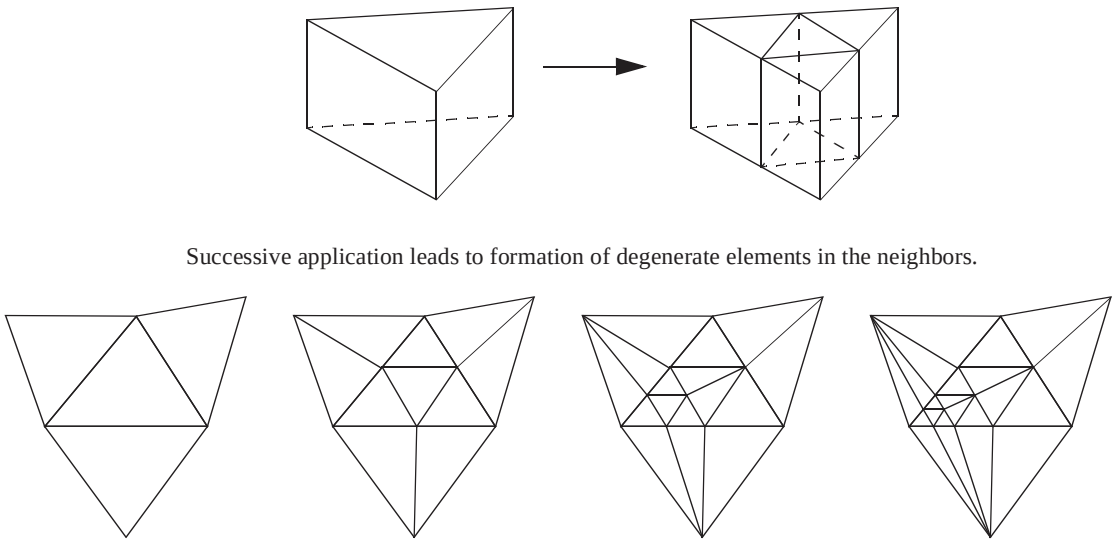



**FIGURE 5.13:** *Subdividing an element into four elements.*

the other side of the bisected edge, if present, must be re-meshed accordingly. This approach guarantees that the element being subdivided is replaced by nicely shaped elements, since only the longest edge is ever split. Unfortunately, this approach also leads to the formation of degenerate wedges in the neighboring elements, as illustrated in Figure 5.13 at the bottom. The degenerate triangles appear because the neighboring elements are not necessarily being split along their longest edge.

The solution to this problem was introduced by Rivara and Inostroza [58], who implemented the following algorithm. Given an element $e$ to be subdivided, split the element along its longest edge $g$. Find the other element $e_2$ sharing the edge $g$. If $g$ is not the longest edge of $e_2$, then recursively apply the subdivision process to $e_2$, until $g$ is its longest edge. Finally, $e_2$ is also split along $g$. This recursive procedure ensures that all elements are bisected only at their longest edges. This algorithm rarely reaches deep levels of recursion when applied to a triangular mesh with well shaped triangles (such as the

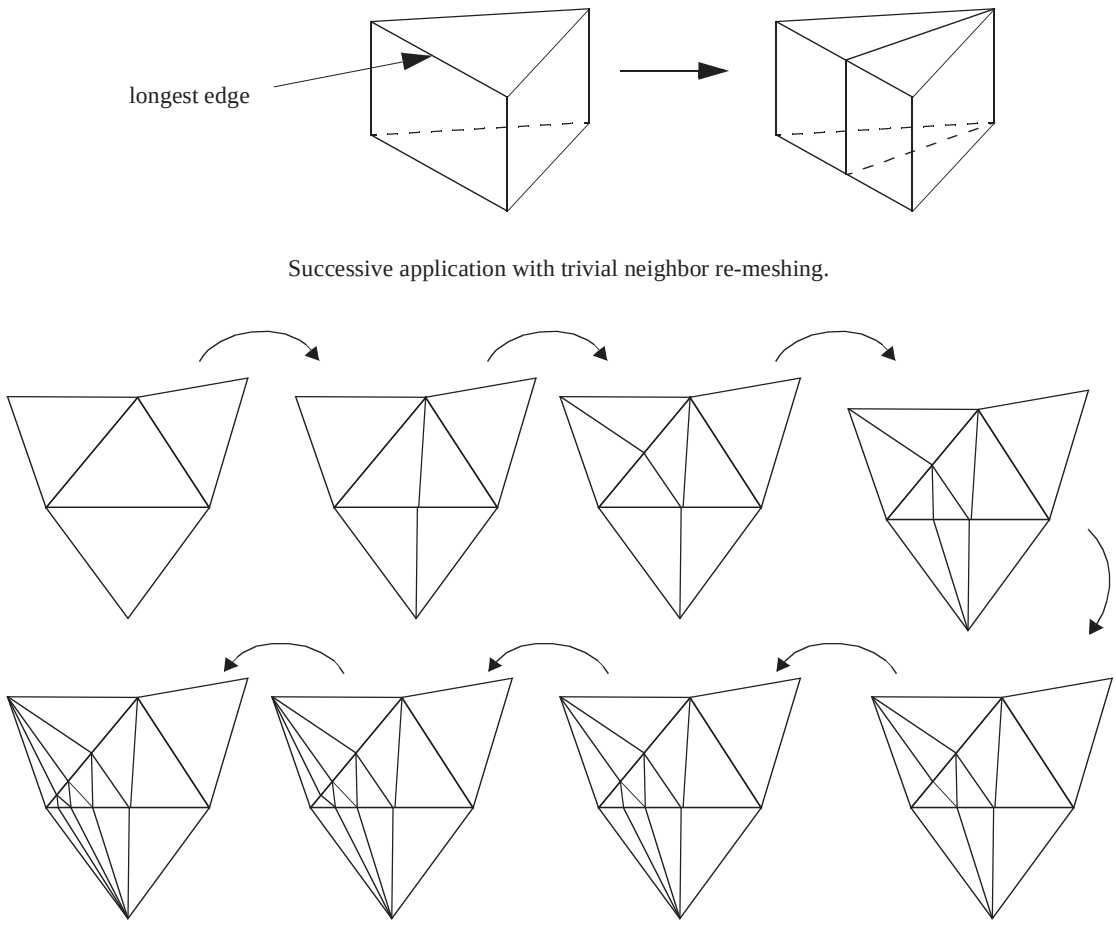



**FIGURE 5.14:** *Subdivision of an element by bisection.*

Delaunay triangulation). An example subdivision process using this approach is shown in Figure 5.15. The algorithm described above can be expressed by a concise pseudo-code given in Algorithm 9.

Once the dynamic subdivision algorithm is incorporated into the simulation, the areas of interest are automatically subdivided when needed. This results in faster simulation times as well as less memory consumed. Figure 5.16 illustrates the difference between results obtained with and without dynamic subdivision. The result on the left was generated using

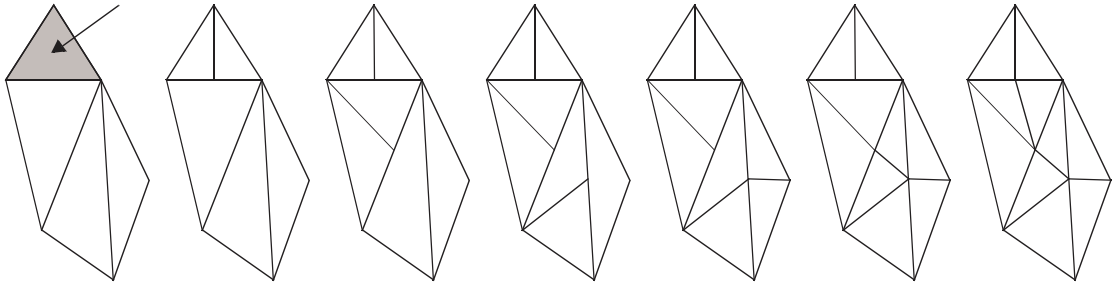


**FIGURE 5.15:** *Splitting an element with a recursive neighbor subdivision.*

```
procedure subdivide_element( e)
        find the longest edge g on top face of e
        split element e at its longest edge
        repeat
                find the neighbor e2 sharing the longest edge g
                if neighbor e2 exists then
                        if the longest edge of e2 is different from edge g then
                                subdivide_element( e2)
                                continue
                        else
                                split element e2 at its longest edge and return
                else
                        return
```

**ALGORITHM 9:** *Dynamic subdivision by recursive bisection.*

a model with 7000 elements, without dynamic subdivision. The result on the right is the result of a simulation where the original model contained only 100 elements, and the dynamic subdivision was enabled. The model on the right eventually contained 1900 elements, and the resulting pattern was generated approximately 10 times faster than the first pattern.

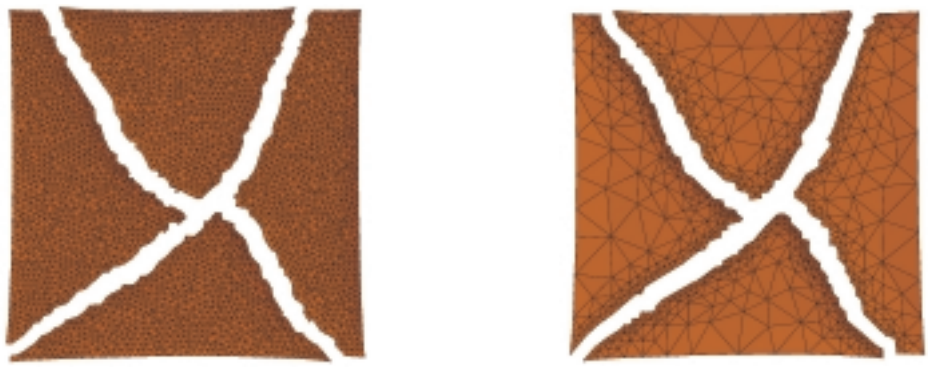


**FIGURE 5.16:** *Results obtained with and without dynamic subdivision.*

## 5.7. Fracture modeling by splitting elements

Another approach to modeling fractures is to split elements along planes of principal stresses. In this approach, stresses are evaluated at the surface nodes, rather than inside the elements. The stress present at a node is called *nodal stress*. When one of the nodal stresses exceeds the material's threshold stress, the plane of fracture is computed. The surrounding elements are then split by the computed fracture plane, as shown in the example in Figure 5.17. This approach is similar to the one described by O'Brien and Hodgins in [45]. The main difference is that I use wedge elements as opposed to the tetrahedral elements used by O'Brien and Hodgins. Another difference comes from the fact that I use the principal stresses to determine the fracture planes, as opposed to a 'separation tensor' used by O'Brien and Hodgins.

This approach of splitting elements along fracture planes is superior to that of element removal in a number of ways. First of all, no material actually disappears from the model. This is a more accurate representation of the real world. Secondly, element removal incorrectly models the fracture, as it releases the stresses of the material in all directions

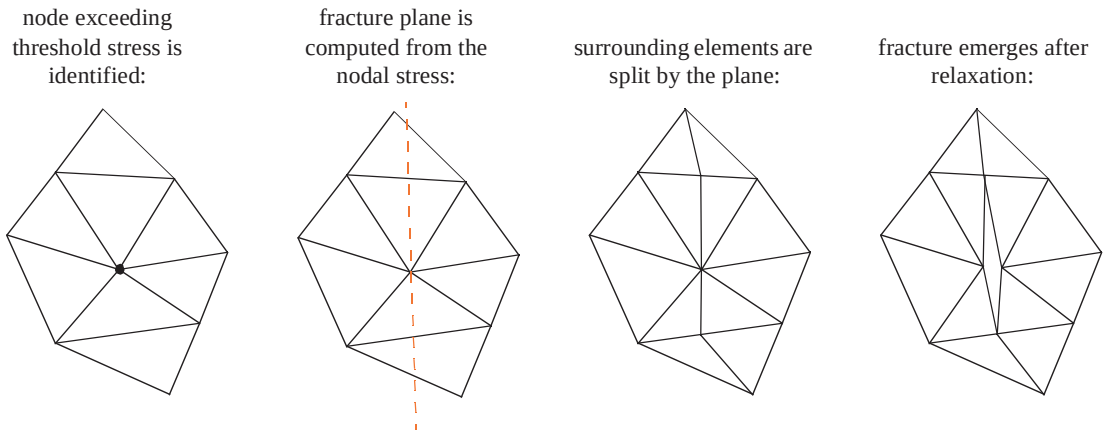
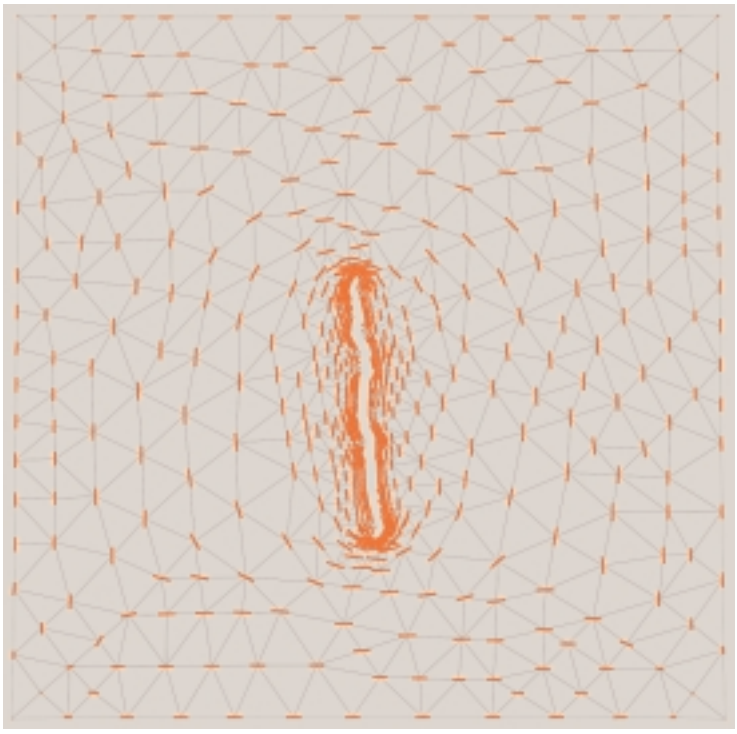



**FIGURE 5.17:** *Example of element splitting at a node.*

around the removed element. This can lead to fracture propagation being influenced by the element's shape. The element splitting approach is more correct in this respect, as it only releases the tensions in the material perpendicular to the plane of fracture. Finally, from the visualization point of view, the element splitting does not create aliasing artifacts of the same magnitude as the element removal technique. The fractures 'cut' through the elements regardless of their shape.

### 5.7.1. Nodal stress evaluation

In order to split an element along the fracture plane, the plane of maximum stress must be calculated. Since such a plane is calculated from the stress tensor evaluated at the node, the nodal stresses at each node in the model must be calculated. Every node is potentially shared by more than one element, so the stress could be determined by choosing one of these elements, and then evaluating the stress at the isoparametric coordinate corresponding to the node. However, the continuity of stress between elements is not guaranteed by the FEM model. It is very likely that each element will evaluate the stress at the same node differently. One approach is to average the stresses as computed inside each



The maximum principal stresses are plotted as vectors.

**FIGURE 5.18:** *Distribution of nodal stresses around a fracture.*

element at the nodal coordinate. This solution, however, assumes that each element will yield a correct stress at its nodes. This is generally not true, unless the nodes happen to be the Gaussian integration points used to derive the elemental stiffness matrix $[K_e]$. In the model described here, none of the Gaussian integration points are located at the nodes. A more common approach is to calculate the nodal stress as the average of stresses in elements connected to the node [75]. The stress in each element is evaluated at the Gaussian point closest to the node. This is the approach I chose for my implementation. An example distribution of nodal stresses around a fracture is shown in Figure 5.18.

