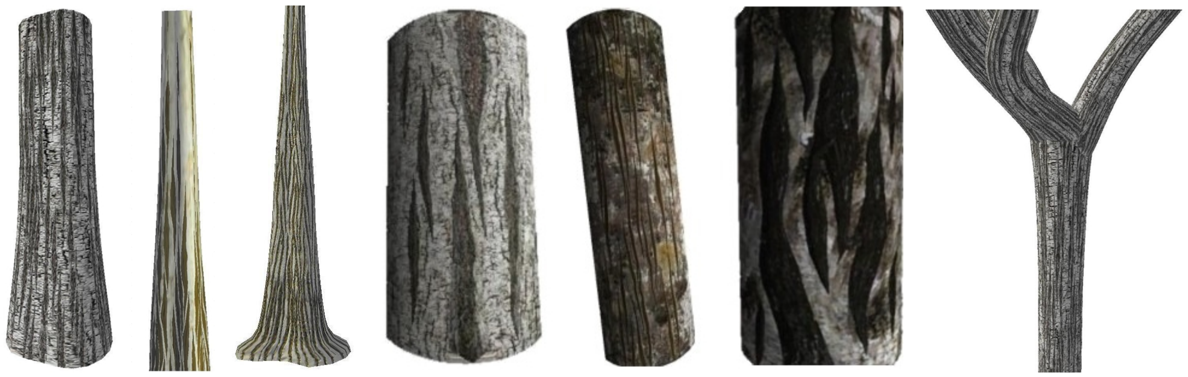
Thirteenth Eurographics Workshop on Rendering (2002)
P. Debevec and S. Gibson (Editors)

Synthesizing Bark

Sylvain Lefebvre
Sylvain.Lefebvre@imag.fr

Fabrice Neyret
Fabrice.Neyret@imag.fr

iMAGIS-GRAVIR/IMAG-INRIA, joint research project of CNRS, INPG, INRIA, UJF



Abstract

Despite the high quality reached by today's CG tree generators, there exists no realistic model for generating the appearance of bark: simple texture maps are generally used, showing obvious flaws if the tree is not entirely painted by an artist. Beyond modeling the appearance of bark, difficulty lies in adapting the bark features to the age of each branch, ensuring continuity between adjacent parts of the tree, and possibly ensuring continuity through time.
We propose a model of bark generation which produces either geometry or texture, and is dedicated to the widespread family of fracture–based bark. *Given that the tree growth is mostly on its circumference, we consider circular strips of bark on which fractures can appear, propagate these fractures to the other strips, and enlarge them with time. Our semi–empirical model runs in interactive time, and allows automatic or influenced bark generation with parameters that are intuitive for the artist. Moreover we can simulate many different instances of the same bark family. In the paper, our generated bark is compared (favourably) to real bark.*

Keywords: *Natural Phenomena, Texture Synthesis, Physically Based Modeling, Tree Rendering.*

1. Introduction

Numerous empirical or botany–inspired tools allow the creation of very realistic tree models [2, 6, 38, 29, 16], and are used for visual effects or impact studies. Recent papers also deal with: ecosystem simulation [7]; influence of lighting conditions [33]; and algorithms dedicated to the rendering of forests [20, 19, 18, 22, 37, 21]. On the other hand, the close–up view is not handled at all: automatic tools use tilable textures to represent the bark, which give unrealistic results on trees (i.e. no continuity at branching, no control of the features, and stretching depending of the branch size). To get good–quality results, special effects artists have to paint the whole surface [10, 5], or have to combine multiple dedicated maps [34] which is a long and tedious task.

These models could be adequate for distant views, but their poor quality becomes obvious when the viewpoint is close to the tree (e.g. namely, closer than a dozen meters). There is a real need in today's applications — such video games, special effects, and landscape management — for a bark model that supports close viewpoints.

Bark is the result of tree growth: The wood dividing tissue (the *phellogen*) is a few millimeters to a few centimeters inside the trunk or branch, producing fresh wood toward the inside and fresh bark toward the outside. Thus the outside layers (i.e. the visible bark) are older than the inside ones. This yields a strong tangential tension resulting in fractures, tears, and others similar mechanical wounds. See Figure 2.

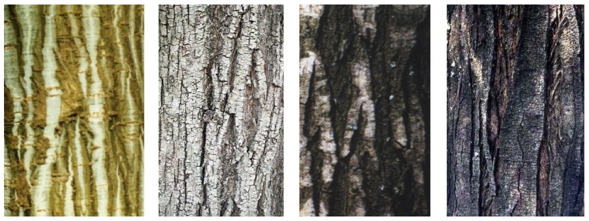


Figure 1: *Fracture–based bark of various tree species (real images).*

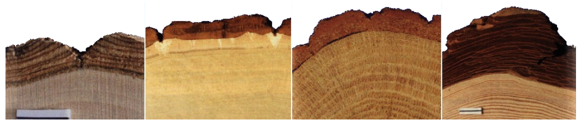


Figure 2: *Slices of trunk of various species, showing the fractured bark.*

To be useful, a bark model should be automatic enough to handle most of the tedious work, but should also allow the artist to control the appearance of the resulting bark. It is not reasonable to develop a completely accurate physical bark simulator, since the number of parameters involved in the underlying biological and mechanical phenomena is enormous. Besides, a completely accurate simulator wouldn't be helpful for the intuitive tuning of the appearance, and a finite element mechanical simulation would lack the interactivity the user needs to tune the parameters.

Our method is physical, empirical, and textural: we rely on crack simulation; we take advantage of bark–related *a priori* knowledge; and we use textures to dress the inside and outside of fractures with details. Our method is based on two complementary evolving structures: circular strips of bark (i.e. transverse to the branch or trunk), and axial cracks (i.e. parallel to the branch or trunk). We simulate in one dimension (1D) the fracture creation and enlargement caused by the tension along a strip, and we simulate in the orthogonal direction the propagation of fractures, which is the only coupling between strips. This scheme is based on the assumption that growth is oriented and that the fracturing bark layer has no influence on the growth "force field", which are valid assumptions for trees.

Once we know the location and width of fractures, we move apart the *epidermis* on each side of the fracture, keeping their textures attached, and we introduce a piece of fracture texture between the two edges of the fracture. Depending on the needs of the application, we can either insert the fracture geometry into the tree mesh, or we can generate a flat bark texture. This model of fracture is continuous in time (because of its manner of construction) which allows us to visualize the growth of a tree.

An important point is that our 1D strips are geometrical, rather than textural. This prevents distortion and discontinuities within fractures, two very common flaws that occur with texturing techniques. However, it happens that — contrary to intuition — real bark *doesn't* show a total continuity of the feature at the branching areas, so our continuity constraints are only partial.

The contributions of the paper are:

- the first complete model of bark synthesis (to our knowledge) whose main visual features are continuous in both space and time. It generates either texture or geometry, and allows both interactive or automatic generation.
- a partly decoupled scheme allowing the efficient. simulation of mechanics, which is extendible to other applications.
- the use of 1D textural space to exploit partial parameterization without suffering the flaws of a 2D mapping.

