Image-Guided Fracture

David Mould

Department of Computer Science
University of Saskatchewan

Abstract

We present an image filter that transforms an input line drawing into an image of a fractured surface, where the cracks echo the input drawing. The basis of our algorithm is the Voronoi diagram of a weighted graph, where the distance between nodes is path cost in the graph. Modifying the edge costs gives us control over the placement of region boundaries; we interpret region boundaries as cracks. The rendering of our crack maps into final images is accomplished either by image analogies or by modulation of an uncracked texture.

Key words: Non-photorealistic rendering, fracture, texture synthesis

1 Introduction

Non-photorealistic rendering typically aims to reproduce a traditional artistic style or medium, and the styles and media reproducible on the computer have proliferated. A parallel thread of research has aimed at sculpting natural phenomena into specific shapes and images. For example, the target-driven smoke of Fattal and Lischinski [1] allowed users to specify a target shape to be crafted of smoke. Though not usually identified as non-photorealistic rendering, this second thread of research has much in common with NPR. It seeks to provide alternative media which artists can employ to create new images, and it has the twin goals of, first, producing a recognizable representative of the phenomenon in question, and second, being controllable enough that an artist's vision can be realized. We share these goals.

Fracture is a specific natural phenomenon which has seen attention from the graphics research community. Much work on fracture has concentrated on fragmentation of three-dimensional objects. However, for many cracks commonly seen in everyday life (such as cracks on pavement and on sidewalks), the cracked object appears reasonably intact; the crack is chiefly perceptible as a surface phenomenon. Because cracks are sharp edges, they are suitable for communicating simple line art images, provided that cracks can be placed at the proper locations.

In this paper, we present research which aims to synthesize easily controllable crack patterns. We use the well-known Voronoi diagram as the basis of our algorithm; we can guide crack placement by modifying the distance metric in specific ways. Our crack patterns are created in image space, and we expect that the usual way to incorporate them into a synthetic scene would be to apply them as textures.

We call the method *image-guided* because we use user-specified images to guide the placement of cracks: the input image will be rendered using cracks. Ideally, the image should be a line drawing or similar; text qualifies. The cracks populating the output image are likely to lie along the lines. Our main contribution is the algorithm which places the cracks.

The remainder of this paper is organized as follows. We have a description of related work in NPR and in fracture simulation. We describe our crack generation algorithm in detail: first the use of the discrete Voronoi diagram, then the modifications to edge weights which enable control over region boundaries. We suggest two methods for rendering – image analogies and intensity modulation – and show several examples of the resulting images. Finally, we discuss the results and give some ideas for future improvements.

2 Background

Numerous researchers have undertaken physical simulation to model cracking. Approaches include mass-spring systems, finite element models, and continuum mechanics. There has also been some work on generating crack-like images without recourse to simulation; in particular, the batik work by Wyvill, van Overveld, and Carpendale [16] is similar in spirit to our own, being concerned both with realistic-appearing cracks and with bringing cracks into the realm of non-photorealistic rendering.

We will first mention some of the results arising from the physical simulation side. Terzopoulos et al. [14] model elastic deformation with mass-spring systems, and this work was a precursor to the later work by Terzopoulos and Fleischer [13] which extended the approach to plastic deformation and fracture: when a spring stretches beyond its elastic limit, it breaks. Norton et al. [9] and Mazarak et al. [6] take similar approaches, attaching vox-

els together with springs which break when the local pressure or force exceeds a designated yield limit. The model of Norton et al. is used to produce an animation of a teapot being smashed, while that of Mazarak et al. is used to generate 3-D debris from explosions. Hirota, Tanoue, and Kaneko [4, 5] also use a mass-spring system; their work produces images and animations of crack patterns. Federl [2] explores the use of both mass-spring systems and finite element approaches to synthesize fracture patterns on growing surfaces. Müller and Gross [7] give a powerful real-time method for simulating elasticity, plasticity, and fracture, which obtains an enormous increase in speed by using a multiresolution simulation.

Brittle fracture was investigated by Neff and Fiume [8] and by O'Brien and Hodgins [10]. In these cases, the researchers treat the situation where an object fails spectacularly and catastrophically, being reduced to fragments.

The physical models tended to be slow and difficult to control. Although most of the physically-based methods could be made faster by reducing resolution, unpleasant aliasing occurs when the resolution is too far reduced, since cracks appear at the element boundaries. Müller and Gross's multiresolution simulation is fast; however, their system still does not address the issue of control over crack placement.