Other approaches to nodal stress evaluation exist as well, such as the superconvergent patch recovery technique introduced by Zienkiewicz and Zhu [75]. The basic idea of the superconvergent patch recovery method is to define a patch of elements around the node

for which the nodal stress is to be calculated. The stress tensors in each of these elements are calculated at their Gaussian points. The nodal stress is then calculated using the least square fit approximation [52] of these stress tensors. I implemented this approach but found it unsuitable for my simulations. The results obtained using this method were affected only marginally (not observable to the naked eye), but the performance suffered dramatically due to the increased computation time required by the least squares fitting. The evaluation of nodal stresses using the least square fitting approach was about 100 times slower than by simple averaging. I found that the precision of the nodal stress calculation can be increased more efficiently by a finer discretization.

### 5.7.2. Modeling fractures

The fractures in a material occur along the principal stress plane associated with the maximum principal stress. The surface in the vicinity of the node is split by a planar fracture. Since the surface in the model is represented by the wedges, the geometry of the elements has to be adjusted accordingly. This is implemented in the following stages. The first step is to create a new copy of the node being split. All elements located on one side of the fracture plane keep the original copy of the node, while the elements on the other side the fracture plane are assigned the new copy of the node. There are up to two elements which can be intersected by the fracture plane. These have to be split by the fracture plane at the point of intersection.

If an element is split by the fracture plane, two new nodes are created at the point of intersection of the fracture plane and the element, as illustrated in Figure 5.19. These nodes are used to split the element into two new elements, located on the opposite sides of the fracture plane. Naturally, if the edge being split is shared by another element, this neighbor has to be split as well, to preserve T-junctions. Once all surrounding elements of a node have been processed, the fracture has been introduced into the model. Following

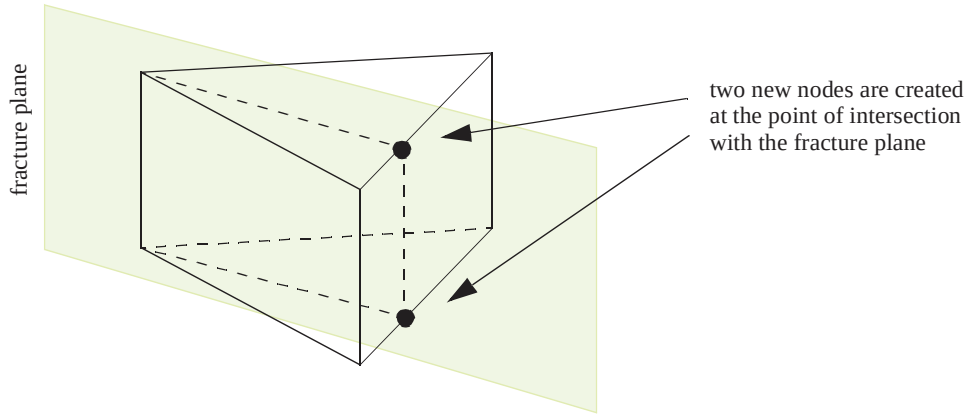



**FIGURE 5.19:** *Splitting an element by a fracture plane.*

```
procedure cut_elements_at_node( Node n)
        create a copy of node n, called cn
        for every element e connected to node n do:
                if e is on the left of the fracture plane then
                        replace its reference to n by cn
                        continue
                if e is on the right of the fracture plane then
                        continue
                split the element along fracture plane
                remesh the neighbor
```

**ALGORITHM 10:** *Element cutting.*

the relaxation step, the fracture emerges as shown in the example in Figure 5.17. The algorithm for cutting an element along the fracture plane is given in Algorithm 10.

### 5.7.3. New fractures

After the nodal stress has been determined, the corresponding principal stresses and stress planes are obtained by computing the eigenvalues and eigenvectors from the stress tensor. If the maximum principal stress exceeds the material's threshold stress, a new fracture is

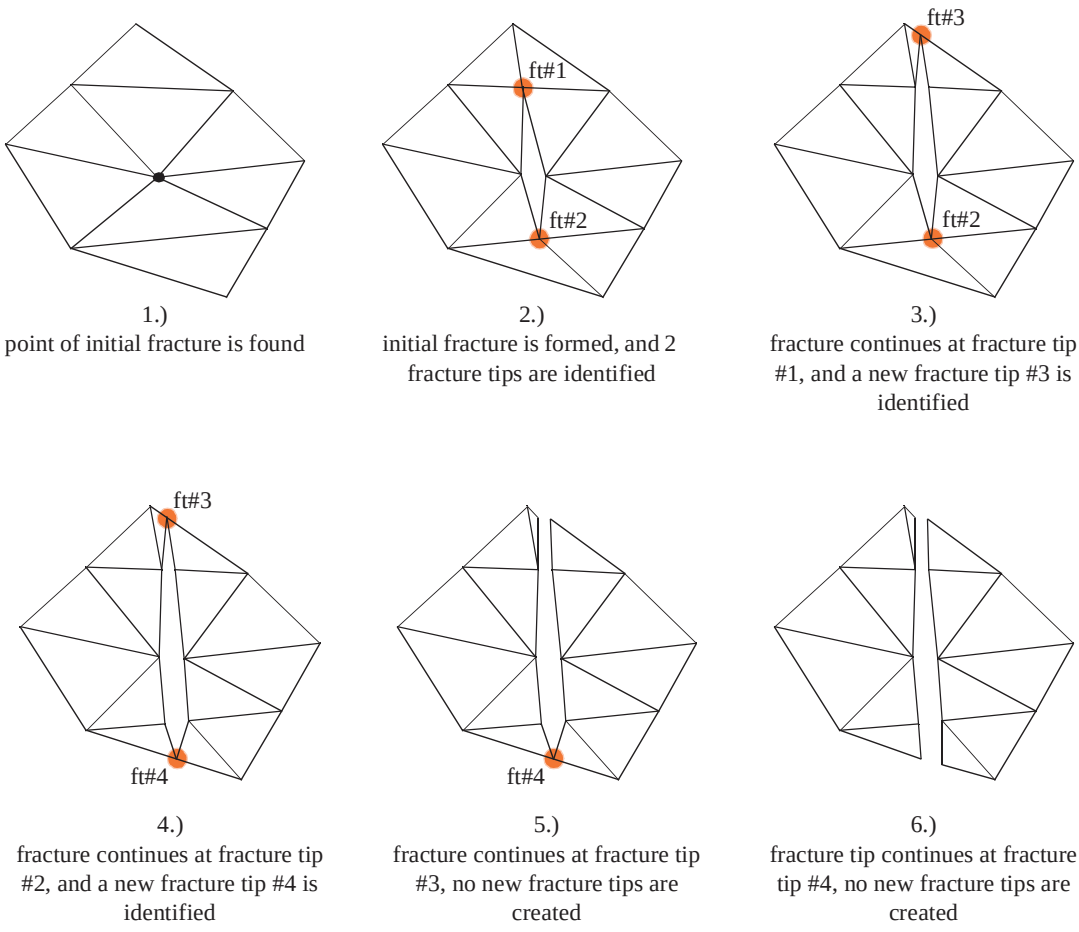



**FIGURE 5.20:** *Tracking fracture tips during fracture propagation.*

formed. If the threshold stress is exceeded at more than one node, the one with the highest ratio of stress to threshold stress is selected, as described in Section 5.6.2.

### 5.7.4. Fracture propagation and termination

To capture the propagating behavior of fractures, I keep track of all fracture tips in the model, illustrated on an example in Figure 5.20. If the fracture is allowed to propagate at

the tip, the adjacent elements are split as described in the previous section, and the fracture tip advances. The simulation loop accommodates the fracture propagation at fracture tips. When the algorithm is looking for nodes where the next fracture will occur, the fracture tips are examined first. Only when there are no more fracture tips left, new fractures are introduced.

Once a new fracture is formed, it propagates through the material until it terminates. Two approaches exist to determine how long the fracture propagates in elastic materials. The approach introduced by Inglis [28] is based on evaluating the stress intensity at the fracture tips as a function of distance from the tip. The second approach was introduced by Griffith [20], which states that a fracture propagates as long as the potential energy released by the fracture exceeds the energy required to form the fracture. I have implemented the Griffith energy approach, because it could be integrated with the rest of my model easier than the Inglis's approach.

The implementation of Griffith criteria (Algorithm 11) for fracture termination is based on measuring the elastic strain energy of the system before ($e_1$) and after ($e_2$) a crack extension. If the amount of energy released by a crack extension per unit area ($G$) is larger than the amount of energy needed to open the fracture ($G_{IC}$), then the fracture extends. If the amount of energy released is too small, the fracture is terminated. The stored elastic strain energy in the material layer is calculated as a sum of elastic strain energies of all elements. The elastic strain energy of a single element is calculated as a dot-product of the displacement vector and the corresponding nodal force vector. Finally, $K_{IC}$ is a material property, called fracture toughness, indicating what the intensity factor around a fracture tip has to be in order for the fracture to propagate. The example in Figure 5.21 illustrates how fractures terminate when the crack tips reach an area of decreased stresses.

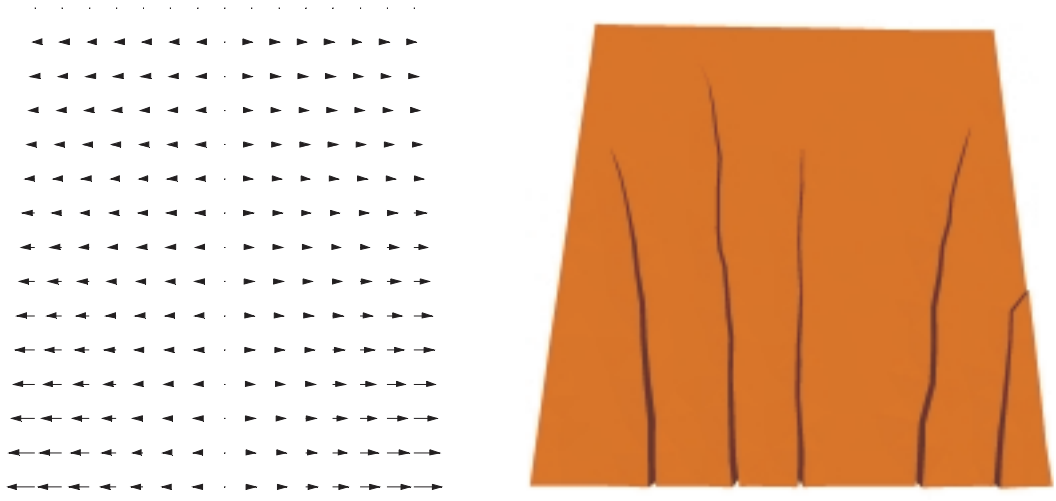
```
procedure extend_fracture_at_tip( t)
        e1 = measure_elastic_strain_energy()
        introduce_fracture_at_tip(t)
        a = area of the newly formed fracture wall
        find the equilibrium of the model
        e2 = measure_elastic_strain_energy()
        G = (e2 – e1)/a  # G=released energy per unit area
        Gic = (1 – υ^2 K_IC^2)/E  # Gic=energy required to extend fracture by one unit
        if( G > Gic)
                # accept crack extension and continue fracture
                # at a newly created fracture tip
        else
                # terminate fracture

function measure_elastic_strain_energy()
        total_energy = 0
        # sum up the strain energies of all elements
        for each element e do
                F = nodal forces of e # F is a 18x1 vector
                u = nodal displacements of e # u is a 18x1 vector
                total_energy = F.u # dot product
        return total_energy
```

**ALGORITHM 11:** *Griffith criteria for fracture termination*

### 5.7.5. Avoiding degenerate elements

When the fracture plane intersects an element close to one of its nodes, a degenerate wedge may be formed as a result of splitting. Some way of dealing with these degenerate elements has to be devised. One possible way is not to allow degenerate elements to be created - a method suggested by O'Brien and Hodgins [45]. When a fracture plane intersects an element too close to one of its nodes, the fracture plane is rotated by a small amount, as if it passed directly through that node. The element then does not have to be divided at all, since it is entirely located on one side of the plane. Figure 5.22 illustrates this technique. Unfortunately, this approach suffers from fracture directions being occasionally influenced by the geometry of the surface subdivision. Namely, when the



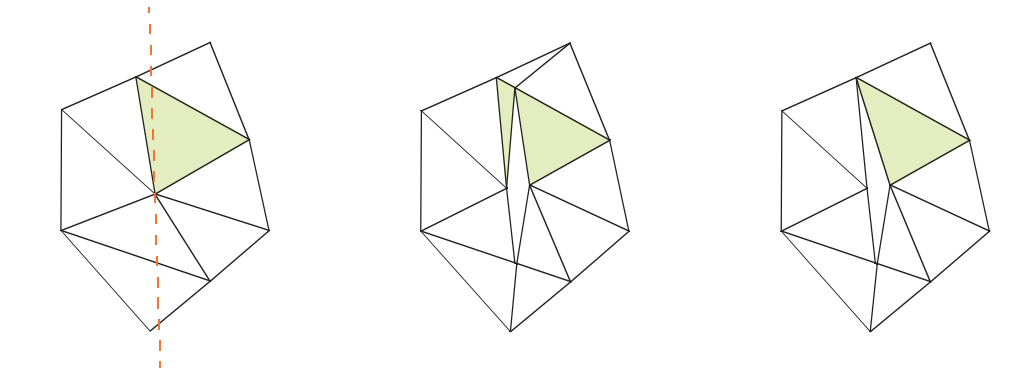
the velocity vector field implementing anisotropic growth (much more stretching at the bottom than at the top)

the resulting pattern - fractures initiate at the bottom, propagate to the top but terminate before reaching the other side

**FIGURE 5.21:** *Fracture termination.*

fracture plane is too close to an existing edge, it will be ‘snapped’ to match the direction of the edge.

The approach I adopted in my implementation is opposite to the one suggested by O’Brien and Hodgins [45]. Instead of snapping the fracture plane to a nearly parallel edge, such edge is snapped to the fracture plane. This is implemented using edge collapsing, which is a technique used in triangular meshes to remove undesired edges from the triangulation (for a sample application see [26]). An edge can be collapsed if one of its nodes is completely surrounded by triangles. Edge collapsing is implemented by removing one of the edge’s nodes (the one that is surrounded by elements), and then adjusting all triangles that referenced the removed node to point to the other node of the edge. Naturally, the triangles which used to share the edge that was collapsed degenerate into a single line and therefore must be removed from the system. An extension of this algorithm to wedge



fracture plane intersects the shaded element close to an edge

if the element is split by the fracture plane, degenerate element(s) will be formed

the fracture plane is rotated into a direction parallel with the edge - no degenerate elements are formed

**FIGURE 5.22:** *Avoiding degenerate elements by rotating the fracture plane.*

```
procedure cut_elements_at_node( Node n)
        create a copy of node n, called cn
        for every element e connected to node n do:
                if e is above fracture plane then
                        replace its reference to n by cn
                        continue
                if e is below fracture plane then
                        continue
                split the element along fracture plane
                remesh the neighbor
                if one of the resulting 2 triangle is degenerate then
                        collapse the corresponding edge
```

**ALGORITHM 12:** *Element cutting combined with edge collapsing.*

elements is straightforward - collapsing an edge is equivalent to collapsing one of the sides of the wedge elements. The element cutting algorithm is given in Algorithm 12. An illustration of the above algorithm is given in Figure 5.23.