The paper is structured as follows: We review in Section 2 the existing work related to texture synthesis, bark and simulation of cracks. We describe the basis of our technique (i.e. flat bark) in Section 3, and we show the corresponding results in 4. Then we extend this model to mappable bark in Section 5, and show the results in 6. We conclude in Section 7.

2. Previous Work

Work on textures

Only classical mapped textures (color, bump, displacement) have been used to handle bark (see next paragraph). Procedural and precomputed textures have not been used to simulate the appearance of bark, despite the fact that these techniques are able to avoid distortions. Problems arise because:

- the bark features are not local (fault lines can be very long);
- conservative laws should apply (the length of the original epidermis in a section is constant);
- appearance varies with the age of the branches, but continuity must exist between branches of difference ages.

Procedural textures, such as Perlin noise [28], can easily handle non–locality (as for veins in marble) and adaptability, but they cannot guarantee any conservative law. Moreover, orienting the features requires a parameterization, and special "events" such as knots cannot easily be taken into account.

Texture resynthesis on the surface, such as Turk proposes [36], is able to reproduce the non–locality and the conservative property (as much as is contained in the reference images). But this family of methods cannot easily handle a progressive change in the parameters (to account for the age of the branch), does not provide control to the artist, and does not allow an interactive tuning of the parameters. Moreover, changes through time are not handled at all. The image quilting approach of Efros and Freeman [8] brings some user control and can handle transitions, but adapting it to three dimensions (and possibly to resynthesis) is not trivial. Furthermore, its method of transition handling does not permit it to manage a continuous variations of parameters (except if the whole tree texture is already available).

Work on bark

Two kinds of work exist in this area: static dressing using textures (i.e. adding details to the geometry using alternative representations, such as textures), and simulation. With static dressing, the main issue is the surface parameterization: variations of the diameter cause distortions, and branching causes discontinuities. Bloomenthal [3] uses an X–ray image of a bark sample to create a bump–map texture. He defined a parameterization of branchings using splines, which did a nice job despite not totally solving the discontinuity and distortion problems. Maritaud *et al.* [17] used displacement maps and proposed a simpler parameterization of branchings. Hart and Baker [11] rely on the parameterization of blended implicit cones. In industry, two classical approaches are direct painting on the surface [10, 5] and blending of multiple maps (overused in the making of “A Bugs Life,” as detailed in [34]).

The second family of approaches, such the one proposed by Federl and Prusinkiewicz [27], rely on mass–spring networks mapped onto the tree surface, where the springs break beyond a constraint threshold. Hirota *et al.* [12] use a multi–layer mass–spring network, extending to bark their paper on cracks [13]. Both methods provide textures with small–scale cracks, but cannot represent open fractures. Moreover, they require a huge number of springs to get interesting results, as details are only created by the simulation.

Work on cracks in Computer Graphics

Several recent papers in Computer Graphics deal with the simulation of cracks. Terzopoulos *et al.* introduced [35] an early representation of fractures together with a model of elastic, plastic, and visco–elastic continuous material. Fractures are represented as discontinuities in the model. Norton *et al.* [25] proposed an approach to represent solid objects as cubes linked by springs. In the same spirit, Smith *et al.* proposed [32] a method relying on the tetrahedrization of solid objects. In both of these works, the cracks follow the element boundaries, thus requiring a very high resolution to make this bias invisible. Muller *et al.* [23] extended the principle to real–time simulation, building fractures at impact events and simulating rigid objects the rest of the time. As in previous methods, fractures have to follow tetrahedron faces. In this case, the resolution cannot be increased without losing real–time computation, so the bias is very visible in the results. O’Brien and Hodgins [26] rely on finite elements to determine the origin and direction of cracks. This gives very nice results at the price of a huge computational cost, since a very small time step has to be used. Neff *et al.* [24] used a procedural model of brittle fracture pattern to break a window and simulate its shattering under the effect of a blast wave. The procedural model of fracturation avoids the huge computational cost of a physical simulation. However, it was only proposed for a flat object.

Note that these works are of interest concerning the appearance of cracks, but they do not model the interaction of a fractured bark layer with the substrate as it occurs in and around the fractures in the bark.

The physics of fractures

A break occurs in a material when the internal constraints exceed a strength threshold. The molecular link is then broken and a crack starts. It propagates orthogonally to the direction of maximal stress, as long as the system doesn’t come back to a steady state. Imperfections in the material have a strong influence on its fracture behavior. Two kind of fracture can be distinguished:

- The *fragile fracture* needs little energy and propagates quickly in the material. It usually results in the breaking of the object. Its study is the mechanics of elastic fracture (see [4, 1, 35] for more details on this topic).
- The *ductile fracture* shows a plastic deformation area at the propagation ends. The energy dissipates into deformation, thus damping the fracture.

The two forms of fractures are linked: A ductile material can show fragile fractures in cases of intense and short stress.

Two approaches exist to model the fracture of elastic materials. The Inglis approach [14] is based on strain, and aims to evaluate the stress intensity at the fracture ends. The strain is a function of the fracture length–to–width ratio. The Griffith approach [9] defines the fracture limit condition as the time when the fracture increases under equilibrium: A fault in a stressed layer increases when enough potential energy is stored to allow the creation of the new surfaces. The study of ductile fractures is difficult and not in the scope of this paper, so won’t be reviewed here.

3. Our Base Representation

As we stated in the Introduction, physical simulation of bark is not reasonable due to the complexity of the material and the variations of its parameters. Our approach consists of characterizing the phenomenological properties of bark, and simulating a simpler material having the same visual characteristics. We study the characteristics of the appearance in Section 3.1, which provides us with the hypothesis used in our model. We explain the basis of our model in Section 3.2, and we describe how to dress it in Section 3.3. We extend this model in Section 5 in order to take into account the sticking to the substrate, the heterogeneous growth, and the mapping on a tree (including branchings).

3.1. Case Study and Hypothesis

The study of trees provides the following observations:

- the original epidermis is conserved;
- large, similar–looking fractures result from tree growth;
- the bark layer is affected by tree growth, but has no retroactive effect on it;
- the growth is strongly oriented (radially), as are the fractures (longitudinally);
- the material is elastic in the short term (thus fracturing occurs), and plastic at long term (thus no tension remains and the state is always at equilibrium);
- fractures influence each other (see Figure 4);
- small strips appear between close fractures (see Figure 4);
- fractures are torn–open wounds whose edges can have various appearances;
- several scales of fractures can exist (e.g. in some species the surface of old fractures can behave like epidermis).

This drives the choices of our model:

- we consider longitudinal slices of bark that are orthogonally crossed by fractures;
- we suppose that the epidermal portions in this slice are quasi–rigid;
- we assume an elastic mode of material fracture;
- we simulate a quasi-static state: The material is affected by series of instantaneous small fractures growing, between which the material is at equilibrium.