We have situated our work with non-photorealistic rendering. Cracks have been treated in an NPR context before: the batik project of Wyvill et al. [16]. These authors avoid physical models, preferring to use the distance transform; their approach is most similar to our own. They had good success at modeling cracks in wax, imitating the appearance of traditional batik crafting. Our approach addresses a different problem than theirs, in that we seek to secure detailed control over the placement of individual cracks.

Voronoi diagrams have been used for pattern synthesis previously, notably by Worley [15] who postulated a general texture synthesis primitive based on the nth-order Voronoi diagram. Although Worley did not explicitly address the use of his primitive for cracks, it is clear from the examples he provided that crack patterns can be generated using his technique. Voronoi diagrams were also suggested by Raghavachary [12] as a lightweight way of producing realistic-looking cracks. In Raghavachary's method, Voronoi sites are scattered across the polygons of an input mesh, and the Voronoi diagram computed. Cracks are produced either by interpreting all Voronoi boundaries as cracks, or by explicitly tracking the progress of a crack through the network – choosing a new edge to proceed along when a vertex is reached, and terminating the crack when it reaches a mesh boundary or a previously existing crack.

Figure 1: Pavement images. Top left, real pavement; top right, synthetic pavement. Bottom, the conference title as pavement cracks.

Our method uses a modified form of the Voronoi diagram, where instead of partitioning a plane we partition nodes in a graph, and our distance metric is path cost within the graph. The graph's edges are weighted, and the edge weights afford us some control over the region shapes. We attempt to produce realistic crack patterns, and have met with reasonable success here, though we have not exceeded the successes of previous researchers. What we have done as well, though, is to give control over the crack placements sufficient to allow an arbitrary line image to appear in the cracks, which to our knowledge no one has previously done. We give a preview of our results in Fig. 1, showing the two goals: realistic-seeming cracks and image-guided cracks.

3 Basic fracture algorithm

The basis of our algorithm is the well-known Voronoi diagram. In our case, we use path distance through a graph rather than the more usual Euclidean distance.

We give a formal definition of Voronoi diagrams, following O'Rourke [11]. Choose a collection of points $P = \{p_1, p_2, ..., p_n\}$ on the plane; call the points P the *sites*. Then, partition the plane into regions based on which site is closest:

$$V(p_i) = \{x : d(p_i, x) \leq d(p_j, x), \forall j \neq i\},$$

where $d(x,y)$ gives the distance between point x and point y. In our case, the distance in question is the cost of the shortest path through a graph. We interpret the boundaries between regions as cracks.

Our graph is a regular 4-connected lattice where each node is a texture pixel. We have a set of noise values $\{N\}$ such that each node i has a corresponding value N_i. The cost of an edge linking nodes i and j is

$$e_{ij} = \alpha \max(N_i, N_j) + \beta \quad (1)$$

for some constants α, β. In practice, almost any positive, nonzero α and β will suffice. The larger the ratio α/β, the more random the crack network becomes – that is, the effect of using nonzero α is to cause the cracks to deviate from straight lines. In this paper, we consistently used $\alpha/\beta = 6$. The noise values are in the range $\{0,1\}$. See Fig. 2 for a depiction of the Voronoi diagram and our derived crack pattern after adding noise.

We achieve local control by making local adjustments to the edge weights, according to the information contained in a user-specified image I, the same size as the output crack texture:

$$e_{ij} = \alpha \max(N_i, N_j) + \beta + \gamma \max(I_i, I_j), \quad (2)$$

where γ is the guide parameter, governing the weight given to the image. The image is most visible with γ considerably larger than $\alpha + \beta$, say $\gamma/(\alpha+\beta) = 4$; however, lower values for γ will provide some guidance as well. Just as with the noise values, the image values range over $\{0,1\}$.

We explain the use of γ and equation 2 in establishing control over crack placement in section 4. First, however, we give a brief description of our implementation of the Voronoi computation.

3.1 Implementation notes

We use breadth-first search to compute Voronoi regions. At every step of the algorithm, we have a "frontier": a set of nodes not yet in any Voronoi region, but adjacent to a node in some region. Each node in the frontier has an associated path cost: the smallest distance to a site among paths considered so far. We take the node from the frontier with the smallest path cost; call this node s. We add s to the appropriate region, and potentially add its neighbours to the frontier. For each neighbour of s not already in the region, say r, we compute a new distance $d(r)$, which is $d(s)$ plus the cost of the edge connecting s and r. If the new distance is smaller than the distance previously associated with r, or if r was not yet in the frontier, we add r to the frontier along with its new cost $d(r)$.