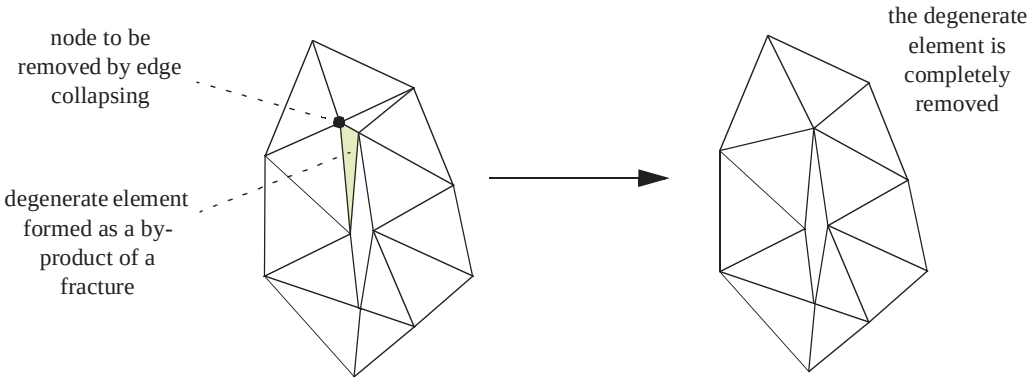



**FIGURE 5.23:** *Elimination of deformed elements by edge collapsing.*

### 5.7.6. Dynamic subdivision around fracture tips

Smaller elements lead to higher accuracy when calculating nodal stresses, which is a special concern in the areas near fracture tips. If the error in the calculation of the nodal stress at a fracture tip is too large, the fracture might incorrectly change the direction of propagation, or even stop propagating altogether. A simple solution would be to globally increase the level of discretization, so that all elements are small enough to achieve acceptable accuracy. Obviously, such an approach is undesirable since it would lead to increased memory requirements and prolonged simulation times. A better solution is to refine the mesh around the fracture tips dynamically, as the simulation progresses. I implemented such an algorithm, as described below.

Once the node at which the fracture will be formed is located, all elements in the vicinity of the node are examined. If any of these elements are too large, they are subdivided using the Rivara et al. algorithm [58] presented in Section 5.6.4. This process is repeated until all elements in the neighborhood of the node are smaller than some desired value. To

define the elements in the "vicinity" of a node, I use the following procedure. First, all elements sharing the node are placed in a list. Then, the neighbors of each element in the list are also added to the list. By controlling the number of iterations of the second step, larger or smaller neighborhoods of elements around the node can be selected. I found that a single iteration leads to satisfactory results.

This dynamic subdivision of elements around fracture tips is analogous to the dynamic subdivision used for modeling fractures by element removal. It allows the use of large elements in areas of no fractures, as it refines the mesh only in places where the increased precision is required. By using the dynamic subdivision, resources are automatically used where they are most needed. The end result is smaller memory requirements and faster simulation times.

### 5.7.7. Adaptive mesh refinement

As mentioned earlier, the nodal stresses evaluated in each element can be quite discontinuous. Such discontinuities can be used as an indicator that the discretization of the continuum is not fine enough, and that the error in the stress calculations will be significant. The discontinuities of stress at nodes also indicate where the mesh should be refined. To further improve the accuracy of the generated fractures, I implemented an adaptive mesh refinement algorithm to minimize the nodal stress discontinuities.

The basic idea of the algorithm is to refine all elements which share a node at which the discontinuity (or error) in the nodal stress is unacceptable. To decide whether the discontinuity is acceptable or not, I use the refinement indicator suggested by Bastian et al. [4]. The discontinuity is unacceptable if the relative difference of the Von-Mises

```
procedure refine_mesh( vm_threshold )
        max_eigenvalue = smallest possible number
        for each node n do:
                reset n.max_vm and n.min_vm
                for each element e sharing node n do:
                        calculate stress s in element e at gauss. point closest to n
                        calculate eigenvalues s1,s2 and s3 from s
                        max_eigenvalue = max( max_eigenvalue, s1, s2, s3)
                        calculate Von-Mises stress svm from s
                        if n.max_vm < svm then n.max_vm = svm
                        if n.min_vm > svm then n.min_vm = svm
        for each node n do:
                diff = (n.max_vm - n.min_vm) / max_eigenvalue
                if diff > vm_threshold then
                        mark all elements sharing node n
        for each marked element e do:
                subdivide element e
```

**ALGORITHM 13:** *Adaptive mesh refinement based on Von-Mises stress indicator.*

stresses [16] from each surrounding element at the node is larger than some user-controlled threshold. The Von-Mises stress is defined as:

$$\sigma_{VM} = \sqrt{\frac{(\sigma_1 - \sigma_2)^2 + (\sigma_1 - \sigma_3)^2 + (\sigma_2 - \sigma_3)^2}{2}} ,$$

where the $\sigma_i$ 's denote the eigenvalues of the stress tensor. The relative difference of the Von-Mises stress calculated at the node is computed by calculating the difference between the maximum and minimum values of $\sigma_{VM}$ obtained at the node, divided by the maximum eigenvalue encountered. The resulting algorithm implementing the adaptive mesh refinement is given in Algorithm 13.

### 5.7.8. Local multi-resolution meshes for nodal stress evaluation

The dynamic subdivision of elements near fracture tips outlined in Section 5.7.6 improves the accuracy of nodal stress calculations by refining the mesh around fracture tips. Yet, although it provides a significant performance improvement over the approach in which the discretization is increased globally, it still yields an unmanageable number of elements for large models. It is possible to improve on this design even further.

The elements around a fracture tip must be quite small in order to calculate the stress at the fracture tip correctly. I observed that after the nodal stress is calculated, the need for the small elements disappears. This observation prompted me to design a local multi-resolution mesh method for evaluating nodal stresses at fracture tips.

Multi-resolution meshes are sometimes used in FEM applications to calculate nodal values with increased accuracy. The overall algorithm is straightforward. First, the geometry of the original mesh is stored, so that it can be later recovered. Next, the entire mesh is refined to the desired level and the nodal values are calculated. The result is then superimposed onto the original, stored mesh. In the end, the original mesh is retained, but the accuracy of the results has been improved.

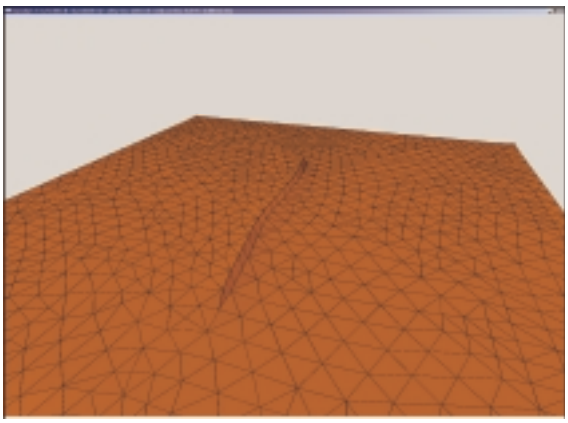
Since the increased accuracy of nodal stresses is only needed around the fracture tips, there is no need to refine the entire mesh. Refinement of the entire mesh would negatively affect the performance of the simulations, as the multi-resolution mesh would have to be constructed every time a change is introduced into the model, which happens with the formation of every new fracture. Consequently, I implemented a local multi-resolution mesh, constructed only around fracture tips. This is achieved by cutting out the part of the mesh in the neighborhood of the fracture tips, and then refining the extracted sub-model. The sub-model is selected by first marking all nodes in some user specified radius around

the fracture tips, and then selecting all elements connected to the marked nodes. The marked elements define a mesh, which is passed to the multi-resolution processing. This approach of sub-model extraction assumes that the elements beyond the radius of the sub-model do not affect the calculation of the stresses at the crack tip. To this end, all unmarked nodes in the sub-model are treated as fixed nodes. Once the sub-model is extracted, it is refined around the fracture tip as described in Section 5.7.6. The nodal stress is then calculated at the fracture tip and transferred back to the original model. A graphical illustration of this process is shown in Figure 5.24.

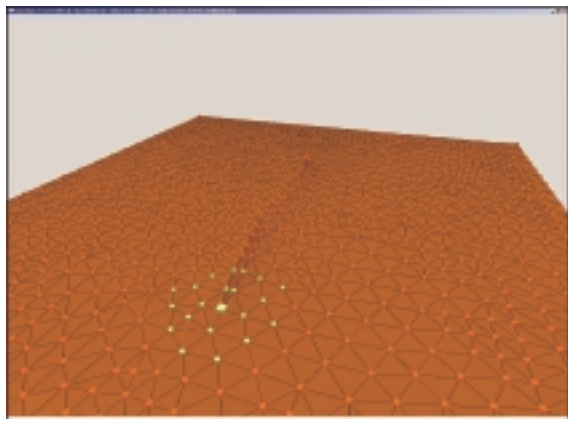
### 5.7.9. Improving element shapes around crack tips

The accuracy of the nodal stress calculation depends on both the size of the elements around the node, as well as their shape [74]. The optimum shape of the wedge element would have the top and bottom faces as 60-degree triangles. Even though the edge-collapsing technique prevents formation of badly shaped triangular faces, it still produces elements of sub-optimal shapes.

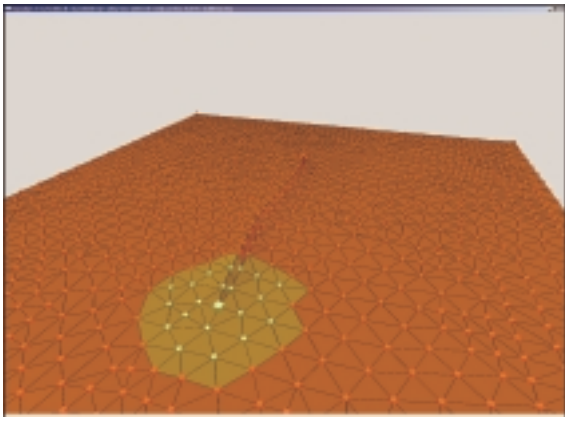
To further improve the quality of elements near fracture tips, I extended the mesh smoothing technique developed by Zhou and Shimada [72] to three dimensions. The mesh smoothing proposed by Zhou and Shimada is based on a system of torsion springs. Each node is iteratively repelled into a position where the force acting on it is in an equilibrium. The adaptation to three dimensions consists of assuring that the nodes are not forced off the surface. This is achieved by snapping the nodes back onto the background surface after each iteration. To make the method work on discretized surfaces, the position to which the node is snapped is derived purely from the triangulation. This is achieved by first determining the average surface normal from the surrounding triangles. A plane is then constructed which passes through a node and is perpendicular to the node's normal. When



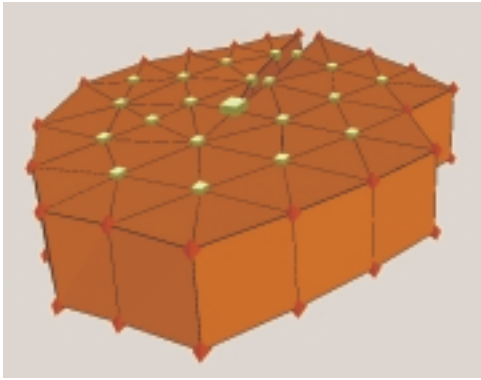
a) view of an existing fracture



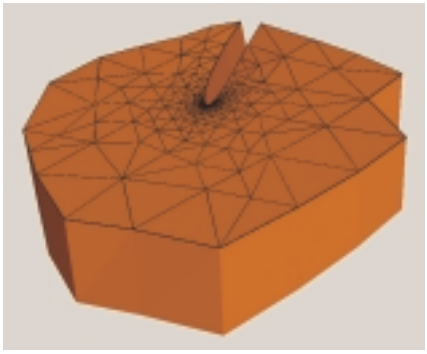
b) the nodes in some radius around the fracture tip are selected



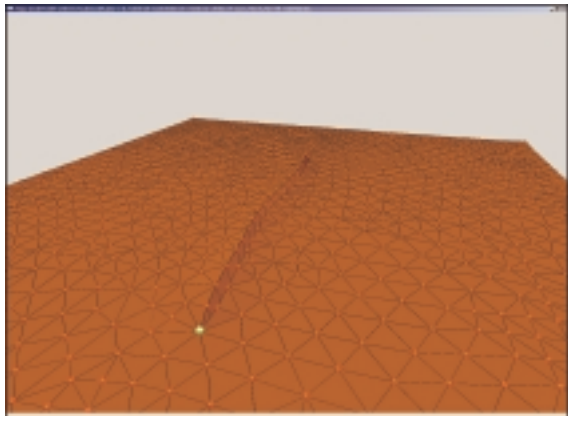
c) the elements connected to selected nodes are selected



d) the unmarked nodes in the sub-model are treated as fixed nodes



e) the sub-model is refined around the fracture tip and the nodal stress is calculated



f) the nodal stress is used to propagate the fracture

**FIGURE 5.24:** *Graphical illustration of the local multi-resolution mesh method.*

the node is repelled into a position off its plane, the node is simply projected back onto this plane. This process is repeated for each iteration.

Because global application of the mesh smoothing would require re-computation of all elemental stiffness matrices, I apply the smoothing locally, only in the neighborhood of the fracture tips. For every new fracture tip, all nodes in a user controlled radius around the fracture tip are marked. From these marked nodes, the ones which are not completely surrounded by elements (nodes on the boundaries of the material) are unmarked. The positions of the remaining marked nodes are adjusted using the angle-smoothing algorithm. After the nodal positions are adjusted, the reference shapes of each element have to be recomputed. Since the elemental stiffness matrices are defined in terms of elements' reference shapes, they have to be recalculated as well. Naturally, by changing the elemental stiffness matrices, the global stiffness matrix is properly adjusted to reflect the changes.

## 5.8. Adaptive relaxation

To speed up the process of finding an equilibrium state of the model, I designed and implemented an adaptive relaxation algorithm. This algorithm is based on the idea that local changes in the geometry of the model, such as introduced by fracture formation, splitting elements or repositioning of nodes during mesh smoothing, do not affect all nodes in the system. Consequently, it is a waste of resources to perform the relaxation on all nodes in the system. Only those nodes in the areas where the changes were introduced need to be relaxed.

The adaptive relaxation proceeds in three steps. In the first step, the nodes affected by the the local change are marked. In the second step, the relaxation is performed on the marked nodes, treating the remaining nodes as fixed nodes. Finally, the error of the new solution is

calculated. If the error is unacceptable, a global relaxation is applied and the radius of local relaxation is increased for future use. Otherwise the radius is reduced. These steps are now explained in full detail.

### 5.8.1. Selecting nodes for local relaxation

The model needs to be relaxed whenever a change is introduced into the model. Changes to the model may be caused by various actions, such as fracture formation or element splitting. To find the nodes affected by such change in the model, the local relaxation procedure must determine where the change has taken place. One solution is to keep track of the local changes introduced into the model, and then pass this information to the local relaxation algorithm. The local relaxation algorithm would then select all nodes in the areas where the changes were applied. In order to make the local relaxation algorithm more general, I implemented a simple, yet efficient way to automatically determine where the local changes were applied.

Whenever the model needs to be relaxed, the error of the current solution is calculated in a manner identical to that used by the conjugate gradient method. Specifically, the error is calculated by evaluating the left hand side of Eq. 5.48 on page 69 by setting $[P]$ to the current nodal coordinates, and then subtracting the right hand side of Eq. 5.48 from the result. The resulting column vector indicates which nodal coordinates contain an unacceptable error. These nodes are marked for recalculation.