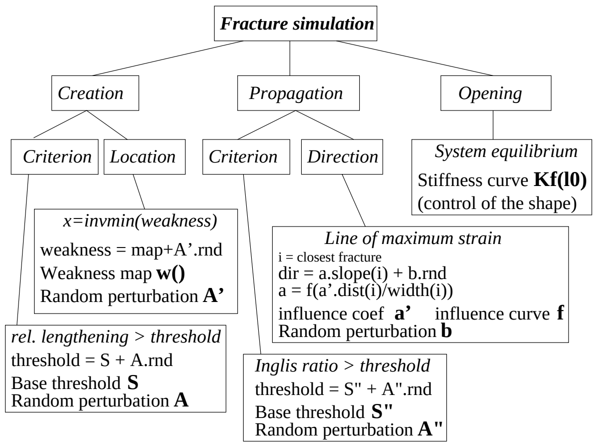


Figure 3: *The aspects of our fracture simulator. All parameters that can be controlled by the user are shown in bold.*

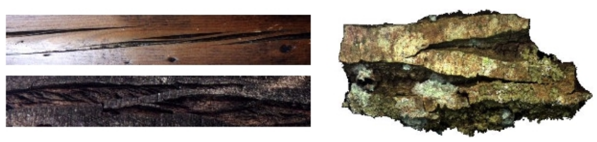


Figure 4: *Fractures tend to interlace (real images).*

3.2. Our Bark Model

3.2.1. Model of Fractures

Three phenomena occur in our material:

- the appearance of new cracks in the bark;
- the propagation of cracks;
- the opening of existing fractures.

Our purpose is to simulate these three phenomena using either physical or phenomenological behavioral laws, while keeping controls for the user. Figure 3 provides a synopsis of our fracture simulation, which we detail in the following paragraphs. The algorithm is given in Figure 8 and results are shown in Figure 10.

3.2.2. Model of Material

As stated in the Introduction, we model the bark with a set of *strips* parallel to the growing direction as shown in Figure 5. A strip consists of a set of alternating elements: original epidermal elements and fracture elements. The epidermal elements are quasi–rigid, while the fracture elements are soft. When the bark breaks at a given location, we introduce a new fracture element between the tear edges. This splits the fracturing epidermal element into two parts. Since the fractures propagates through strips, the resulting tears (formed by the set of fracture elements) will be orthogonal to the growing direction.

The next growing steps will result in the widening of the soft fracture element. Since the release of the stress is partial (and local, in the model extension of Section 5) the bark may break again in an adjacent strip.

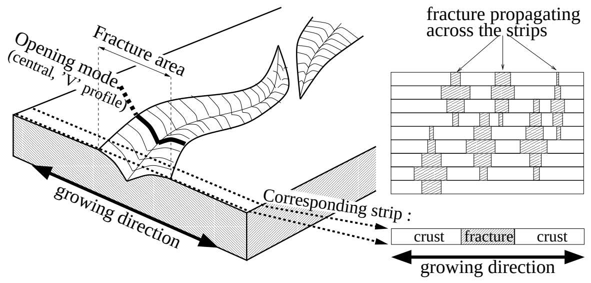


Figure 5: Left: *Characteristics of a tear.* Right*: Strips model.*

Bark stiffness: According to Hooke's law, an elastic element of stiffness K affected by a lengthening e yields a strain $F = Ke$. But one has to distinguish the *material stiffness* and the *element stiffness*: an element of rest length l_0 made of a material of stiffness R shows a global stiffness $K = \frac{R}{l_0}$. When an element is split, the stiffness of each part has to be evaluated as indicated in Figure 6.

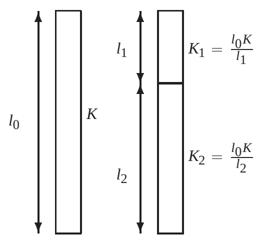


Figure 6: *Stiffness of elastic elements.*

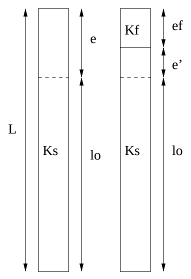


Figure 7: *Fracture criterion evaluation.*

Fracture stiffness: The previous behavior is true for the solid bark, but the fracture element is not a real material and do not obey this law: Its stiffness is linked to the quality of the connection to the substrate and to the shape of the fracture.

We assume the bark's evolution is quasi–static, so the strip has to be at equilibrium: At each time step, the following system of equations representing the strips has to be solved:
$\left\{ \forall i \in [0\ N],\ K_i e_i = K_{i+1} e_{i+1}\ ,\ \ \sum_i e_i = L^{tot} - l_0^{tot} \right\}$
As the system is linear, a standard resolution method can be used. We rely on the bi–conjugate gradient method of the ITL library [15]. Our simulation algorithm is shown in Figure 8.

```
foreach strip
   lengthen the strip
   solve equilibrium
   while an element obeys the fracture criterion
     choose the location of fracture
     break the element
     solve equilibrium
   end while
   propagate fractures
end foreach
```

Figure 8: *Fracture opening and propagation algorithm.*

3.2.3. Fracture creation

The **fracture criterion** is based upon the relative lengthening of elements, which corresponds to the Griffith energetic approach [9]. The threshold S has to be chosen in a reasonable range; otherwise an infinite fracture loop may occur. We describe now how to determine this range.

Figure 7 shows an element of length L, stiffness K_s and rest length l_0 (still after fracture). The element breaks if $\frac{L}{l_0} > S$. This decreases its lengthening from e to e'. The remainder $e_f = e - e'$ corresponds to the fracture width. We don't want it to break again, so we want $\frac{l_0 + e'}{l_0} < S$. Let K_f be the fracture stiffness. Since the system is at equilibrium again, we have $K_s e' = K_f e_f$, thus $e' = e \frac{K_f}{K_s + K_f}$. So we want $S > 1 + \frac{e}{l_0} \frac{K_f}{K_s + K_f}$.

On the other hand, if S is too big no fracture will appear at all. A fracture is guaranteed to occur in the strip if $S < \mathcal{G}$, with $\mathcal{G}$ the growing rate of the tree circumference.

In nature, the fractures do not occur simultaneously, especially for heterogeneous materials. We model this heterogeneity by jittering the fracture threshold. In practice, we use a base criterion equal to the growing rate $\mathcal{G}$ and we only vary the amount of noise. This provides the user with good control over density of fractures.

Fracture location: The bark breaks at its weakest location, which assumes that it is heterogeneous. We model this using both a weakness map and jittering: The jittering encodes the random variations in the material, while the map allows the user to paint weak areas where fractures are more likely to appear. The model is also capable of applying recursive fractures to simulate the behavior of some species of bark for which the fracture surface becomes fresh bark (see Figure 9): a weakness map is created so that new fractures tend to appear into the previously generated fractures.

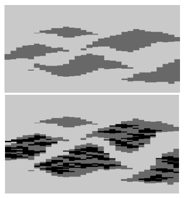


Figure 9: *Refracturation of an old fracture surface.*

3.2.4. Fracture propagation

To propagate a fracture, we have to determine if the fracture element can break the neighboring strip. Our criterion and direction choice are based on the Inglis observations [14]:

- the stress at fracture extremities is proportional to the fracture length–to–width ratio (called *Inglis ratio*);
- a crack follows the line of maximum strain;
- the iso–strain curves around a fracture are parallel to the fracture edges; (thus fractures tend to interlace rather than to connect)
- imperfections in the material cause angular cracks propagation which can be modeled by a random angular deviation [1, 30].