The algorithm begins with a frontier consisting of the sites, all at distance zero, and continues until the frontier is empty. When the algorithm terminates, all nodes in the graph will have been assigned to a region.

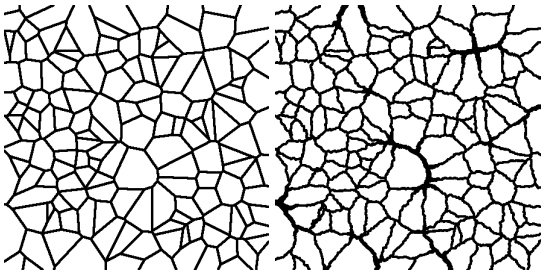

Figure 2: Comparison between the regular Voronoi diagram and the noise-Voronoi diagram.

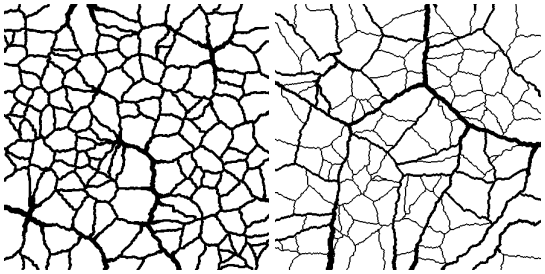

Figure 3: Basic cracking algorithm. Right, scattered sites; left, multiresolution site placement.

By implementing the frontier as a heap, we limit the insertions and deletions to $O(\log m)$, for m nodes in the heap. Usually $m \ll n$ for n nodes in the graph; certainly m is no larger than n. Each node can be inserted into and removed from the heap at most once, giving us a complexity of $O(n \log n)$ in the worst case, and more typically not much larger than $O(n)$. Note that n is the number of pixels in the output texture, and we should not expect to do better than $O(n)$ (we need to touch each pixel at least once to set its value).

3.2 Crack widths

We also want to have some variation in crack widths: a given crack should taper along its length. We achieved this by relating the crack width to its distance from the generating site; the further the site, the wider the crack. Enforcing this connection had the beneficial effect of placing additional emphasis on the cracks within the image edges, since those cracks come from Voronoi boundaries in expensive regions of the graph, with correspondingly high path cost.

Real cracking patterns sometimes exhibit multiscale cracking: earlier, more widely spaced cracks divide regions containing smaller and more closely spaced cracks. We can straightforwardly obtain multiresolution cracks within our framework, as follows. We choose a few sites

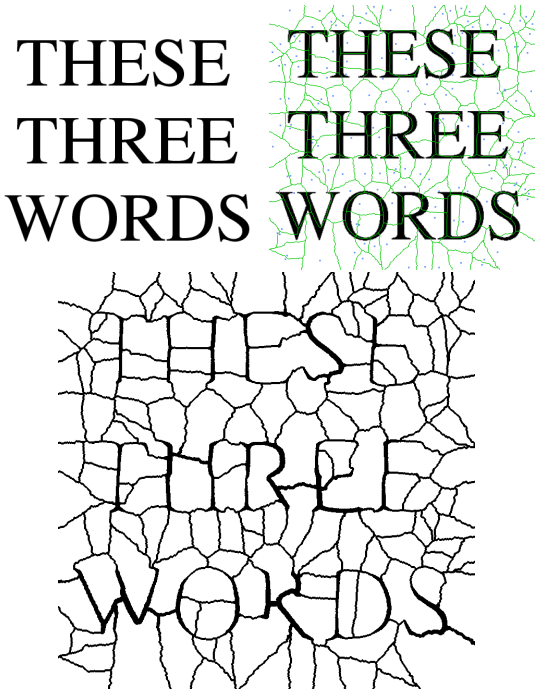

Figure 4: Some cracked words. Above, the original image and the Voronoi sites and region boundaries; below, the resulting crack pattern.

and obtain a crack map, use the resulting map to modulate the weight values, and select a new (larger) set of sites for a new crack map. The results are shown in Fig. 3, where the regular crack map is on the left, and a 3-level multiresolution crack map is on the right.