The next step is to mark all nodes that will likely be affected by the relaxation of the already marked nodes. They will lie in the proximity of the marked nodes, and will also have to be relaxed in order to achieve an equilibrium state. To this end I select all nodes whose closest distance to any marked node is less than some threshold value, which I call the ‘local relaxation radius’. To take into account both the curvature of the surface, as well

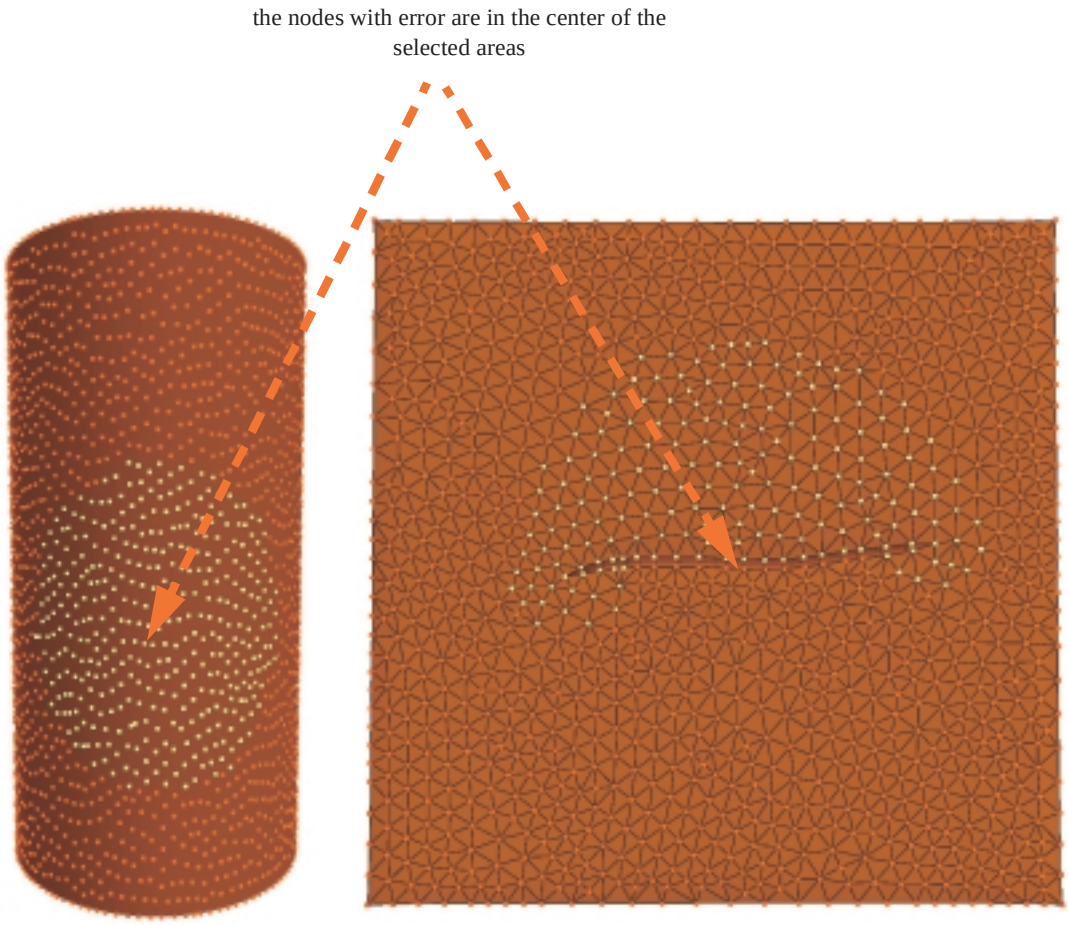



**FIGURE 5.25:** *Selection of nodes for local relaxation.*

as the discontinuities of the surface introduced by fractures, I define the distance between two nodes as the shortest edge path along the top faces of elements. By defining the distances this way, discontinuities in the material caused by fractures will present natural barriers - i.e. two nodes separated by a fracture, even if physically close to one another, will likely have no effect on each other. Figure 5.25 illustrates how the nodes are selected on a cylinder and in a plane containing a fracture.

### 5.8.2. Local relaxation

Once the nodes in the neighborhood of local changes have been marked, a local relaxation is applied to find their equilibrium state. This is implemented by treating all remaining nodes as fixed and solving Eq. 5.51 on page 73 for the unknowns. Since in many cases the number of marked nodes is much smaller than the number of unmarked nodes, the Eq. 5.51 reduces to a very small system of linear equations. Such a small system of linear equations is solved very efficiently, improving the overall performance of the relaxation.

### 5.8.3. Adaptive control of the local relaxation radius

Since it is difficult to estimate a good value for the local relaxation radius ahead of time, I decided to adaptively modify this radius as the simulation progresses. After a local simulation step is applied, the global error of the solution is re-evaluated using Eq. 5.51 on page 73. If the global error is larger than the required precision, the global relaxation is applied. The presence of the global error indicates that the current value of the local relaxation radius is too small and therefore must be increased. If the global error after the local relaxation is acceptable, the local relaxation radius is reduced.

### 5.8.4. Local stress recalculation

Following the relaxation step, the stresses in the system must be recalculated. Recall that two types of stresses are calculated: elemental stresses and nodal stresses, the nodal stresses being calculated from the elemental stresses. If local relaxation was successful, i.e. global error was acceptable and global relaxation therefore did not have to be applied, then it is not necessary to recalculate all elemental stresses in the system. Specifically, since only a small subset of nodes actually changed their coordinates, elemental stresses

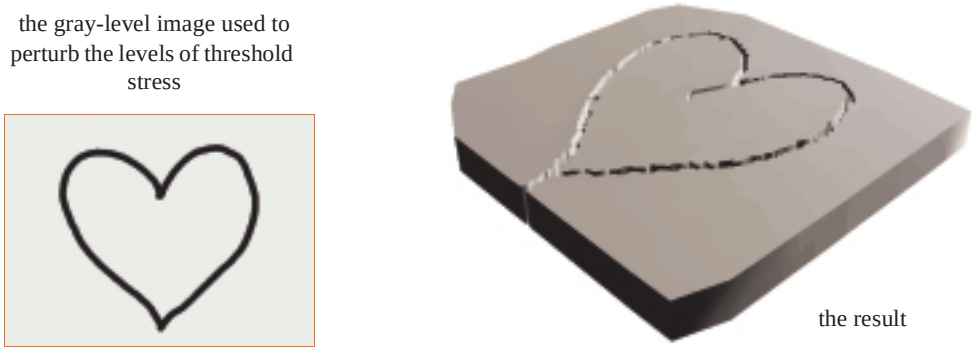



**FIGURE 5.26:** *Synthesizing "breaking heart".*

need only be recalculated in elements sharing such nodes. Nodal stresses are then re-calculated only for those nodes shared by elements whose elemental stresses were recalculated.

## 5.9. Randomizing material properties

Real life materials are rarely truly homogeneous. To model non-homogeneous properties of materials in my simulations, I allow the material properties of each element to be perturbed by a small amount. The level of perturbation is specified by a gray-level texture image, which is mapped onto the surface. The material properties at a given point on the surface are determined by finding the value of the pixel mapped at that coordinate. To find the perturbation of the material property for an element, all pixels of the gray-level image which map inside the element are averaged. This is done both when the initial mesh is generated, as well as during simulations (for example after local refinement and Delaunay re-triangulation). Figure 5.26 (left) shows a gray-level texture map representing a heart, which is used to perturb the threshold stresses over the surface. The dark gray represents

areas of lowered threshold stress. From the resulting pattern (Figure 5.26 right) in can be seen that cracks only formed in areas of lowered threshold stress.

## 5.10. Discussion of results

The main reason for abandoning the mass-spring approach for simulating fracture formation was its inability to represent the continuous properties of the material layer. This deficiency of the mass-spring model was particularly visible when I attempted to simulate anisotropic planar growth. The underlying mesh of springs affected the directionality of the fractures at a global level. In the finite element approach, the generated fractures are not affected by the underlying mesh, at least not to the same degree as the mass-spring approach. This is illustrated in Figure 5.27, where the results were generated using the same mesh, and only varying the velocity vector fields.

One of the observed properties of real cracked mud is the tendency of the cracks to form 90 degree angles with respect to each other. To verify that the simulated mud behaves the same way, I synthesized a mud pattern by allowing the material layer to shrink with respect to a static background. The result is illustrated in Figure 5.27. The fractures in the synthesized mud indeed often intersect each other perpendicularly.

Another interesting property of drying mud is that more and thinner fractures form in areas where the mud is thin. This behavior of cracks was successfully simulated by changing the height of the material layer modeled. The resulting mesh of elements, as well as a comparison of a synthesized pattern to real cracked mud is illustrated in Figure 5.29. The dynamic subdivision method was used on an initial mesh of 100 elements. The result contained approximately 40,000 elements. The result was produced with the element removal technique, and the aliasing effect on fractures can be noticed.

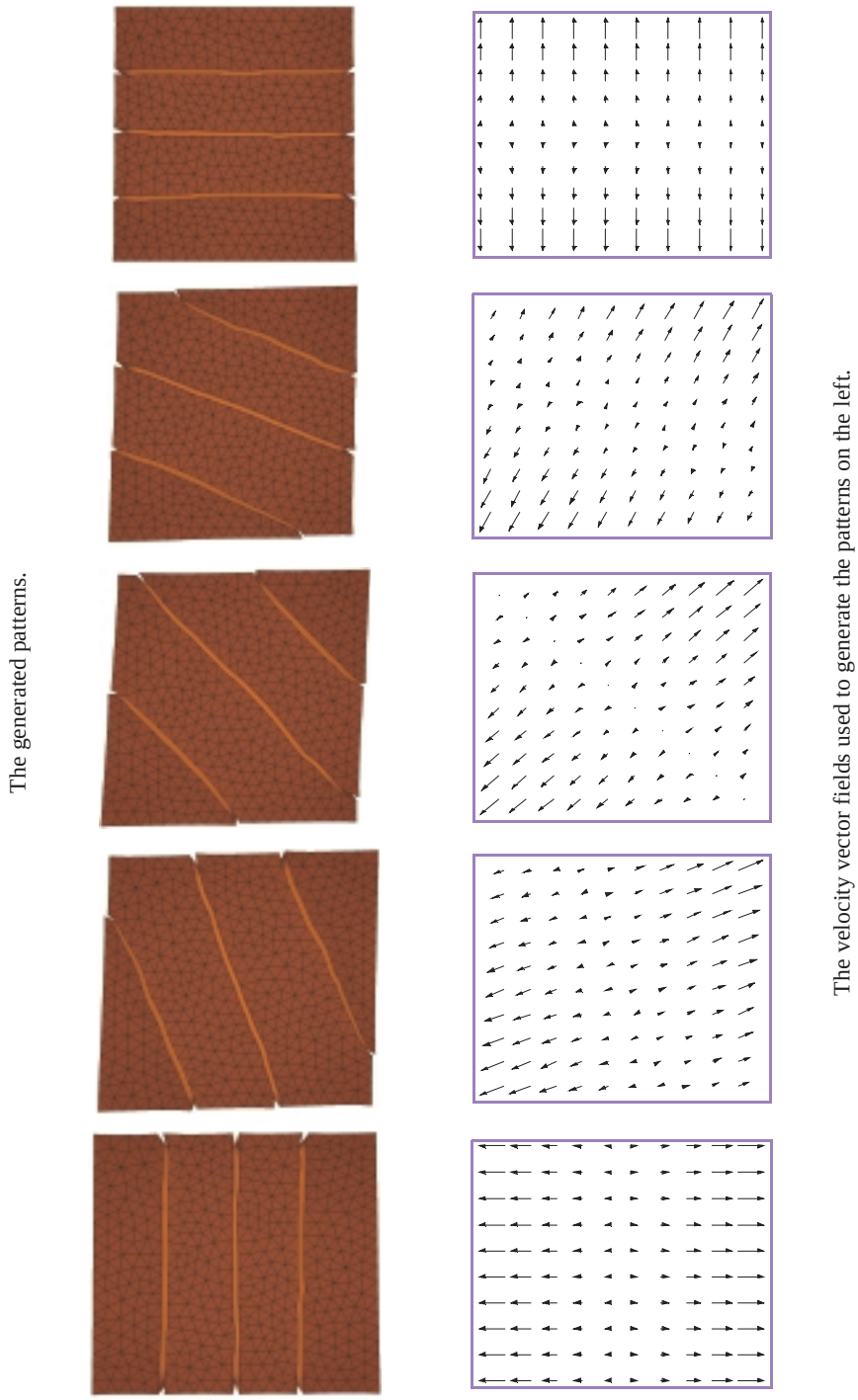
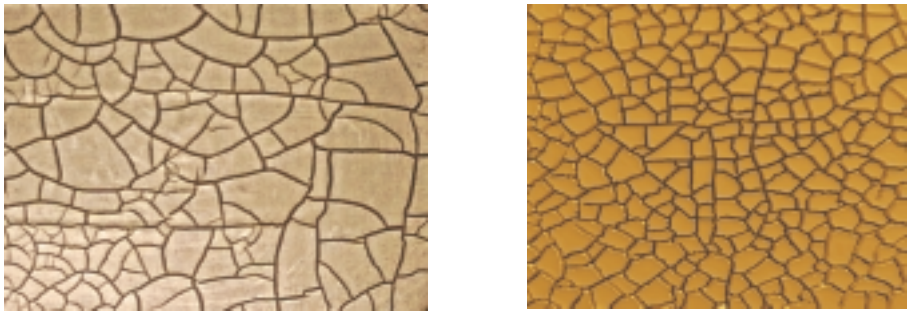



**FIGURE 5.27:** *Fracture patterns generated using anisotropic growth.*

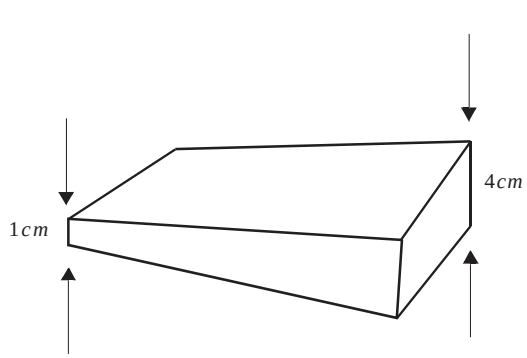


real mud synthesized mud (using element removal)

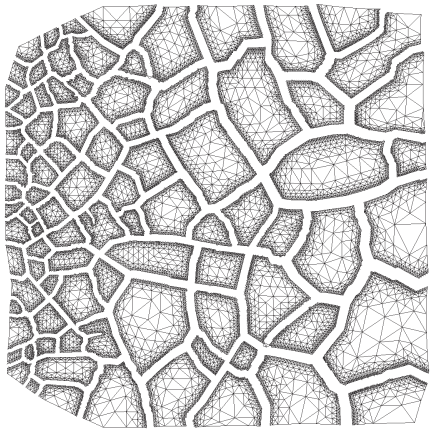
**FIGURE 5.28:** *Fractures often form 90 degree angles with respect to each other.*

A visually more pleasing result was obtained on the same model by using the element splitting technique, illustrated in Figure 5.30 at the bottom. The superiority of modeling fractures using the element splitting technique can be observed by comparing this result to the one shown in Figure 5.28. The fracture surfaces in the synthesized pattern are free of aliasing artifacts, although the final result contained only 9,000 elements. The temporal sequence shown at the top of Figure 5.30 illustrates the ability of the presented model to capture the development of the fracture pattern.