Propagation criterion: We compare the fracture length–to–width ratio to a threshold. Moreover, we keep a link between the fracture elements across strips in a graph structure that we call the *fracture skeleton*. This structure allows us to identify each fracture entity.

Propagation direction: we first search for fractures in the neighborhood that may influence the local strain line. We assume the influence of a fracture is proportional to its width — thus the $f(a' \frac{dist}{width})$ in Figure 3 (where f is a weight function). In our implementation we used the mean fracture width, and we chose a simple threshold test function for f.

This base direction (from zero slope to the nearest fracture slope) is jittered within a user controlled amount in order to tune the angular aspect of the fractures.

3.2.5. Fracture opening

The widening of fracture directly depends on the stiffness of its fracture elements. Thus the variation law of the fracture stiffness provides a control of the fracture shape. In nature the resistance to opening is due both to the sticking to the substrate and to the rigidity of the bark material around the fracture end. Since it influences the shape and behavior of fractures, we prefer making the tuning of the $K_f(l_0)$ curve available to the user.

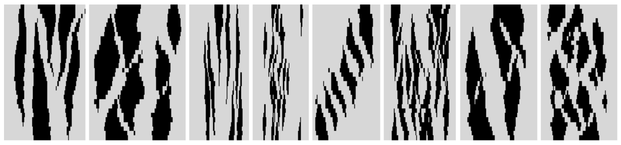


Figure 10: *Some results of bark simulation using various parameters.*

3.3. Dressing the Model with Details

Once the location and width of fracture elements is computed by the simulator, this raw data has to be turned into appearance information. That is, we need to texture a surface as shown in Figure 11 (the steps are shown in Figure 12). We rely on existing texture samples to cover the fracture and epidermal surfaces. The epidermal texture must not contain large scale fractures, since our model will generate it. However the texture can contain as much detail as needed to improve the visual quality of resulting bark.

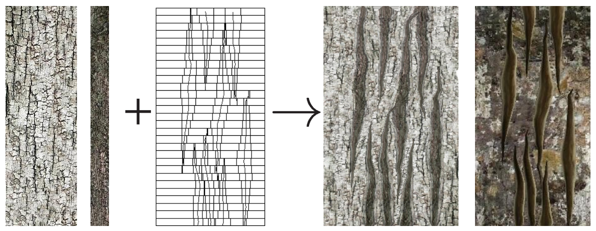


Figure 11: From left to right: *user provided texture samples for epidermis and fracture; the raw data to be texture produced by the fracture simulator the final bark.* Right: *The same simulation textures using other texture samples.*

The final step will depend on whether the user wants geometry or texture to represent the bark. In both cases, we first need to:

- reconstruct smooth fracture silhouettes from the discreet data;
- produce a mesh which embeds these silhouettes;
- map the user textures onto the epidermal and fracture regions of the mesh;
- optionally, determine and store the relief information.

If geometry has to be generated, the textured mesh can be used directly. Otherwise it should be considered as a two dimensional (2D) mesh in the textural space associated to the tree, and used to generate a 2D map.

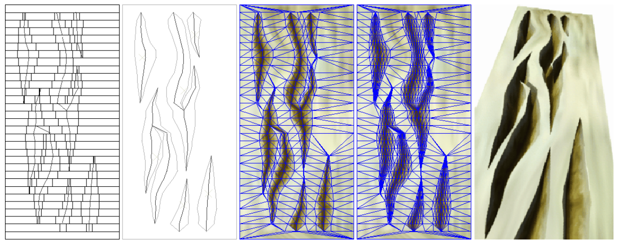


Figure 12: From left to right: *raw data; fracture silhouettes reconstruction; meshing; meshing with depth information; and generated geometry.*

Fracture silhouette construction: It is easy to produce a polygonal contour from the fracture skeleton. The only real problem is to handle correctly the case of fracture connection.

Meshing the fractured bark: The problem is to produce a mesh embedding the fracture silhouettes. In our implementation, we relied on the *Triangle* library [31] which efficiently creates a constrained Delaunay triangulation.

Texturing We have to map the epidermal and fracture texture samples onto this mesh. We stored in each fracture element on strips the coordinate where it appeared in the epidermis. This allows us to map the epidermis outside the fractures. The mapping of the inside depends on the mode of opening: The appearance of new material can occurs on the fracture axis, on one side, or both (see Figure 13).

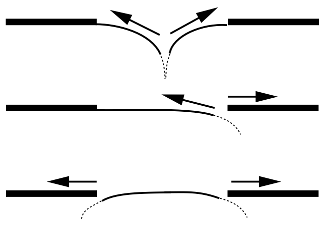


Figure 13: *Various modes of fracture opening.*

Relief information: Besides the bump or displacement information that may be embedded in the sampled textures provided by the user, we have to take the fracture depth into account. This might be wanted by the user even if he required a texture for result, since this depth can be encoded as a bump or displacement map. We compute the depth according to a normalized depth profile provided by the user (see Figure 14). It is evaluated on extra lines that we incorporate in the mesh, corresponding to depth isovalues (shown in Figure 12.4).

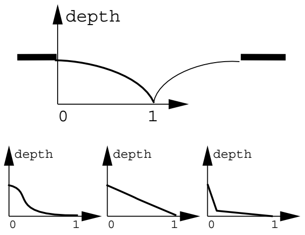


Figure 14: *Various depth profile in fractures.*

4. Results for flat bark

All the presented results but the Figure 9 and the 5th image of Figure 10 were produced without the use of a weakness map and without any intervention of the user during the simulation. After setting the initial parameters, the simulation had run automatically. Despite our model allows the tuning of the opening $K_f(l_0)$ curve, in practice we used a constant K_f for all fracture elements. Figure 9 was produced using a previous simulation as a weakness map. The 5th image of Figure 10 was created using interactive picking during the simulation to trigger fractures on the growing surface.

Some results of the simulation stage are shown in Figure 10 with varying stiffness and threshold parameters. Figure 17 shows the texturing of brittle fractures. Figure 15 illustrates that our bark model compares well with real images. The important point is that the appearance of tears and their distribution are equivalent to the original.

Figure 16 shows that different instances can be produced with the same tunings, which allows us to generate a forest composed of different trees. Conversely, it is easy to produce tilable bark samples by adding cycling boundary conditions.

Figure 18 presents an animation of the fracture propagation during the growth of the tree. The simulations presented on this page are interactive (a few steps per second). In practice, the calculation time varies linearly with the total number of elements (i.e. it is proportional to the number of strips times the average number of fractures per strip). The computation time for the dressing stage is linear for fracture silhouette construction and $O(n\log(n))$ for Delaunay triangulation. In practice, computing the texture of a bark with hundreds of elements takes less than one second. Our texturing algorithm can produce textured polygons, a bump map texture, or a colored texture. Rendering can be done using today's hardware. The bark is computed only once, after which the tree can be rendered in real time from any viewpoint, be it distant or close.