4 Controlling crack placement

We are able to exercise control over the crack placement chiefly by adjusting the edge costs in the underlying graph. A second form of control involves changing the locations of the Voronoi sites. Automatic mechanisms for placing Voronoi sites to produce images are not available, although local details can be created by manipulating the site placements by hand. However, some global effects can be accomplished by adjusting the distribution of sites and automatically choosing sites from the distribution.

We typically alter the edge costs according to the values in an input image – the image-guided cracking of this paper's title. We discuss this topic in the following subsection. Subsequently, we give some examples of simple effects made possible by modifying the site distribution.

4.1 Image-guided cracks

We have caused cracks to wander by using a quantity of additive noise on our edge costs. However, the results of doing so are not dramatically different from modifying the straight-line cracks after the fact. The main benefit of modifying edge costs comes when we do so in a structured way.

The most general means of modifying edge costs comes from using an image to do so, as in Equation 2. We have had most success by using binary images: edges which lie within the foreground of the image have their costs increased by some quantity, while edges in the background of the image do not. Under this edge cost system, if an area of the graph has high cost, it is likely to contain Voronoi edges. We next give some intuition as to why this is so, following our incremental implementation from section 3.1.

The graph edges in the image foreground have increased cost. Therefore, the cost of crossing a foreground area becomes prohibitive, so that a region entering from one side is not likely to complete the crossing before another region's expansion reaches it from the other side. When the two Voronoi regions do meet, it is in the middle of the costly foreground area, and the resulting boundary is interpreted as a crack.

Fig. 4 shows the Voronoi sites and region boundaries as an overlay on the input image; the region boundaries can be seen to preferentially occur within the image edges. That is, the cracks approximately follow the edges in the input image where possible, and otherwise wander in the usual cracking style. The guided edges sometimes follow paths which are not physically plausible; in particular, the curved paths along the S, D, and O characters are unlikely. The ability of physically-based methods to generate such paths is limited, while our method is able to satisfy the instructions of the user.

4.2 Fracture distribution

Both the number of sites (global density) and their distribution (local density) affect the final appearance of the crack pattern. We can create a pattern akin to pavement cracks by using a few sites distributed entirely randomly across the image, as shown in Fig. 1. Alternatively, we can deviate from a uniform distribution in the interest of other effects. Fig. 5 shows two of the possibilities. An effect like dehydration cracking of mud is obtained when the sites are distributed evenly over the image. When the local density is higher near the center, the cracks are clustered more closely, and the effect suggests that an impact took place near the image center.

We have not made much use of local density variations in obtaining our non-photorealistic effects. We typically used 150-200 sites uniformly distributed over the image. Nonetheless, purposefully varying the density remains a possibility for exerting some control over the crack patterns.

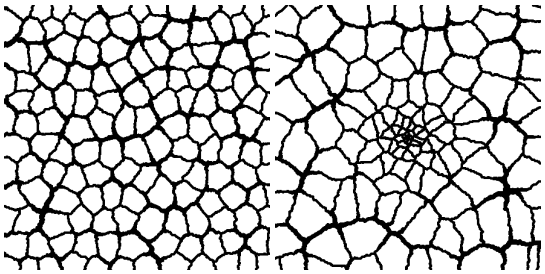

Figure 5: Different crack patterns obtained by manipulating the site distributions.

Figure 6: Image analogy for pavement cracks. Left to right: A:A'::B:B'. A' is a photo of real pavement, A is the derived crack map, and B is a synthetic arrangement of cracks.