To model non-homogeneous materials, I allow various material properties to be perturbed by user-specified textures. Figure 5.31 at the top illustrates the result of perturbing the Young modulus and the threshold stress of the material. The simulated growth was anisotropic - exclusively in the horizontal direction. As a result of the perturbation of the simulation parameters, the fractures are no longer straight (compare to the pattern at the bottom of Figure 5.27). A similar twisting effect on the fractures can be obtained by applying growth to a material layer of non-uniform thickness, as illustrated in Figure 5.31 at the bottom.

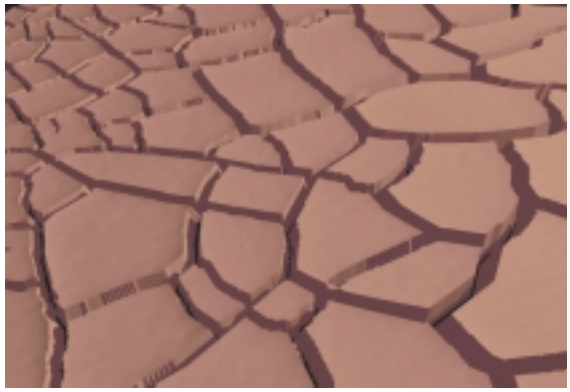



the thickness of the modeled material
layer changes linearly from left to right



the resulting mesh

real mud



synthesized mud

**FIGURE 5.29:** *Varying density and size of cracks.*

The methods for simulating fracture formation I presented in this thesis can be applied to arbitrary surfaces. To demonstrate this, I have synthesized crack patterns on a growing sphere, considering both isogonic and anisotropic growth. The pattern in Figure 5.32 on the left was generated when the sphere grew at the same rate in all directions, while the pattern on the right was obtained by allowing the sphere to elongate 4 times faster along the Z-axis.

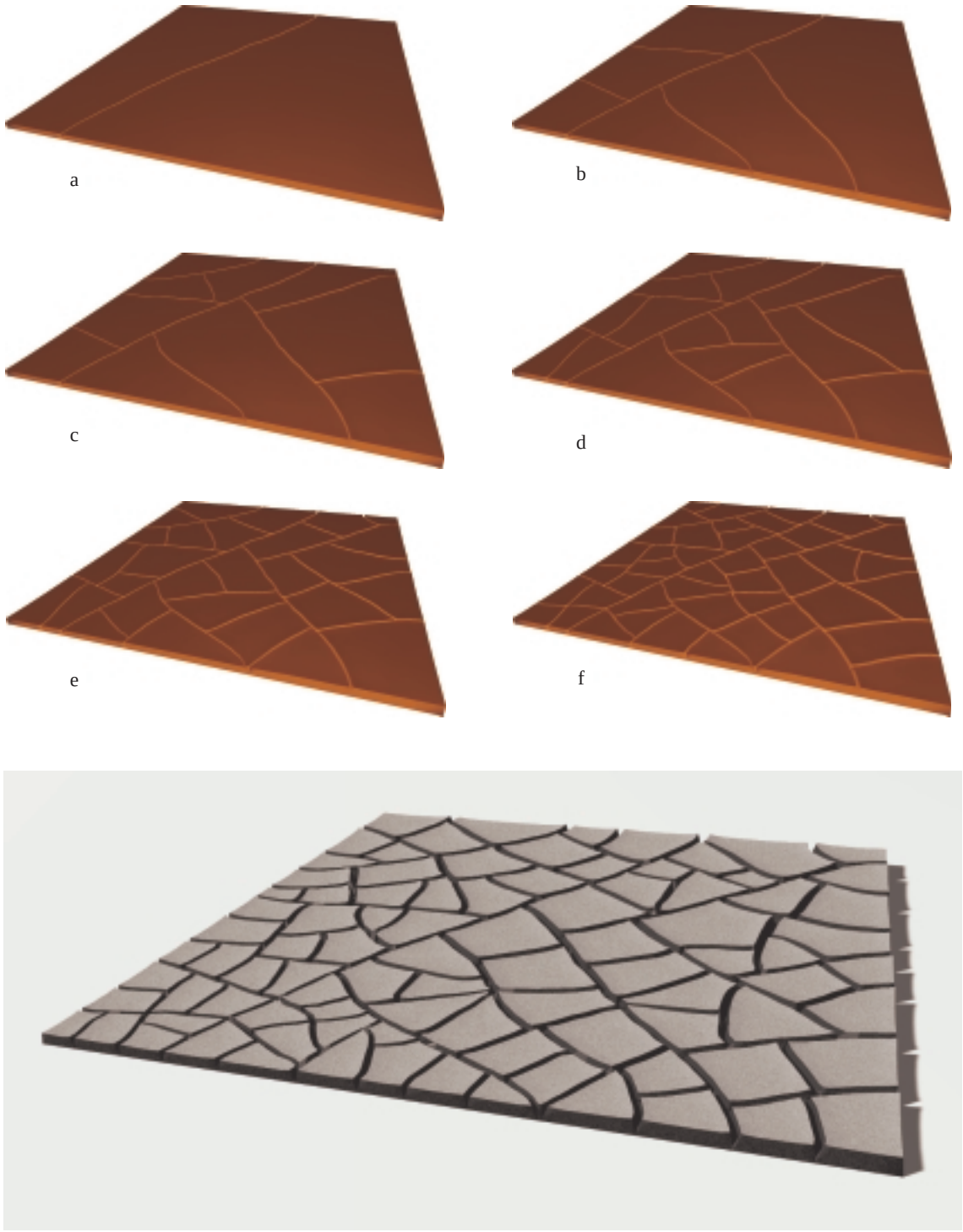



**FIGURE 5.30:** *Temporal sequence of a simulated fracture formation in drying mud.*

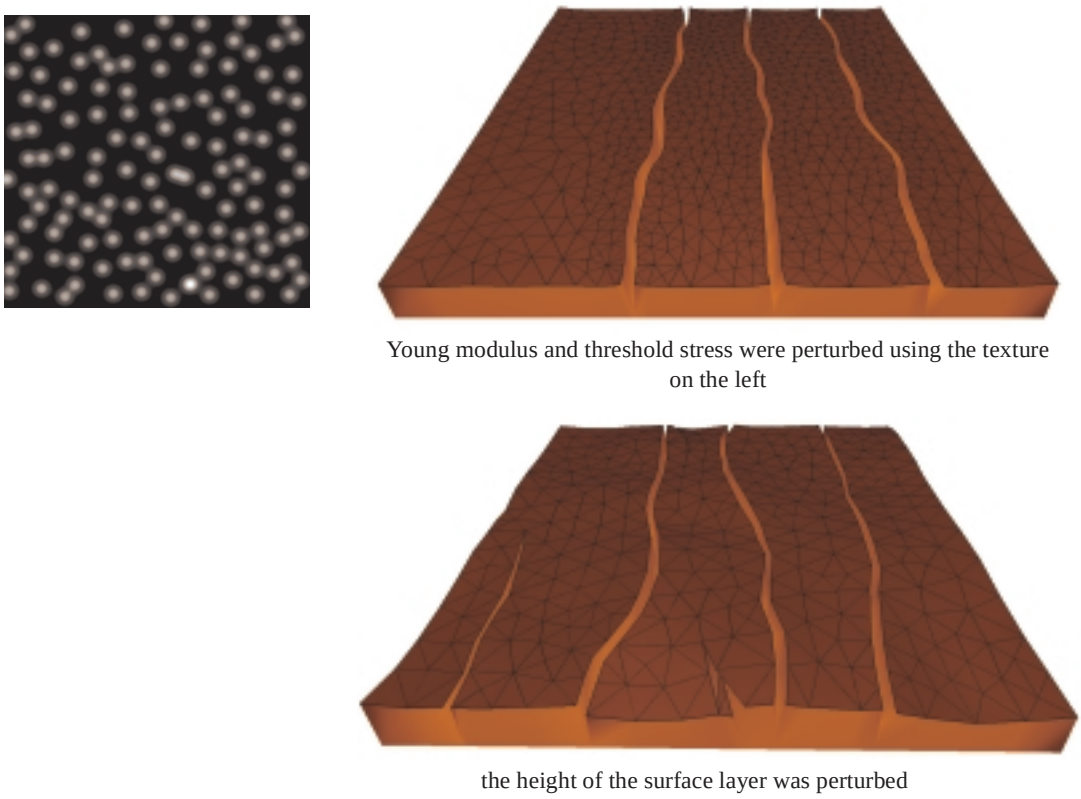



**FIGURE 5.31:** *Perturbation of material properties.*

A number of interesting, bark-like patterns have been obtained by simulating anisotropic growth on cylindrical surfaces while varying some of the simulation parameters. A selection of the synthesized patterns is shown in Figure 5.33. Some of the generated patterns are compared to real-life bark in Figure 5.34. Although the difference between the real and synthesized patterns is noticeable, the correspondence between the geometries of the real and simulated fracture patterns is significant. The simulation parameters used to generate the patterns in Figure 5.33 are listed in Table 5.2.

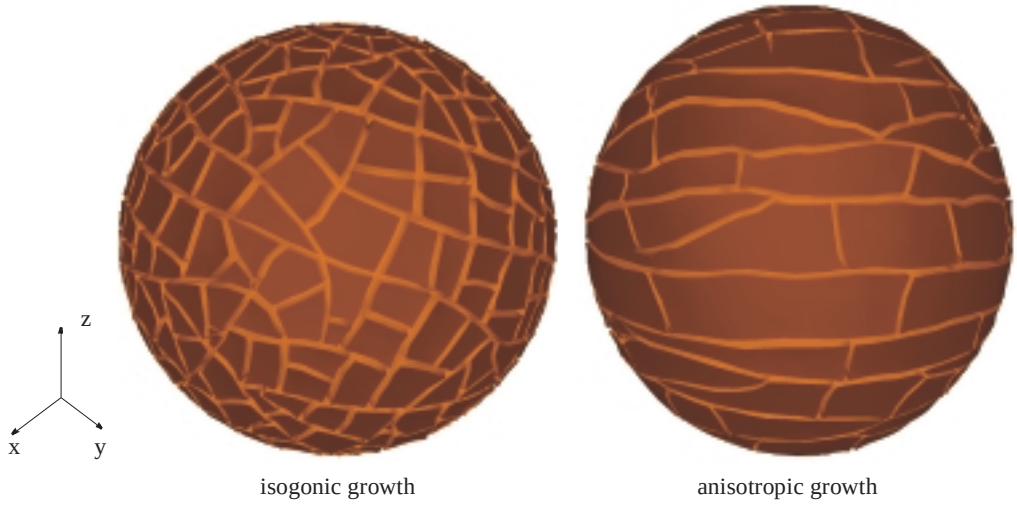



**FIGURE 5.32:** *Synthesized fracture patterns on growing sphere.*

It is interesting to note that some of the generated patterns are not observed in real life (e.g. patterns a, b, g and j in Figure 5.33). This raises an intriguing question: Which combinations of parameters lead to patterns that cannot be observed in nature and why? I believe that finding the answer to this question would be a worthwhile direction of future research as it could help us better understand nature.

This concludes my presentation of the finite element approach to modeling fractures on differentially growing bi-layered surfaces. In the next chapter, I end the thesis by summarizing the contributions of my work and by suggesting areas for future research.