Our model can generate bark patterns semi–automatically. However, it suffers from the same flaws as the usual texture mapping approaches [3, 17] (i.e. discontinuities and scaling issues). The next section presents an extended model, that solve these problems by growing fractures directly on the tree surface.

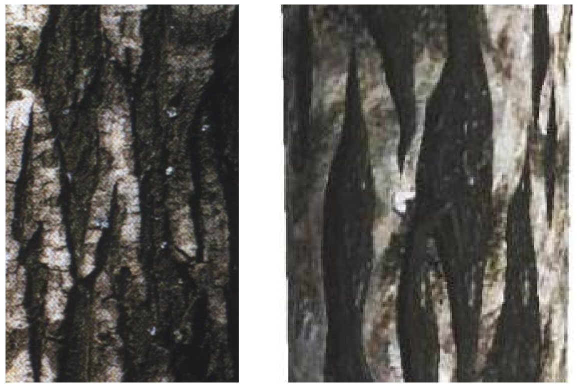


Figure 15: *Real (*left*) and synthetic (*right*) bark.*

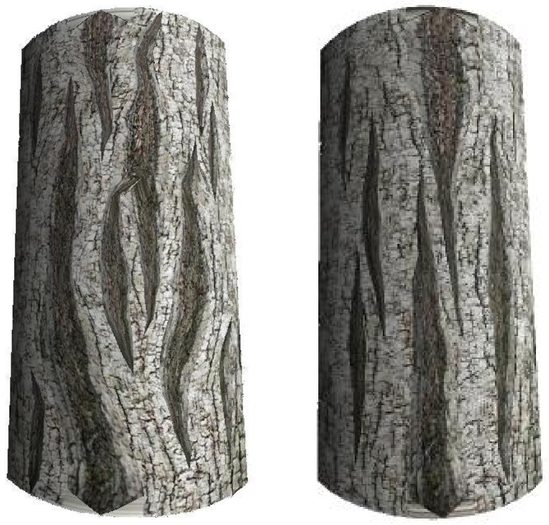


Figure 16: *Different instances of the same bark (*i.e. *same parameters)*

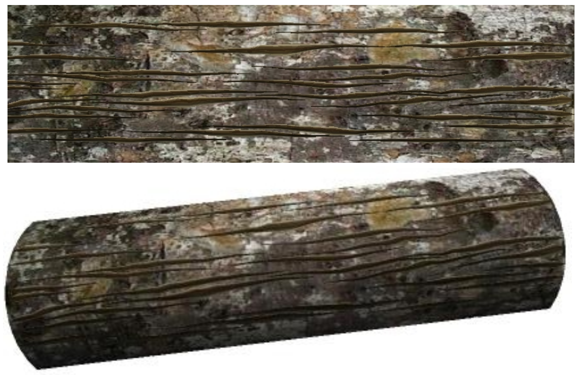


Figure 17: *Brittle fractures.*

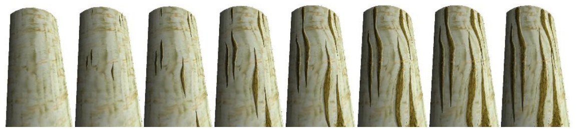


Figure 18: *Fracture propagation during the tree growth.*

5. Extended Model: Mappable Bark

We presented in Section 3 the base ingredients of our bark model: the strip system; the fracture system; and the texturing principle. This produces the convincing simulation of samples of bark shown in Section 4. To handle bark that really grows on a tree (which we call *mapped bark* despite the fact that it is not the simple mapping of a flat tile) we have to take new constraints into account:

- cyclic strips;
- attachment of the bark to the substrate;
- strips of varying width and length;
- branchings.

Handling deformed strips is necessary because trunk and branches are potentially general cylinders: their axis is curved, their radius varies longitudinally and also possibly around the circumference. Moreover we want to allow *heterogeneous growing* of the tree, which corresponds to a *dilatation field* varying along the axis and around the circumference.

The addition of an attachment property is needed. Otherwise, a degree of freedom remains due to the rotational symmetry. However, we define the attachment property precisely since it provides control over the locality of fractures: If an area grows more than the neighborhood (*e.g.* for the bulges due to the starting of main roots on the base of the trunk), fractures are more likely to occurs in the close vicinity rather than all around the circumference.

We deal with the attachment on cyclical strips in Section 5.1. We explain how we handle the mapping in Section 5.2. We treat the consequences of the mapping on fracture simulation in Section 5.3 and on the texturing in Section 5.4. Final results are shown in Section 6.

5.1. Handling the Attachment to the Substrate

The bark segments must remain rigid while the substrate grows smoothly, so the bark surface grows mainly by fracturing. Since the bark is 'glued' on the substrate, this is not sufficient to totally release the tension: Each unfractured bark segment sticks on a piece of substrate that has grown. This has consequences on the location of the segment (which tries to minimize the strain) and on the predominance of fractures (the higher the tension the more likely fractures are to appear). This gluing force can be modeled by "zero rest–length" springs attaching each bark location to the corresponding location on the substrate (see Figure 19): Their length is zero at the beginning of each time step (since the material is plastic at long term), so they yield a $K_a e_a$ resisting force if the dilatation causes a displacement of e_a. K_a is a parameter characterizing the attachment strength of the tree species, and controls the locality of fractures (in particular when the dilatation is not homogeneous).

The system to be solved for a strip is now:

$$\left\{ \ \forall i, \ -K_i e_i + K_{i+1} e_{i+1} + K_a e_a^i = 0 \ \right\}$$

that we rewrite as:

$$-K_i(x_i - x_{i-1} - l_0^i) + K_{i+1}(x_{i+1} - x_i - l_0^{i+1}) + K_a(x_i - a_i) = 0$$

where x_{i-1} and x_i are the curvilinear bounds of the i^{th} segment and a_i is the curvilinear location on the substrate of the bark vertex x_i. Once the equilibrium is obtained, we reset a_i location to x_i in order to simulate the long–term plasticity.

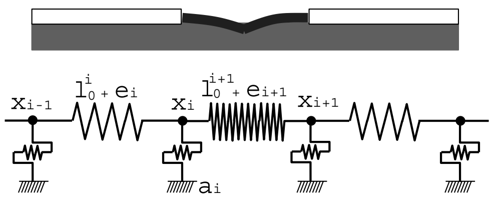


Figure 19: *Our mechanical model with attachment to the substrate.*

5.2. Mapping Fractures on a Tree

In this paper we assume that the tree geometry is provided by the user (*e.g.* it may come from specialized tools [2, 38]). In our implementation we generate the tree using generalized cylinders: trunk and branches are defined using a spline axis and a cylindrical radius function (user controlled at key positions). Whether the shape is given or generated, it is possible to compute a curvilinear parameterization on generalized cylinder shapes. Thus we assume the curvilinear parameterization is available for the trunk and each branch. (Note that we only need to account for the main branches since the epidermis of small branches is not fractured.)

Another issue is the evolution of the parameterization in time. Several possibilities occurs:

- **The growing of the tree is handled by the application.** The new shape is obtained from the shape at the previous time step using some user controllable growth law. So the law linking the parameterizations of a given strip through time is known.