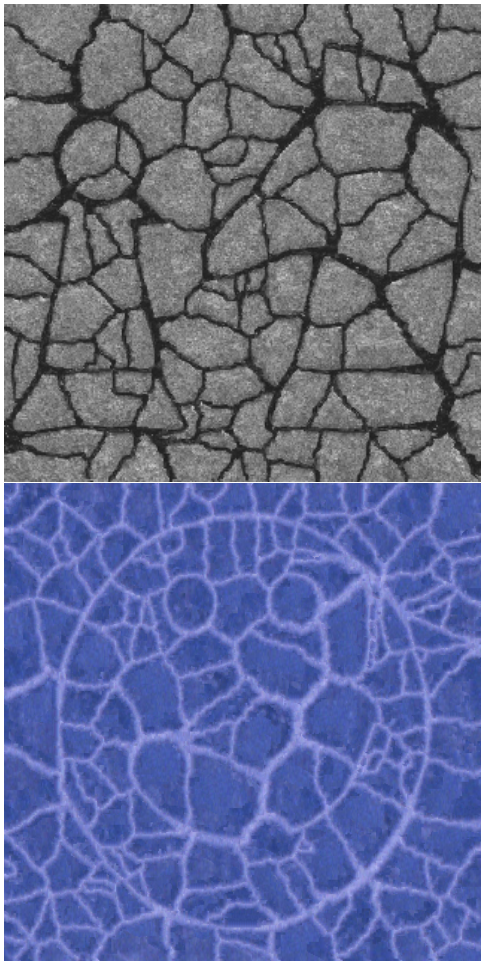

Figure 7: Textures transferred onto crack paths. Top: stone texture, with chess pieces. Bottom: ice, with a frozen smile.

5 Rendering

Having obtained crack locations, we should make a reasonable texture from them. One method for doing so is to employ a texture transfer process, such as that supplied by image analogies. Another method is to modulate an existing texture based on distance from the crack locations. The first method provides more visually pleasing results, but the second is considerably faster. We discuss both methods in turn.

5.1 Texture transfer

Texture transfer involves extracting the texture from one image for application to another. In our case, we want to take the texture from some cracked material so that we can apply it to the automatically generated crack maps.

We took photographs of cracked materials and marked each pixel as either crack or non-crack. The output from our own crack generation process is a binary image, also specifying each pixel as crack or non-crack. The *image analogies* work of Hertzmann et al. [3] provides a mechanism for applying the texture of the original photograph to the structure of the crack map.

Image analogies operates as follows. Given an analogy A:A', and an image B, the program attempts to learn a filter representing the transform A→A' so that the filter can be applied to B, producing B'. Our initial pair consists of the original photograph (A') and the derived crack map (A); B is the crack map generated by our method; and the final desired texture is B'. A specific analogy is shown in Fig. 6, and produced the pavement cracks previously shown in Fig. 1. Some additional results from applying image analogies to our crack generation problem are shown in Fig. 7, with different cracking patterns done in stone and ice styles.

5.2 Texture modulation

The textures produced by image analogies are appealing, but the processes of learning the filter and synthesizing the texture B' are slow. More troubling, the image analogies process is sensitive to small changes in the input map: thus, for a coherent input sequence, the output sequence may lack coherence. This is problematic when we want to create animations.

In consequence, we implemented a simple alternative method for obtaining a detailed crack texture: modulating the intensity of an input texture according to the

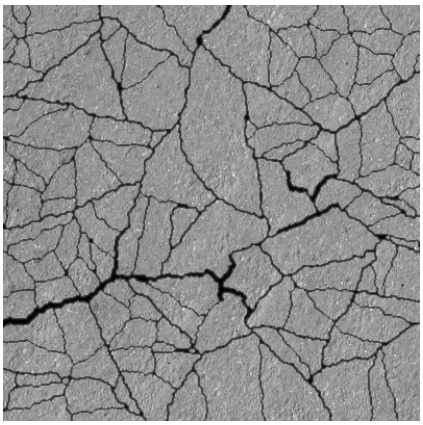

Figure 8: Cracks rendered by modulating an existing texture with the crack map.

crack map. Although the results are less attractive than the analogized textures, there are situations in which the modulation technique is more appropriate.

The method works as follows. Begin with an input texture showing the uncracked material, of the same size as the crack map. Then, for each pixel in the texture, compute the distance to the nearest crack pixel. Modulate the intensity by a monotonically increasing function of distance: for example,

$$I_{out} = I_{in} \times (1 - e^{\tau d}) \tag{3}$$

could be used, for a distance value d and a normalization factor τ. In practice, it is not necessary to visit every pixel of I, only those sufficiently close to a crack pixel that they risk having their intensity reduced. Each crack pixel can broadcast its presence within a small neighbourhood, and any texture pixel which lies outside all such neighbourhoods can be left untouched.

In Fig. 8 we show a modulated texture. As stated previously, the cracks are not as compelling as those generated by image analogies. Also, the technique is more restricted, in that it produces only dark cracks; for situations such as cracks in a mirror, or the ice cracks of Fig. 7, modulation does not work. However, for those materials where we would expect to see dark cracks, the modulated crack patterns are acceptable.