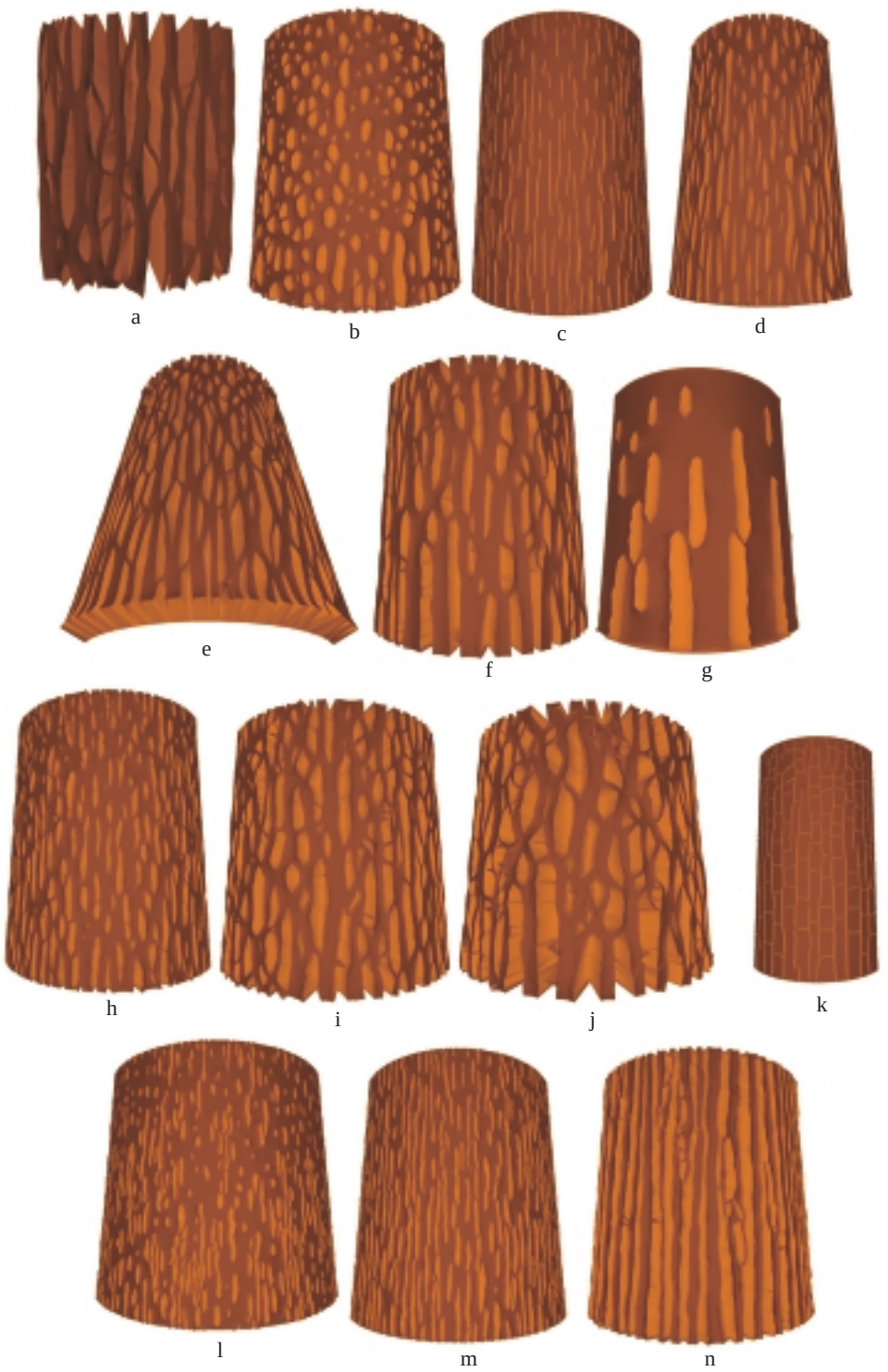



**FIGURE 5.33:** *Synthesized bark-like patterns.*

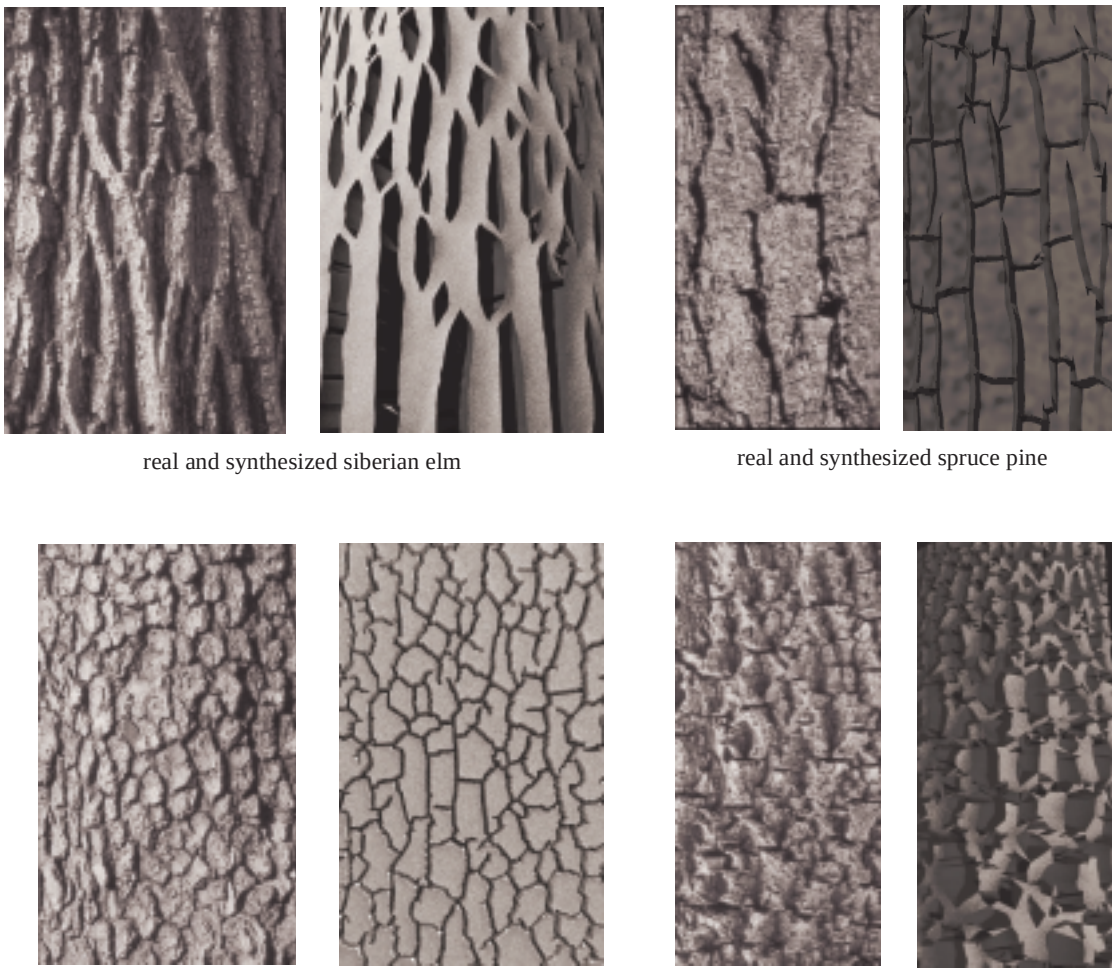



**FIGURE 5.34:** *Real bark compared to synthesized bark.*

| Pattern | Young modulus [MPa] | Poisson ratio | threshold stress [kPa] | Fracture toughness [J] | Rate of growth of the radius | Shrinkage rate top | Shrinkage rate bottom | Thickness at top (initial/ final) [cm] | Thickness at bottom (initial/final) [cm] |
|---|---|---|---|---|---|---|---|---|---|
| Fig. 5.33a | 10000 | 0.3 | 200 | 1000 | 1.0 | 0 | -0.5 | 0.5/10.0 | 1.0/20.0 |
| Fig. 5.33b | 10000 | 0.3 | 100 | 700 | 0.5 | 0.1 | -0.2 | 0.25/1.5 | 0.5/2.5 |
| Fig. 5.33c | 10000 | 0.3 | 200 | 1400 | 0.27 | 0 | 0 | 0.03/0.2 | 0.06/0.4 |
| Fig. 5.33d | 10000 | 0.3 | 200 | 1400 | 0.3 | 0 | 0 | 0.03/0.3 | 0.06/0.6 |
| Fig. 5.33e | 10000 | 0.3 | 100 | 300 | 1.0 | 0 | -0.5 | 1.0/5.0 | 2.0/10.0 |
| Fig. 5.33f | 10000 | 0.3 | 100 | 700 | 0.43 | 0 | 0 | 0.2/1.0 | 0.4/2.0 |
| Fig. 5.33g | 10000 | 0.3 | 1000 | 14000 | 1.0 | 0 | 0 | 0.3/0.3 | 0.6/0.6 |
| Fig. 5.33h | 10000 | 0.3 | 200 | 1400 | 0.67 | 0 | 0 | 0.7/0.7 | 0.7/0.7 |
| Fig. 5.33i | 10000 | 0.3 | 100 | 700 | 1.0 | 0 | -0.5 | 1.0/5.0 | 2.0/10.0 |
| Fig. 5.33j | 10000 | 0.3 | 100 | 700 | 1.0 | 0 | -0.5 | 1.0/10.0 | 1.0/10.0 |
| Fig. 5.33k | 10000 | 0.3 | 200 | 1400 | 1.0 | 0.5 | 0.5 | 1.0/1.0 | 2.0/2.0 |
| Fig. 5.33l | 10000 | 0.3 | 200 | 1400 | 1.0 | 0 | -0.5 | 1.0/1.0 | 1.0/1.0 |
| Fig. 5.33m | 10000 | 0.3 | 200 | 2800 | 1.0 | 0 | -0.5 | 1.0/1.0 | 1.0/1.0 |
| Fig. 5.33n | 10000 | 0.3 | 100 | 1200 | 1.0 | 0 | 0 | 5.0/5.0 | 5.0/5.0 |

**TABLE 5.2:** *Simulation parameters used to generate the bark-like fracture patterns.*

# CHAPTER 6 *Conclusions and Future Work*

The objective of my research was to provide a firm foundation for simulating growth on surfaces and the subsequent formation of fracture patterns. The motivation was to develop a scientific tool for conducting what-if type experiments, which could help us achieve a better understanding of the processes involved in fracture formation on growing surfaces. A related motivation was to provide a new procedural texture synthesizer for the purposes of computer graphics.

In my work, I considered fracture formation on differentially growing, bi-layered surfaces. The top layer, called the material layer, was assumed to grow slower than the bottom, background layer. Through the attachment of the material layer to the background layer, such differential growth resulted in the development of stresses in the material layer, leading to subsequent fracture formation. I described two different, physically based approaches for modeling fracture formation. The first approach was based on a mass-spring model, while the second was based on a finite element method. The modeled surfaces included drying mud and tree bark.

My mass-spring based model for simulating fracture formation was implemented by extending the Skjeltorp and Meakin mass-spring model [61], to which I introduced growth of the background layer. I demonstrated that this approach is prone to artifacts, because

the fractures exhibit a strong tendency to align themselves with the underlying mesh. This undesired directionality of fractures was particularly visible when anisotropic growth was simulated. Furthermore, the relationship of the simulation parameters to real physical properties of materials is not clear. This prevents a proper discretization of the continuum and results in patterns being dependent on both the orientation of the mesh and the level of discretization. Consequently, I abandoned the mass-spring based model in favor of the more robust finite-element based technique, which represents the continuous properties of the material layer much more precisely.

The second approach I presented for modeling fracture formation was thus based on solid mechanics simulated using finite element methods. I incorporated growth into the framework of finite element methods, and to the best of my knowledge, my attempt to use finite element methods to model growing structures is the first of its kind. I modeled the growth of the underlying background by considering global pre-strain. This was accomplished by changing the boundary conditions of the system to reflect the changing positions of the nodes attached to the background layer. The trajectories of the attachment points were specified by their initial position and an appropriate velocity vector field. I used the mathematics of growth tensors to determine what these trajectories should be, as well as to verify that the properties of the velocity vector fields corresponded to the desired types of growth. I considered both isogonic and uniform anisotropic growth of the background layer on planar, cylindrical and spherical surfaces. Shrinkage, or negative growth, of the material layer was modeled by adjusting the reference shapes of the wedge elements, introducing elemental pre-strain. The wedges were allowed to deform their reference shapes unevenly, representing different rates of shrinkage of the material layer near the top and bottom.

I described and implemented two different techniques for introducing fractures into a model. In the first technique, fractures were modeled by removing elements that exceeded

the threshold stress of the material. This technique was simpler to implement and produced convincing results when the sizes of elements around fractures were kept small. Nevertheless, the resulting fractures often exhibited aliasing artifacts, which had negative impact on the appearance of the results.

The second technique introduced fractures into the model by splitting the elements along computed fracture planes. This method of modeling fractures is superior to that of element removal, because no material is actually removed and therefore its mass is preserved. The opening of a new fracture was determined by comparing the computed nodal stress to the threshold stress of the material. The direction of propagation of a fracture was established by computing the principal stresses at the fracture tip, which were in turn used to calculate the fracture plane. The fracture was geometrically modeled by cutting the elements adjacent to the fracture tip node along the computed fracture plane, based on the approach used by O'Brien and Hodgins [45]. The closure of an existing fracture was determined by comparing the amount of energy released by the fracture to the amount of energy that would be required to propagate it. This method of crack-closure correctly terminates a fracture independently of the element sizes. Although this technique for modeling fractures by element splitting was substantially more involved than the element-removal-based method, the resulting fractures were alias-free and therefore visually more appealing.

I developed and implemented a number of efficiency and quality enhancing techniques in the finite-element-based model of fractures. I introduced an adaptive mesh refinement around fracture tips, which reduced the total number of elements required and therefore decreased the space and time requirements of the simulations. I also described and implemented the construction of a temporary local multi-resolution mesh around the fracture tips, which was used to calculate stresses at fracture tips with increased accuracy but without introducing any additional elements permanently into the model. The use of

the multi-resolution mesh helped in keeping the number of elements low. Further, I showed how the equilibrium state of the model can be efficiently recalculated after a local change, such as fracture or local refinement, is introduced into the model. To this end, I recalculated the solution only in the area(s) affected by the local change and I provided a general method for automatic determination of such areas. This technique of local recalculation of the solution was described independently of the notion of fractures, and could be therefore applied to other problems involving introduction of local changes in the model. To automatically and efficiently determine the next optimal time step when applying the growth to the model, I described and implemented an adaptive time step control. In addition, I suggest a number of techniques for maintaining a good mesh quality around fracture tips, which is essential for achieving acceptable accuracy of the results. First, by repositioning the nodes during element splitting, the formation of degenerate elements was avoided when possible. Second, an angle-based mesh-smoothing algorithm was used to locally improve the shapes of elements around fracture tips. Finally, an edge collapsing technique and subsequent Delaunay re-triangulation were applied to remove remaining degenerate elements.

Future research is still needed in a number of areas. For example, I model the surfaces using only two layers - the material layer and the background layer. This has the limitation that the generated fractures are always as deep as the height of the material layer. In reality, however, the depth of neighboring fractures is not necessarily the same, and the depth along a single fracture may even vary. In order to extend my model to accommodate for varying crack depths, multiple material layers should be considered. This could be achieved, for example, by modeling multiple, stacked material layers, so that fractures could propagate both through and parallel to the surface.

In my simulations I reduced the number of elements required by avoiding global mesh refinement. Instead, I used a dynamic approach, refining the mesh only around the fracture

tips. This design can be still improved upon. As the fracture propagates, the mesh remains refined along the entire fracture. Unless the fracture curves rapidly, these small elements are no longer needed. An automatic method for dynamic de-refinement could be devised, which would replace these small elements by bigger ones, resulting in improved efficiency of the simulations. Another promising approach to this problem may lie in the recently published work of Grinspun et. al [21], where the authors introduce an adaptive, hierarchical refinement method for finite element based simulations. Instead of refining the actual elements in the areas of interest and need, the authors suggest a different approach: they keep the elements, but hierarchically refine the base functions inside the elements. Their method is independent of domain dimension, element type as well as basis function order. I feel it would be worthwhile to examine whether their approach could be adopted to modeling fracture formation.

It is known from elastic fracture mechanics that the stresses approach infinity near the crack tip [1]. If the elements are very small, this leads to elemental and nodal stresses around the fracture tips being higher than the real stresses would be. In reality, the threshold stress of a material is never exceeded. The fracture either extends before this happens, or the material in the neighborhood of the fracture tip (the plasticity zone) is plastically deformed. In my simulations I determine the condition for fracture opening by comparing the nodal stress to the threshold stress. If the issue of plasticity is ignored and the elements are sufficiently small, this results in the fracture propagation even after it has been terminated. Currently, I avoid this unwanted fracture propagation by artificially defining a zone around a closed fracture tip, where new fractures are prohibited to initiate. A physically more sound approach would be to extend the model by considering the plasticity of the material, such as presented recently by O'Brien et al. [47].

In the current implementation, I only consider a 6-node wedge finite element, with linear interpolation functions. As a result, the fracture surfaces are approximated by linearly

interpolated rectangular walls. A swiftly bending fracture therefore requires a large number of small elements to correctly represent its surface. An alternative approach would be to consider higher order elements, such as the 18-node wedge element, or an element whose interpolation functions are hierarchically refined (Grinspun et al. [21]). The main benefit of higher order elements is that fewer elements are required to correctly represent a bending fracture. Also, and perhaps more importantly, the stresses in higher order elements are more correctly approximated. The use of higher order elements would then lead to a smaller number of elements to correctly approximate the rapidly changing stress distribution near fracture tips. The disadvantage of using higher order elements is the added computational costs. It would be worthwhile to investigate whether the higher order elements would afford more efficient simulations.

The differential growth of surfaces leads to the formation of cracks, which is often accompanied by a peeling effect. Once a patch of the material surface has been detached from the surrounding material, its edges may also detach from the underlying background, and then curl upward. This curling phenomenon can be observed in some types of tree bark, and occasionally in drying mud. In the current implementation, the fractures are only allowed to initiate at nodes on top of the material layer, and can only propagate along the surface. To simulate the curling effect, the existing model should be extended to allow fractures to be initiated from nodes attached to the background layer, and to propagate parallel to the surface. Combining this extension with the multi-layer extension suggested above could further improve both the correctness and the realism of the results. Abstracting even further, the differential growth of surfaces can also lead to buckling. The buckling effects resulting from differential growth have been studied by Dimian [9] and Matthews [34]. In my future work I plan to combine their work with the fracture model presented here.

As it is difficult to predict which simulation parameters need to be changed in order to obtain a different pattern, only a small collection of drying mud and tree bark fracture patterns have been generated so far. However, a framework has been set up for further exploration of the parameter space, which could determine the effects of various simulation on the generated pattern. A better understanding of the parameter space would not only allow for a wider selection of generated patterns, but also enhance our understanding of nature.

Another area of improvement is the simulation of anisotropic material properties. In the current implementation, I assume isotropic properties of the materials modeled. Wood, however, is an anisotropic material. It has different material properties in different directions, such as Young modulus, Poisson ratio, threshold stress and even fracture toughness [15]. To simulate the behavior of wood under stress more correctly, anisotropic properties of materials should be considered.

The goal of my research was to provide a firm basis for simulating fractures on growing surfaces, which I accomplished in my finite element based model. I provided both computational and simulation methodologies for modeling growth, and demonstrated its potential by applying it to generate convincing crack patterns in drying mud and in tree bark. The work presented here sets the stage for future, more detailed study of bark formation and even studies of growth and pattern formation in general.

# *References*

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

[2] Arnold C. A. 1947. *An Introduction to Paleobotany.* McGraw-Hill Book Co. New York.