- **The tree shape is given at various time steps**. Each shape can be parameterized with curvilinear coordinates. We can get the law linking the parameterizations by choosing an *a priori* projection between the shapes at two time steps. We have tested radial projection and normal projection. The latter is more natural but it suffers from the discrete calculation of normals on general cylinders using curvilinear parameterization (moreover, it is expensive), so we preferred the radial projection.

- **The user is not interested in the animation of the growing tree.** He simply wants to texture a given tree with bark. Then it is reasonably equivalent (and much more simple) to cover this tree with fractureless epidermis and then simulate the *shrinking* of the epidermis. We rely on a user–controllable shrinking law which is the reciprocal of the one mentioned above. However we only need the *amount* of shrinking along the strip; the displacement in space is useless.

The two first cases are interesting when one wants to visualize the continuous growth of a tree, such as for botanical simulation or special effects. This latter case is probably the most useful and the easiest to define in terms of user interaction. Note that despite the fact that our method does not handle the tree growth itself, it still generates continuous animation of fractures in time.

We assume from above that the mapping of strips along the tree and possibly through time is defined. Dealing with the consequences of the fracture simulation is the object of the next section.

5.3. Adapting the Fracture Simulation

Two aspects of the fracture simulation are affected by the mapping:

- the mechanical properties in distorted strips
- the propagation of fractures through strips, in particular at branchings.

These are the object of the two following paragraphs.

Adapting the strips material
As stated in Section 5, since the strips correspond to slices of trunk or branches that are potentially generalized cylinders, their width and length may vary (*e.g.* the strip length decreases with height and the strip width is smaller on the concave side of curved branches). This strip length variation is handled by parameterizing the strip using curvilinear coordinates. In that way, solving the strip equilibrium is done using the real lengths. The strip width variation is accounted for by adapting the K_i stiffness coefficient of bark segments on strip: We extend the expression in Section 3.2.2 by stating that an element of rest length l_0 and width w made of a material of stiffness R shows a global stiffness $K = \frac{Rw}{l_0}$.

Adapting the propagation of fracture through strips
We consider that each strip is parameterized independently and that a law connecting the parameterization of two adjacent strips is known. The best formal solution is to connect via the space coordinates: The fracture end on a strip is converted from curvilinear coordinates to space coordinates; then it is converted back to obtain the entry location in the next strip (see Figure 20). This formal definition can be optimized at implementation by relying on the global curvilinear parameterization of the generalized cylinder. However it has the advantage of being a *universal connection definition* ensuring spatial continuity. In particular, it allows the propagation of fractures from one object to another if an intersection exists, which provides an easy way of handling continuity of fractures at branchings. In our implementation we use the simplified conversion between adjacent strips along a branch and we use the formal solution at branchings: If an intersection with a branching cylinder is detected, we stop the fracture propagation on the main branch and we determine the strip and location of the propagation starting point on the child branch (see Figure 20, right).

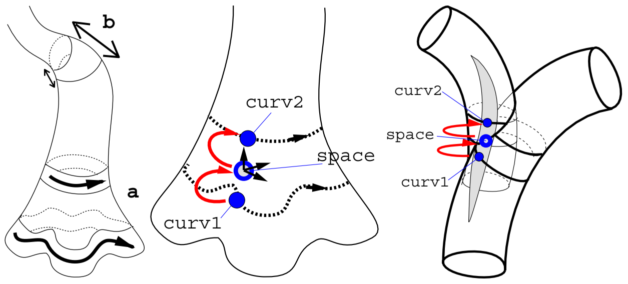


Figure 20: Left: *The length* (a) *and the width* (b) *of strips varies. Connection between strips along a branch* (middle) *and at branchings* (right)*.*

5.4. Mappable Dressing

As stated in section 3.3, once the fracturing of each strip is known, the problem is to reconstruct smooth fractures silhouettes and to texture the epidermis and the fracture areas with details. The technique is illustrated in Figure 21.

Fracture shape construction
This step depends upon whether a bark mesh or a bark texture is wanted by the user:

In the first case, the location and width of fracture segments can be converted to space coordinates and a mesh or a spline can be reconstructed on the surface of the generalized cylinder. Finally, the mesh of the tree is enriched with the geometry of fractures.

In the second case, we assume that the UV mapping of each generalized cylinder to be used for texturing is known (i.e. the user only wants us to compute the maps, and we should avoid distortion and discontinuity by painting correctly the content of each separate map). If no such mapping is available, it is easy to recover a parameterization of generalized cylinders.

By converting from strip curvilinear coordinates to UV coordinates we can reconstruct a smooth fracture shape in texture space, much like for the flat bark base model. Once it is textured with details as explained in the next paragraph, we render this 2D geometry using OpenGL and we store the image to obtain the resulting bark texture.

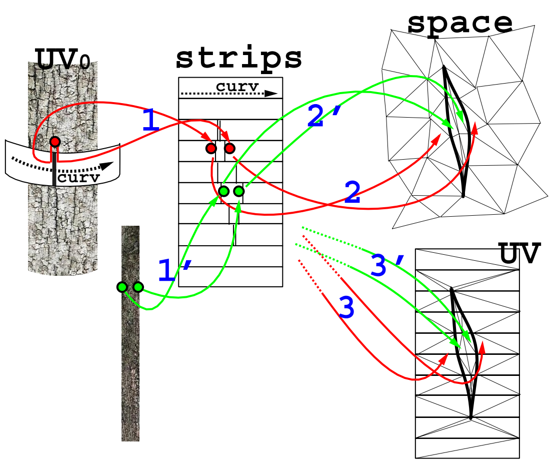


Figure 21: *1: The* UV_0 *coordinates are sampled at segment frac tures. 2: The reconstruction and texturing in 3D. 3: The reconstruc tion and texturing in a texture map* UV.

Texturing fractures and epidermis

The user provides an unfractured epidermal texture and a texture sample of the inside of a fracture. The epidermal texture can be defined by any classical means, depending upon the desired quality. For example, the texture can be painted on the whole tree or can be a simple tilable sample. The point is that the epidermal texture is quite homogeneous. Thus, low resolution, distortions, and discontinuities are not an issue: Our key hypothesis is that the dominant visible feature of bark is the fractures.

Let UV_0 be the provided mapping of the epidermis. If we are in the case for which the tree growing is simulated then the situation is much similar to Section 3.3: The UV_0 coordinates are stored at the edge of fracture segments when a bark segment breaks.

If the shrinking of the epidermis is simulated instead then we have to recover the UV_0 coordinates corresponding to a given break location. The UV_0 mapping is in bijection with the curvilinear coordinate on the unfractured trunk. We can obtain this coordinate from the current curvilinear coordinate by inverting the amount of dilatation between the two time steps, so we can recover the UV_0.