6 Animation

We have so far shown how static crack patterns may be generated. However, our framework offers two ways of animating the crack patterns: unconventionally, by adjusting the image guide parameter, and in the expected way, by allowing the cracks to progress across the texture. We discuss each of these in turn.

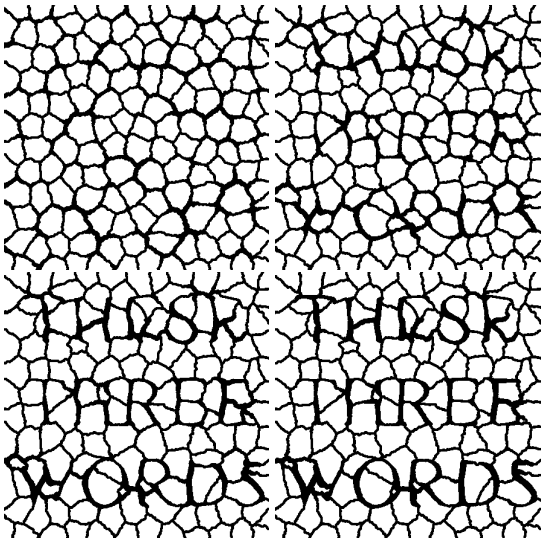

Figure 9: Words gradually emerging from the crack pattern.

6.1 Dynamic image content

By increasing the value of the guide parameter γ in equation 2, we increase the influence of the input image over crack placement. We have found that in practice the transition from no influence to strong influence is visually smooth. Any discontinuities in region adjacency are hidden by the width of the cracks. Fig. 9 shows a few frames from an animation in which γ is being increased: from a relatively regular crack pattern, the image slowly shifts until the text is unmistakably visible.

Each frame is a separately computed texture. Key to maintaining interframe coherence is that the noise map and site locations do not change: the only difference is in the value of γ. It is possible to accelerate the texture computation by only recalculating those regions potentially affected by the changed γ, but our implementation does not exploit this information.

It is possible to interpolate between crack patterns generated with different underlying images, say I^1 and I^2, by using two dynamic γ's: say

$$e_{ij} = \alpha \max(N_i, N_j) + \beta + \gamma_1 \max(I_i^1, I_j^1) + \gamma_2 \max(I_i^2, I_j^2), \tag{4}$$

and obviously we can extend the set of images indefinitely to obtain an animation sequence. Interpolation in the construction is crucial for crack images, because of their large-scale high-frequency content (i.e., prominent edges); the usual alpha blending performs extremely poorly on such images, creating obvious and distracting ghosting artifacts.

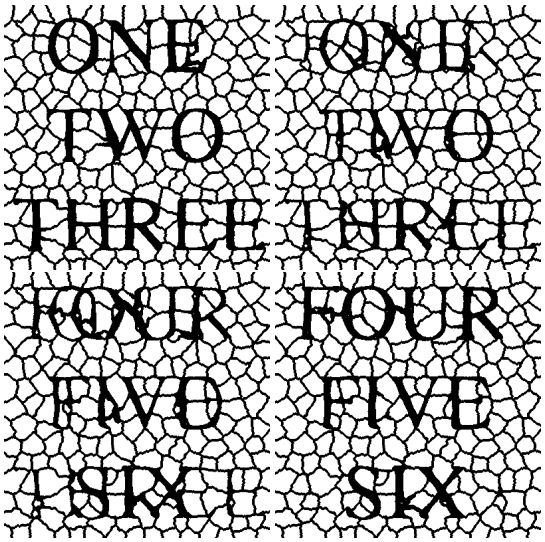

Figure 10: Words gradually emerging from the crack pattern.

6.2 Fracture expansion

Our crack widths are a function of their distances from the generating site, with further cracks being wider. We can simulate the effect of fracture expansion by introducing a time parameter into the width function. The simplest way of suppressing cracks early in time while allowing the full crack pattern to appear later is to modulate crack width over time:

$$w(t) = w_{\max} \min(t/t_{\max}, 1). \tag{5}$$

Though simple, the preceding equation is effective. Fig. 11 shows the effect of altering the crack widths, giving the illusion of a progression of cracking. The method works best when widely disparate crack widths are present in the image. When the cracks are all nearly the same width, the effectiveness of modulation is greatly reduced; the implicit assumption is that the cracks are all nearly the same age.