[3] Ball P. *The Self-made Tapestry: Pattern Formation in Nature*. Oxford University Press, 1999.

[4] Bastian P., Lang S., Eckstein K. Parallel Adaptive Multigrid Methods in Plane Linear Elasticity Problems. *Numerical Linear Algebra with Applications*, Vol 1(1), 1-1, 1996.

[5] Clough R. W. The Finite Element Method in Plane Stress Analysis. *Proceedings of 2nd ASCE Conference on Electronic Computation*, Pittsburgh, PA, September 1960.

[6] Courant R. Variational Methods for the Solutions of Problems of Equilibrium and Vibrations. *Bull. Am. Math. Soc.*, vol. 49, pp. 1–23, 1943.

[7] Crampin E. J., Gaffney E. A. and Maini P. K. Reaction and Diffusion on Growing Domains: Scenarios for Robust Pattern Formation. *Bulletin of Mathematical Biology,* 61:1093-1120, 1999.

[8] Desbrun M., Schroder P. and Barr A. Interactive animation of structured deformable objects. *Graphics Interface* 1999.

[9] Dimian D. A Physically-Based Model of Folded Surfaces with an Application to Plant Leaves. MSc Thesis, University of Calgary, 1997.

[10] Ebert D. S. at al. *Texturing and modeling: a procedural approach*. Academic Press, inc. 1994.

[11] Erickson R. O. *Ann. Rev. Plant Physiol*. 27:407, 1976.

[12] Erickson R. O. and Silk W. K. The kinematics of plant growth. *Scientific American,* 242(5):134-151, 1980.

[13] Federl P., Prusinkiewicz P. A Texture Model for Cracked Surfaces, with an Application to Tree Bark. *Proceedings of the Seventh Western Computer Graphics Symposium*, 1996.

[14] Federl P. and Prusinkiewicz P. Modelling fracture formation in bi-layered materials, with applications to tree bark and drying mud. *Proceedings of the Thirteenth Western Computer Graphics Symposium*, 2002.

[15] Forest Products Laboratory. *Wood handbook -- Wood as an engineering material*. Gen. Tech. Rep. FPL-GTR-113. Madison, WI: U.S. Department of Agriculture, Forest Service, Forest Products Laboratory, 1999.

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

[17] Gobron S. and Chiba N. Crack pattern simulation based on 3D surface cellular automata. *The Visual Computer*, 17(5):287–309, 2001.

[18] Greenstadt J. On the Reduction of Continuous Problems to Discrete Form. *IBM J. Res. Dev.*, Vol. 3, pp. 355–363, 1959.

[19] Gresho P. M. and Sani R. L. *Incompressible Flow and the Finite Element Method*, Volume 1, Wiley, April 2000.

[20] Griffith A. A. The phenomena of rupture and flow in solids. *Philosophical Transactions*, 221:163–198, 1920.

[21] Grinspun E., Krysl P. and Schroder P. CHARMS: A Simple Framework for Adaptive Simulation. *SIGGRAPH'02*, 2002.

[22] Heckbert P. S. Fast Surface Particle Repulsion. New Frontiers in Modeling and Texturing course, *SIGGRAPH'97*, 1997.

[23] Hejnowicz Z. and Romberger J. Growth Tensor of Plant Organs. *Journal of theoretical biology*, 110:93-114, 1984.

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

[25] Hirota K., Tanoue Y. and Kaneko T. Simulation of three dimensional cracks. *The Visual Computer*, 16:371-378, 2000.

[26] Hoppe H. Progressive Meshes. *Computer Graphics*, vol. 30:99-108, 1996.

[27] Huxley J. S., Needham J. and Lerner I. M. *Nature*, 148:225, 1941.

[28] Inglis C. E. Stresses in a plate due to the presence of cracks and sharp corners. *Transactions of the Institute of Naval Architects*, 55:219–241, 1913.

[29] Kajiya J. and Kay T. Rendering Fur with Three Dimensional Textures. *SIGGRAPH '89*, 1989.

[30] Keeve E., Girod S., Pfeifle P, Girod B. Anatomy-Based Facial Tissue Modeling Using the Finite Element Method. *Proceedings of Visualization'96*, 1996.

[31] Lefebvre S. and Neyret F. Synthesizing bark. *Proceedings of the Thirteenth Eurographics Workshop on Rendering*, 2002.

[32] Lewis R. W., Morgan K., Thomas H. R., Seetharamu K. N. *The Finite Element Method in Heat Transfer Analysis*. Wiley, April 1996.

[33] Lloyd S. Least Square Quantization in PCM. IEEE Transactions on Information Theory. 28:129-137, 1982.

[34] Matthews M. J. Physically Based Simulation of Growing Surfaces. MSc Thesis, University of Calgary, 2002.

[35] Mazarak O., Martins C. and Amanatides J. Animating exploding objects. *In Graphics Interface'99*, June 1999.

[36] McCool M. and Fiume E. Hierarchical poisson disk sampling distributions. Graphics Interface '92, 94-105, May 1992.

[37] Meinhardt H., Prusinkiewicz P. and Fowler D. *The Algorithmic Beauty of Sea Shells*. Springer Verlag, 2nd edition, 1998.

[38] Metaxas D., Terzopoulos D. Dynamic Deformation Of Solid Primitives with Constraints. *SIGGRAPH'92*, 1992.

[39] Morse P. M. and Feshback H. *Methods of Theoretical Physics*. McGraw-Hill, New York, , Section 9.4, 1953.

[40] Murray J. D. *Mathematical Biology.* Springer-Verlag, 1989.

[41] Nakielski J. Tensorial Model for Growth and Cell Division in the Shoot Apex. *Pattern Formation in Biology, Vision and Dynamics*. Edited by Carbone A., Gromov M. and Prusinkiewicz P., World Scientific Publishing, 1999.

[42] Neff M. and Fiume E. A visual model for blast waves and fracture. *Graphics Inteface'99*, June 1999.

[43] Neyret F. Modeling, Animating and Rendering Complex Scenes using Volumetric Textures. IEEE Transactions on Visualization and Computer Graphics, 4(1):55-70, 1998.

[44] Norton A., Turk G., Bacon B., Gerth J. and Sweeney P. Animation of fracture by physical modeling. The Visual Computer (1991) 7:210-219. Springer-Verlag 1991.

[45] O'Brien J. F., Hodgins J. K. Graphical Modeling and Animation of Brittle Fracture. Proceedings of ACM SIGGRAPH '99, 1999.

[46] O'Brien J. F. Graphical Modeling and Animation of Fracture. PhD Thesis. Georgia Institute of Technology, 2000.

[47] O'Brien J. F., Bargteil A. W., Hodgins J. K. Graphical Modeling and Animation of Ductile Fracture. Proceedings of ACM SIGGRAPH'2002, 2002.

[48] Paquette E. The Simulation of Paint Cracking and Pealing. Proceedings of Graphics Interface 2002.

[49] Peachey D. R. Solid Texturing of Complex Surfaces. Proceedings of ACM SIGGRAPH '85, vol. 19, no. 3, pages 279-286, 1985.

[50] Perlin K. An Image Synthesizer. Proceedings of ACM SIGGRAPH ‘85, vol. 19, no. 3, pages 287-296, 1985.

[51] Preparata F. P., Shamos M. I. Computational geometry : an introduction. Springer-Verlag, 1985. ISBN 0387961313.

[52] Press W. H., Teukolsky S. A., Wetterling W. T., Flannery B. P. Numerical recipes in C: the art of scientific computing. Second edition. Cambridge University Press.

[53] Prusinkiewicz P. In Search of the Right Abstraction: The Synergy Between Art, Science, and Information Technology in the Modeling of Natural Phenomena. Art @ Science. Springer-Verlag/Wien, 1998.

[54] Prusinkiewicz P., Lindenmeyer A. Algorithmic Beauty of Plants. Springer-Verlag 1996.

[55] Richards O. W., Kavanagh A. J. The Analysis of Relative Growth Gradients and Changing Form of Growing Organisms: Illustrated by the Tobacco Leaf. Amer. Nat. 77:385-399, 1943.

[56] Richards O. W. and Kavanagh A. J. In: Essays in Growth and Form (E.E. leGros Clark and P. B. Medawar eds). Oxford: Clarendon Press, 1945.

[57] Richards O. W. and Riley G. A. Journal of Experimental Zoology, 77(159), 1937.

[58] Rivara M. and Inostroza P. Using Longest-side Bisection Techniques for the Automatic Refinement of Delaunay Triangulations. *The 4th International Meshing Roundtable*, Sandia National Laboratories, pp.335-346, October 1995.

[59] Romberger J. A., Hejnowicz Z. and Hill J. F. *Plant Structure: Function and Development*. Springer-Verlag,1993.

[60] Silk W.K. and Erickson R.O. Kinematics of plant growth. *Journal of Theoretical Biology*, 76:481-501, 1979.

[61] Skjeltorp A. T., Meakin P. Fracture in Microsphere Monolayers Studied by Experiment and Computer Simulation. *Nature*, 335:424-426, Sept. 1988.

[62] Smith J., Witkin A. and Baraff D. Fast and Controllable Simulation of the Shattering of Brittle Objects. *Computer graphics forum*, 20(2):81-91, Blackwell Publishing, 2001.

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

[64] Terzopoulos D. and Fleischer K. Modeling inelastic deformation: Viscoelasticity, plasticity, fracture. *SIGGRAPH'88*, 1988.

[65] Thompson D'A. W. On Growth and Form. *Trans. Roy. Soc. Edinburgh*, 50:857-895, 1915.

[66] Thompson D'A. W. *Growth and Form*. Cambridge, 1917.

[67] Turing A. The chemical basis of morphogenesis. *Phil. Trans. R. Soc. London*, 1952.

[68] Turk G. Generating Textures on Arbitrary Surfaces Using Reaction-Diffusion. *Computer Graphics*, 25(4):289-298, 1991.

[69] Witkin A. P. and Heckbert P. A. Using particles to sample and control implicit surfaces. *SIGGRAPH'94*, pages 269–277, July 1994.

[70] Witkin A. and Kass M. Reaction-Diffusion Textures. *Computer Graphics*, 25(4):299-308, 1991.

[71] Wyvill G., McPheeters C. and Wyvill B. Animating soft objects. *Visual Computer*, 2:235-242, 1986.

[72] Zhou T. and Shimada K. An Angle-Based Approach to Two-Dimensional Mesh Smoothing. *The 9th International Meshing Roundtable*, pp.373-84, 2000.

[73] Zienkiewicz O. C. and Cheung Y. K. Finite Elements in the Solution of Field Problems. *Engineer*, Vol. 220, pp. 507–510, 1965.

[74] Zienkiewicz O. C. and Taylor R. L. *Finite element method: Volume 2 - Solid Mechanics*. Butterworth Heinemann, London, 2000.

[75] Zienkiewicz O. C. and Zhu J. Z. The superconvergent patch recovery and a posteriori error estimates. Part 1: The recovery technique. *International Journal for Numerical Methods in Engineering*, 33:1331-1364, 1992.

# APPENDIX A *Fracture formation in 1D using mass-spring systems*

In an effort to address the self-similarity issue when modeling fracture formation using mass-spring models, I developed an equivalent model in one dimension. The main objective of this simplified 1D model was to provide more insight into the processes taking place. The 1D model I adopted is a simplification of the 2D model already discussed in Chapter 4. For the reader's convenience, I will briefly recall this model:

- I am modeling a segment of an elastic surface attached to an expanding background. If breaking is ignored, the surface segment has a spring-like behavior, obeying Hooke's law.
- The connection between the surface and the background is elastic, approximated by simulating anchor springs interconnecting each segment of the surface with the background. The physical properties of an anchor spring are related to the size of the segment to which it is attached, so that the force necessary to move the entire segment is directly proportional to that segment's size.
- The material breaks at a point where a threshold tension of the material is exceeded.

Two different approaches of the 1D model were developed: the discrete model, solved numerically, and the continuous model, solved analytically. The discrete model assumes the material is composed of a finite number of connected sub-segments, whose behavior must be simulated at small time intervals. In the continuous model, the surface is assumed to be a continuous medium, for which I developed a mathematical description. This mathematical model can predict the behavior of the surface at any time instance, without any need for a time-based simulation. The next two sections describe these two implementations in detail.

# A.1. Discrete model

The surface segment is discretized into sub-segments of equal sizes, each with spring-like behavior (see Figure A.1). Every sub-segment is attached to the background by two anchor springs, located at the sub-segments' endpoints. There are three sources of force acting on each sub-segment's endpoint: the left spring (simulating left sub-segment), the right spring (simulating right sub-segment) and the anchor spring. When the material segment is in an equilibrium state, the sum of the forces acting at each endpoint has to be equal to zero. The equation describing this equilibrium state can be expressed as:

*Eq. A.1*

$$-\frac{K_S}{h}[p(x_i)-p(x_{i-1})-h]-\frac{K_S}{h}[p(x_i)-p(x_{i+1})+h]-K_A h[p(x_i)-a(x_i)] = 0 \quad 0<x<l_0$$

where $i$ identifies the node, $x_i$ identifies its coordinate in the reference state, $p(x_i)$ denotes the node's current coordinate and $a(x_i)$ represents the coordinate of the node's anchor point. $K_S$ denotes the stiffness coefficient of the material spring when its rest length is one unit. The stiffness coefficient of a material spring whose length is $h$, is calculated as $K_S/h$. Similarly, $K_A$ denotes the stiffness coefficient of an anchor spring connecting a material segment of unit length to the background. The stiffness coefficient of an anchor spring is