6. Results for Mappable Bark

The purpose of the extended model was to avoid the two classical mapping flaws, i.e. the distortion on irregular shapes (e.g. having a varying radius) and the discontinuity at branchings. Figure 22 shows several bark types mapped on irregular shapes. Note how the amount of fracture varies in order too keep a constant bark aspect. Figure 23 illustrates the case of branchings: the fractures propagate between the trunk and the branches. Note that the mesh is simply made of several intersecting parts. As shown on Figure 24 our model

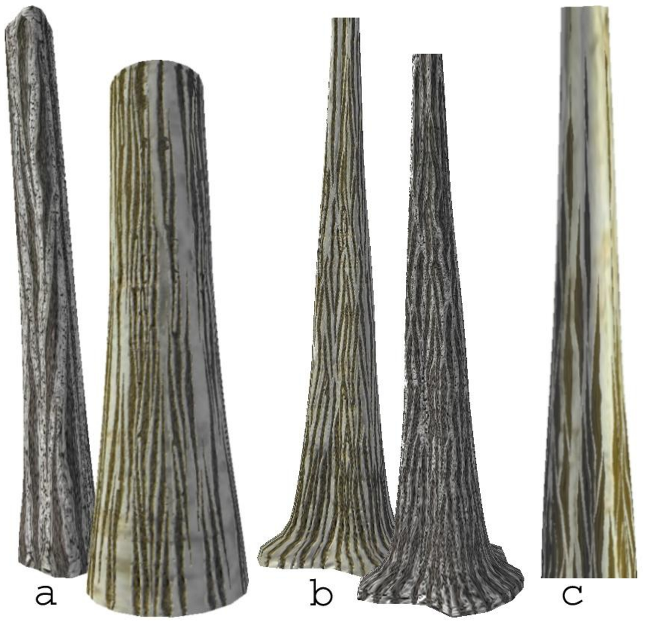


Figure 22: *Mappable bark simulated on generalized cylinders.* a) *Different fracture shapes and density.* b) *Constant fracture density along the trunk (despite the varying radius and the irregular section).* c) *Constant epidermis circumference along the trunk.*

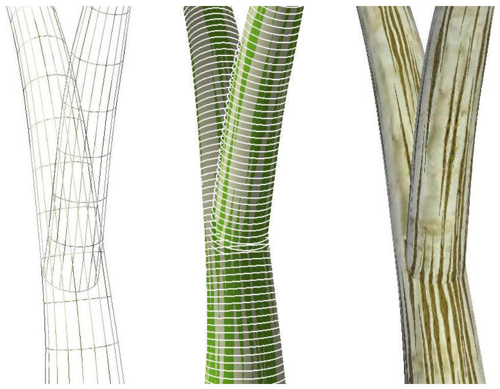


Figure 23: *Dealing with branchings: the fractures propagate from one texture to the other through the branch-trunk intersection.* Left: *The mesh of the tree model, corresponding to a set of independent intersecting generalized cylinders.* Middle: *The strips used on the two components.* Right: *Resulting bark textures.*

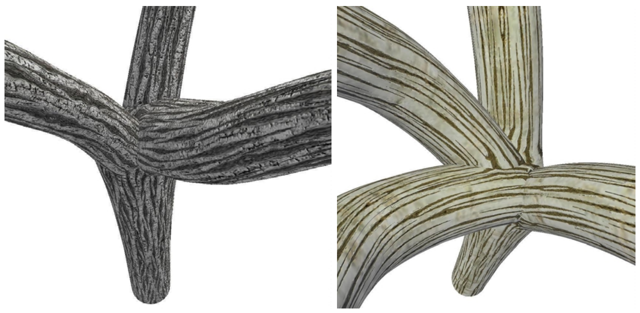


Figure 24: *Complex branching situations: the propagation through intersections works relatively well even in the case of sharp angles and multiple intersections (though such continuity is not expected to occur in nature, and might fail in some cases).*

works reasonably well even in complex situations. However it is possible to create arbitrary ill conditioned cases, while in such situation the nature itself do not provide continuity of features (e.g. on the sharp side of branchings). Note that the continuity would look even better if the underlying geometry itself was continuous at the branching.

For all these images, we relied on the generation of a bark texture including color and bump maps, and we used the shrinking strategy. The texture samples used for the dressing are the same as for the examples of Section 3 and 4. For Figure 23 we used 133 strips. The computation time was 49 s on a PentiumIII 900Mhz to simulate the fractures and was 11 s to generate the two textures (one for the trunk and one for the branch). For Figure 24 we used 225 strips. The computation time was 341 s to simulate the fractures and 37 s to generate the 5 textures. Once the texture are generated, the rendering is real time including the bump-mapping (we used a GeForce graphics board).

7. Discussion and future work

We have presented a model of bark generation that simulates convincingly and efficiently the fracture of the epidermis during the tree growing. The user controls material parameters that are closely linked to appearance, such as the density of fracture, the fracture shape, the locality, jittering, and so on (for a list of these parameters, see Figure 3). Moreover, the user can interact with the bark during the simulation: for example, by triggering fractures at given locations.

We have extended this model so that it can be mapped without distortion onto tree shapes and can take branchings into account. The propagation through intersecting surfaces works reasonably well, even in complex cases. It might fail in some ill conditioned cases, but it is correct on places where continuity is expected in nature (i.e. large branching angles, on the bottom side). This model is targeted to handle time continuity during tree growth. However, it can also be used on non–growing trees by shrinking the epidermis instead of growing the tree.

This model only addresses the family of *fracture–based* bark. However, this is justified by the fact that this type of bark is widespread and common, and cannot be handled by previous methods. The model can easily be modified to handle bark types having horizontal fractures. Note that our model produces realistic renderings from close viewpoints, a feature which is lacking in most CG applications showing trees. Moreover if a texture is generated the model handles far viewpoints as efficiently as current raw approaches.

Our model can generate either 3D bark mesh or 2D bark texture, both of which have been illustrated in this paper. Once we have generated a bark texture, today's accelerated hardware can be used to render the tree without any loss of performance.

The model generates the shape of tears and fractures, and 'dress' them with two user provided texture samples (i.e. it does not handle the generation of details). Actually, producing these maps is not a difficult task for skilled artists, who are used to this kind of work. Moreover, the visual aspect of our barks could be improved by jittering the border curve of the tears, adding lichens, and other features that do not have an impact on the growth simulation.

We are currently working on new texturing methods in order to handle other bark types, other visual features, and more natural continuity. For instance we would like to rely on recursivity to generate the bark details (i.e. microfractures as seen on Figure 9) instead of requiring two texture samples to dress the fractures. Moreover, various dressing features which do not influence the fracture growing should be introduced, such as rough surface aspect and lichen. Concerning "natural continuity", real bark appear to be only partly continuous, showing an hyperbolic fold on the sharp side of branchings. This kind of natural behavior should be introduced in our bark simulation in order to better approach the natural look.

Acknowledgments

We wish to thank James Stewart who carefully reread this paper. Thanks are also due to Alexis Angelidis for rereading an early version of the paper.

References

1. T.L. Anderson. *Fracture Mechanics, Fundamental and Applications*. CRC Press, 1995. ISBN 0-8493-4260-0. 3, 5

2. Bionatics. AMAP: a plant and scenery modeling tool. `http://www.bionatics.com/`. 1, 8

3. Jules Bloomenthal. Modeling the mighty maple. In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 305–311, 1985. 3, 7

4. D. Broek. *Elementary Engineering Fracture Mechanics*. Kluwer Academic, 1991. ISBN 9-0247-2580-1. 3

5. Eric Daniels. Deep canvas in disney's tarzan. In *SIGGRAPH 1999 Technical Sketches and Applications*, page 200, August 1999.
See also the CGW article:
`http://cgw.pennnet.com/Articles/Article_Display.cfm?Section=Archives&Subsection=Display&ARTICLE_ID=50479`. 1, 3

6. Phillippe de Reffye, Claude Edelin, Jean Françon, Marc Jaeger, and Claude Puech. Plant models faithful to botanical structure and development. In John Dill, editor, *Computer Graphics (SIGGRAPH '88 Proceedings)*, volume 22(4), pages 151–158, August 1988. 1

7. Oliver Deussen, Patrick Hanrahan, Bernd Lintermann, Radomír Mech, Matt Pharr, and Przemyslaw Prusinkiewicz. Realistic modeling and rendering of plant ecosystems. *Proceedings of SIGGRAPH 98*, pages 275–286, July 1998. Held in Orlando, Florida. 1

8. Alexei A. Efros and William T. Freeman. Image quilting for

texture synthesis and transfer. *Proceedings of SIGGRAPH 2001*, pages 341–346, August 2001. 3

9. A.A. Griffith. The phenomena of rupture and flow in solids. *Philosophical Transactions*, 221:163–198, 1920. 3, 5

10. Pat Hanrahan and Paul E. Haeberli. Direct WYSIWYG painting and texturing on 3D shapes. In Forest Baskett, editor, *Computer Graphics (SIGGRAPH 90 Proceedings)*, volume 24, pages 215–223, August 1990. 1, 3

11. J. Hart and B. Baker. Implicit modeling of tree surfaces, 1996. In Implicit Surfaces '96. 3

12. K. Hirota, H. Kato, and T. Kaneko. A physically-based simulation model of growing tree barks. *IPSJ Journal*, 39(11), 1998. in Japanese. 3

13. K. Hirota, Y. Tanoue, and T. Kaneko. Generation of crack patterns with a physical model. *The Visual Computer*, 1998. 3

14. C.E. Inglis. Stresses in a plate due to the presence of cracks and sharp corners. *Transactions of the Institute of Naval Architects*, 55:219–241, 1913. 3, 5

15. Iterative template library (a system solver). `http://www.osl.iu.edu/research/itl/`. 5

16. Bernd Lintermann and Oliver Deussen. A modelling method and user interface for creating plants. *Computer Graphics Forum*, 17(1):73–82, 1998. 1

17. K. Maritaud, J. M. Dischler, and D. Ghazanfarpour. Rendu réaliste d'arbres à courte distance. In *AFIG'00, (Proceedings of 13rd AFIG)*, 2000. in French. 3, 7

18. N. Max. Hierarchical rendering of trees from precomputed multi-layer Z-buffers. In *Eurographics Rendering Workshop 1996*, pages 165–174, June 1996. 1

19. N. Max and K. Ohsaki. Rendering trees from precomputed Z-buffer views. In *Eurographics Rendering Workshop 1995*, June 1995. 1

20. Nelson L. Max. Unified sun and sky illumination for shadows under trees. *CVGIP: Graphical Models and Image Processing*, 53(3):223–30, May 1991. 1

21. Alexandre Meyer and Fabrice Neyret. Multiscale shaders for the efficient realistic rendering of pine-trees. In *Graphics Interface*, pages 137–144, May 2000. `http://www.graphicsinterface.org/proceedings/2000/184/`. 1

22. Alexandre Meyer, Fabrice Neyret, and Pierre Poulin. Interactive rendering of trees with shading and shadows. In *Eurographics Workshop on Rendering*, Jul 2001. 1

23. M. Muller, L. McMillian, J. Dorsey, and R. Jagnow. Real-time deformation and fracture of stiff materials. In *Eurographics Workshop on Animation*, pages 113–124. Springer Wien NewYork, 2001. 3

24. M. Neff and E. Fiume. A visual model for blast waves and fracture. In *Proceedings of Graphics Interface '99*, pages 193–202. Communications Society, 1999. `http://www.graphicsinterface.org/proceedings/1999/117/`. 3

25. A. Norton, G. Turk, B. Bacon, J. Gerth, and P. Sweeney. Animation of fracture by physical modeling. *Visual Computer*, 7:210–219, 1991. 3

26. J.F. O'Brien and J.K. Hodgins. Graphical modeling and animation of brittle fracture. In *SIGGRAPH'99 Conference Proceedings*, pages 137–146. ACM SIGGRAPH, 1999. 3

27. Federl P. and Prusinkiewicz P. A texture model for cracked surfaces, with an application to tree bark. In *Proceedings of Western Computer Graphics Symposium*, pages 23–29, March 1996. 3

28. Ken Perlin. An image synthesizer. In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19(3), pages 287–296, July 1985. See also `http://www.noisemachine.com/`. 2

29. Przemyslaw Prusinkiewicz, Aristid Lindenmayer, and James Hanan. Developmental models of herbaceous plants for computer imagery purposes. In John Dill, editor, *Computer Graphics (SIGGRAPH '88 Proceedings)*, volume 22, pages 141–150, August 1988. 1

30. Sharif Rahman. Probabilistic simulation of fracture by meshless methods. `http://www.ccad.uiowa.edu/projects/solidmech/probsim.html`. 5

31. Jonathan Richard Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *Applied Computational Geometry: Towards Geometric Engineering*. Springer-Verlag, 1996. From the First ACM Workshop on Applied Computational Geometry. `http://www-2.cs.cmu.edu/~quake/triangle.html`. 6

32. Jeffrey Smith, Andrew Witkin, and David Baraff. Fast and controllable simulation of the shattering of brittle objects. In *Graphics Interface*, May 2000. `http://www.graphicsinterface.org/proceedings/2000/145/`. 3

33. Cyril Soler, François Sillion, Frédéric Blaise, and Philippe Dereffye. A physiological plant growth simulation engine based on accurate radiant energy transfer. Technical Report 4116, INRIA, February 2001. `http://www-imagis.imag.fr/Publications/2001/SSBD01`. 1

34. Pixar Animation Studios. Flagrant abuses of a perfectly nice texture system. In *SIGGRAPH Course Notes 24*, 1999. 1, 3

35. D. Terzopoulos and K. Fleisher. Modeling inelastic deformation: Viscoelasticity, plasticity, fracture. In *SIGGRAPH'88 Conference Proceedings*, pages 269–278, 1988. 3, 3

36. Greg Turk. Texture synthesis on surfaces. *Proceedings of SIGGRAPH 2001*, pages 347–354, August 2001. 2

37. Jason Weber and Joseph Penn. Creation and rendering of realistic trees. In Robert Cook, editor, *Computer Graphics (SIGGRAPH '95 Proceedings)*, pages 119–128, August 1995. 1

38. Xfrog. `http://www.xfrogdownloads.com/`. 1, 8