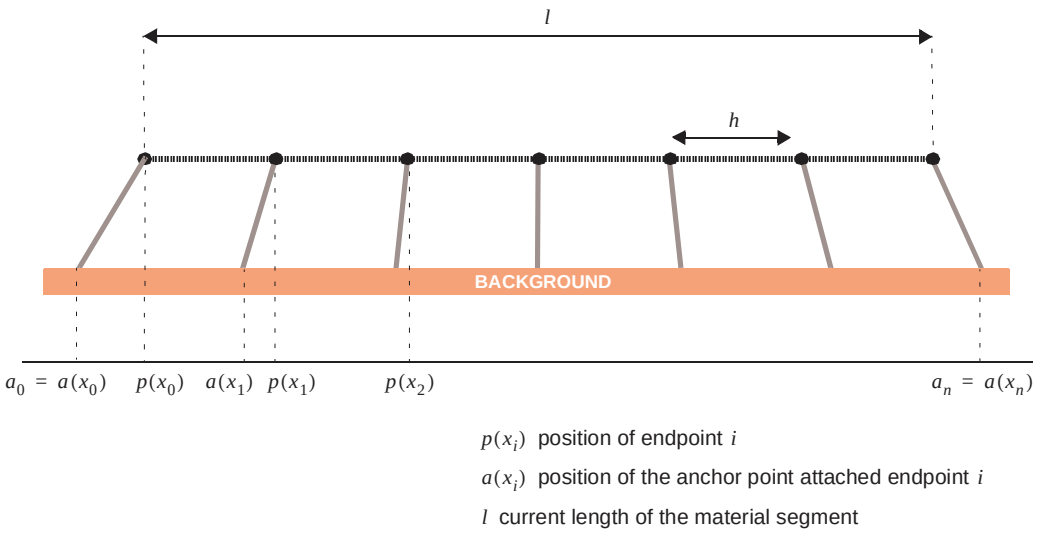



**FIGURE A.1:** *Discrete 1D model.*

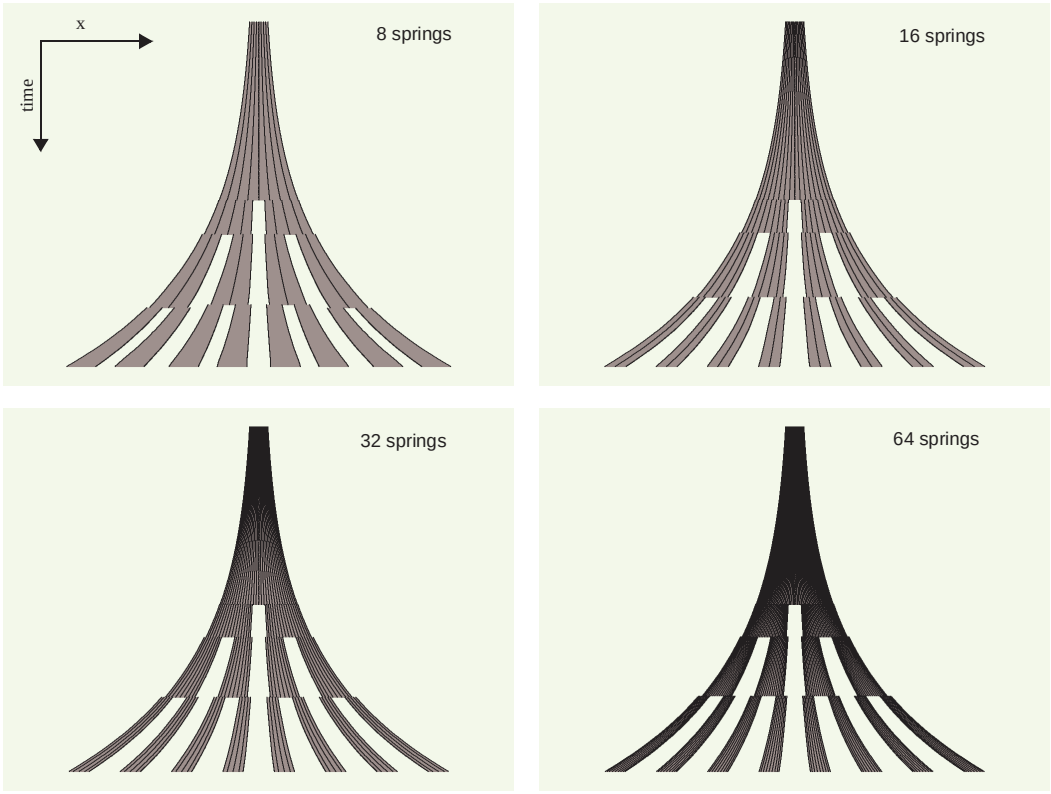
directly proportional to the rest length of the surface segment to which it is attached and is calculated as $hK_A$. Describing the properties of the surface independently of its size allows for proper refinement of the subdivision, as shown in Figure A.2.

Assuming uniform distribution of anchor points we can express the positions of anchor points: $a(x_i) = \frac{x_i}{l_0}(a_n - a_0) + a_0$, where $a_0$ and $a_n$ represent the left-most and right-most anchor point locations, respectively. Further, assuming that all sub-segments have constant rest-lengths, we can write $x_{i-1} = x_i - h$ and $x_{i+1} = x_i + h$. Substituting these into Eq. A.1 and dividing both sides of the resulting equation by $h$ we obtain:

*Eq. A.2*

$$\frac{K_S}{h^2}[2p(x_i) - p(x_i - h) - p(x_i + h)] + K_A\left[p(x_i) - \frac{x_i}{l_0}(a_n - a_0) - a_0\right] = 0$$

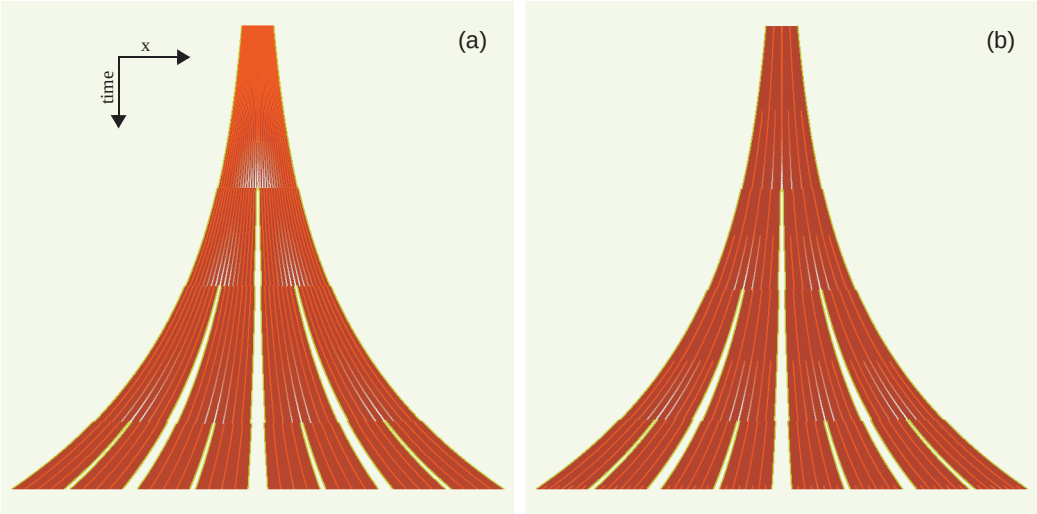
By substituting the known quantities for anchor point locations into Eq. A.2 we can solve for the unknown positions of nodes $p(x_i)$. The growth is therefore effectively simulated by




Legend: *the light gray represents the springs, the black represents the endpoints.*

**FIGURE A.2:** *Fracture formation in 1D for four different levels of subdivision.*

changing the values of the left-most and right-most anchor points, $a_0$ and $a_n$ respectively. Similarly, shrinkage can be simulated by reducing the length of the sub-segments $h$. The sub-segments break when they reach a critical length $l_{break}$, i.e. when $p(x_i) - p(x_{i-1}) > l_{break}$. Such break can be simulated by setting the stiffness coefficient of the broken spring to zero. The results of simulations for different number of sub-segments are shown in Figure A.2. The generated patterns are similar even though different levels of discretization were used.




Fixed subdivision (the pattern does not repeat infinitely because the number of cracks is limited by the fixed number of segments).

Automatic subdivision (the pattern can repeat infinitely).

**FIGURE A.3:** *Recursive patterns.*

By allowing some of the material properties to change over time, it is possible to achieve effects that could not be produced before. For example, by allowing the rest lengths of the surface sub-segments to grow, it is possible to obtain self-repeating patterns (Figure A.3a). Figure A.3b also demonstrates another extension to the model - automatic subdivision refinement. As soon as a sub-segment reaches a certain length, it is replaced by two sub-segments, automatically adjusting the subdivision. Automatic subdivision refinement speeds up the simulation process as less sub-segments and connections need to be modeled at the beginning of the simulation.

## A.2. Continuous model

In the continuos model I assume the material is a continuous medium, i.e. composed of an infinite number of infinitely small sub-segments. In order to convert the discrete model into a continuous one, we manipulate Eq. A.2 by taking its limit as $h$ approaches $0$:

*Eq. A.3*

$$\lim_{h \to 0} \left\{ \frac{K_S}{h^2}[2p(x_i) - p(x_i - h) - p(x_i + h)] + K_A\left[p(x_i) - \frac{x_i}{l_0}(a_n - a_0) - a_0\right] \right\} = 0 \ ,$$

which yields an ordinary differential equation of 2nd degree:

*Eq. A.4*

$$-K_S p''(x) + K_A p(x) - \frac{K_A(a_n - a_0)}{l_0}x - K_A a_0 = 0$$

Solving this ODE (using Mathematica) yields the following general solution:

*Eq. A.5*

$$p(x) = \frac{x(a_n - a_o)}{l_0} + a_0 + C_1 \sinh\left(\sqrt{\frac{K_A}{K_S}}x\right) + C_2 \cosh\left(\sqrt{\frac{K_A}{K_S}}x\right)$$

where the constants $C_1$ and $C_2$ depend on the boundary conditions at the endpoints of a sub-segment. If we perform the following substitution $r = \sqrt{\frac{K_A}{K_S}}$, the equation becomes more readable:

*Eq. A.6*

$$p(x) = \frac{x(a_n - a_o)}{l_0} + a_0 + C_1 \sinh(rx) + C_2 \cosh(rx)$$

There are two types of boundary conditions at each endpoint: the sub-segment is either attached to another sub-segment or it ends freely. These boundary conditions generate 4 pairs of values for $C_1$ and $C_2$. Depending on whether a material segment is connected to other material segments, the 4 types of boundary conditions are:

i. no neighbors,
ii. neighbor on the left, no neighbor on the right,
iii. no neighbor on the left, neighbor on the right,
iv. neighbor on the left, neighbor on the right.

**Boundary condition type I: no neighbors**

When a material segment has no neighbors the boundary conditions can be described by the following equations (for the discrete model):

*Eq. A.7*
$$-\frac{K_S}{h}[p(0)-p(h)+h]-K_A\frac{h}{2}[p(0)-a(0)] = 0$$

*Eq. A.8*
$$-\frac{K_S}{h}[p(l_0)-p(l_0-h)-h]-K_A\frac{h}{2}[p(l_0)-a(l_0)] = 0$$

After applying $\lim_{h\to 0}$ to Equation A.7 and Equation A.8, we obtain two initial conditions:

*Eq. A.9*
$$\lim_{h\to 0}\left\{-\frac{K_S}{h}[p(0)-p(h)+h]-K_A\frac{h}{2}[p(0)-a(0)]\right\} = 0$$

$$K_S\lim_{h\to 0}\left\{\frac{p(h)-p(0)}{h}-1\right\}-\frac{K_A}{2}\lim_{h\to 0}\{hp(0)\}-\frac{K_A}{2}\lim_{h\to 0}\{ha(0)\} = 0$$

$$K_S\lim_{h\to 0}\left\{\frac{p(h)-p(0)}{h}-1\right\} = 0$$

$$K_S[p'(0)-1] = 0$$

$$p'(0) = 1$$

*Eq. A.10*
$$\lim_{h\to 0}\left\{-\frac{K_S}{h}[p(l_0)-p(l_0-h)-h]-K_A\frac{h}{2}[p(l_0)-a(l_0)]\right\} = 0$$

$$-K_S \lim_{h \to 0} \left\{ \frac{p(l_0) - p(l_0 - h)}{h} - 1 \right\} - \frac{K_A}{2} \lim_{h \to 0} \{hp(l_0)\} - \frac{K_A}{2} \lim_{h \to 0} \{ha(l_0)\} = 0$$

$$K_S \lim_{h \to 0} \left\{ \frac{p(l_0 - h) - p(l_0)}{h} - 1 \right\} = 0$$

$$K_S[p'(l_0) - 1] = 0$$

$$p'(l_0) = 1$$

With these boundary conditions, the constants for Equation A.6 are:

*Eq. A.11*

$$C_1 = -\frac{a_n - a_0 - l_0}{rl_0} \text{ and } C_2 = \frac{(a_n - a_0 - l_0)(e^{rl_0} - 1)}{rl_0(e^{rl_0} + 1)}$$

After substituting $d = a_n - a_0 - l_0$ Equation A.11 becomes:

$$C_1 = -\frac{d}{rl_0} \text{ and } \quad C_2 = \frac{d(e^{rl_0} - 1)}{rl_0(e^{rl_0} + 1)} \quad .$$

**Boundary condition type II: left neighbor**

The material segment is connected to its neighbor at a coordinate $Z_L$. The boundary conditions for our ODE now change to:

$$p(0) = Z_L, \quad p'(l_0) = 1$$

The constants for Equation A.6 are:

*Eq. A.12*

$$C_1 = \frac{-d + \sinh(l_0 r) rl_0 a_0 - \sinh(l_0 r) rl_0 Z_L}{\cosh(l_0 r) rl_0} \text{ and } C_2 = Z_L - a_0 \ .$$

**Boundary condition type III: right neighbor**

The the boundary conditions for a material segment with a neighbor on the right side are:

*Eq. A.13*

$$p'(0) = 1, \quad p(l_0) = Z_R$$

The constants for Equation A.6 are:

*Eq. A.14*

$$C_1 = -\frac{d}{l_0 r} \text{ and } C_2 = \frac{a_0 l_0 r + d l_0 r + l_0^2 r - d\sinh(l_0 r) - Z_R l_0 r}{\cosh(l_0 r) r l_0} .$$

**Boundary condition type IV: left and right neighbors**

Finally, the boundary conditions for a material segment with two neighbors are:

*Eq. A.15*

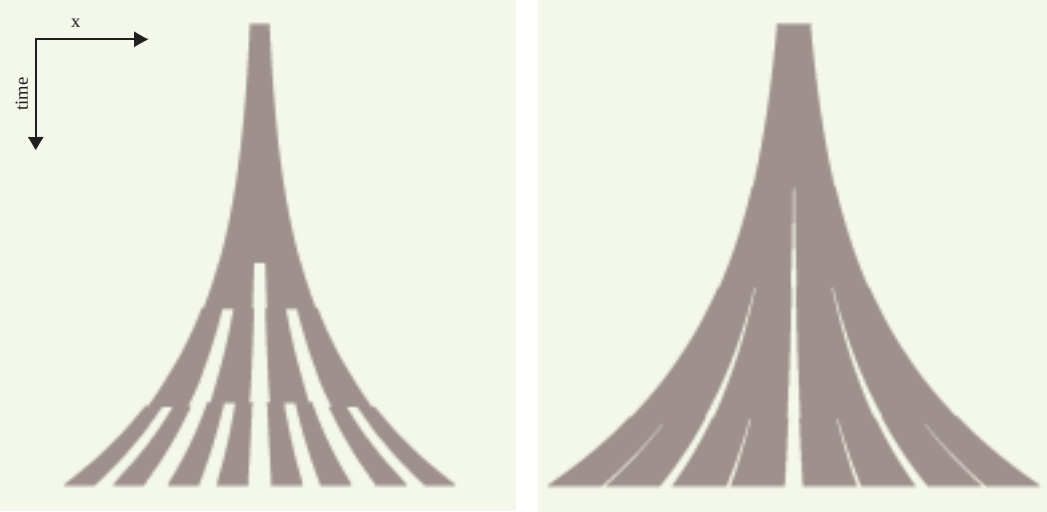
$$p(0) = Z_L, \quad p(l_0) = Z_R$$

The constants for Equation A.6 are:

*Eq. A.16*

$$C_1 = \frac{-a_0 - d - l_0 + \cosh(r l_0) a_0 - \cosh(r l_0) Z_L + Z_R}{\sinh(r l_0)} \text{ and } C_2 = Z_L - a_0 .$$

Equation A.5 represents the heart of the analytical model as it can be used to completely derive the behavior of the sub-segment. For example, the location of maximum tensile stress along a surface sub-segment corresponds to the point of maximum strain, i.e. where $p'(x)$ reaches maximum value. Candidates for such points can be found by solving for $p''(x) = 0$ and $p^{(iii)}(x) > 0$. Similarly, given the rate of growth of the surface, we can determine the precise time at which a threshold stress in a material segment is exceeded.




regular fracture pattern self-similar fracture pattern

**FIGURE A.4:** *Fracture formation in 1D using the continuos model.*

The results produced by this continuous implementation (Figure A.4) match the results obtained using the discrete (simulation-based) implementation from the previous section (Figure A.2). The small differences in the results between these two implementations are due to the level of discretization - which disappear when the discretization is refined. By producing closely matching results, the two models confirm each other.