7 Discussion

The chief contribution of this work is the image-guided crack placement. The user need only supply a binary image showing the preferred edges, and the system will automatically produce a crack texture within which the cracks are aligned with the image edges. The system has some parameters, such as the number of sites and the guide parameter γ, but a user need not interact with these in order to achieve good results.

We also wish to highlight the use of path cost in a graph as a Voronoi distance metric. Other work using distance metrics typically uses Euclidean or Manhattan distance,

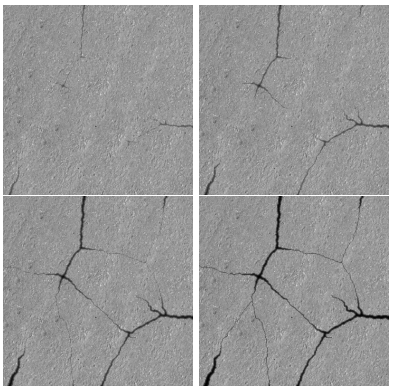

Figure 11: Cracks progressing across the pavement.

relying on later modification of region edges to escape excessive regularity. Different pattern formation effects are possible within the framework of distances in a weighted-edge graph, which might be seen as an augmentation of cellular texture. We have not much pursued this possibility here, since our interest in this paper has been in achieving local nonstationary control over the crack image, and not in texture synthesis per se.

The fact that we generate two-dimensional images could be seen as a drawback of our system. Much of the work in physical simulation of fracture has dealt with fragmentation of 3-dimensional objects. However, real cracks do not inevitably shatter the object on which they appear; there exist cases such as dehydration cracking of drying mud, or a 'cracked' glaze, or cracking leather or skin (e.g., elephant skin). We view our work as a form of NPR of the image filter type, in which an input image is transformed into a similar image done in the chosen artistic style. To our knowledge, no one has yet created images using cracks as the drawing primitive.

Not all images are amenable to being rendered in the crack style. We have shown examples using text and line art, which are the types of images best suited to crack rendering. Since comparatively few cracks appear in the final image, effects such as shading and crosshatching do not translate well. Rather, the best images are those containing simple shapes delineated with bold strokes.

One shortcoming of the system as implemented is that the graph must be computed at the texture resolution. Although there is scope for a multiresolution approach (the graph could be refined near the region boundaries) we have not yet undertaken such. Note, however, that computing the graph at the texture resolution does prevent aliasing artifacts from becoming prominent – we did not observe aliasing in any images we synthesized. Note that the final rendering process can remove aliasing artifacts which may appear in the crack map.

Lastly, we give a note on timing. We tested our system on maps of size 1000×1000 and found that generating the crack map required approximately 4.7 seconds on a 1.8 GHz Pentium. For the more typical 400×400 images shown in the paper, the synthesis time was approximately 0.9 seconds. These times do not account for the final render, which can be very fast (if crack modulation is employed, sub-0.5 seconds) or very slow (if image analogies is used, 5-15 minutes).

8 Conclusions and Future Work

At the outset, we gave two goals. First, we sought to create crack patterns difficult to distinguish from real crack patterns. Second, we attempted to control the crack placements so that an image could be seen in the crack segments. Of the two, the second is more important, since fracture has been so well studied in computer graphics.

We have given examples showing the plain crack patterns and the image-guided crack patterns. The real pavement in Fig. 1 can be distinguished from the synthetic pavement chiefly because of differences in the texture and lighting, and not because of discrepancies in the crack pattern. The numerous examples of image-guided crack placement show the cracks lining up with the input edges as instructed. We achieved image-guided crack placement by computing Voronoi regions on a graph with weighted edges, and we postulated that using the same system with different ways of generating edge weights could be used as a more general pattern synthesis tool.

There is room for improvement in the existing method, as well as scope for further research directions. We have cracked only flat surfaces, and may wish to apply cracks to the surfaces of three-dimensional objects, avoiding distortion in texture mapping. Our current method could be adapted to create texture directly on the surface of a mesh, by sampling the surface with Voronoi sites and nodes for the graph. That is, nodes would be distributed over the surfaces to be cracked, with the density of nodes corresponding to the resolution of the crack map; the nodes would be hooked together (e.g., by a Delaunay triangulation); and weights would be assigned to the resulting edges. The graph produced by this approach could be used as input to the cracking algorithm described in this paper. Note that nothing in the algorithm depends on the number of edges per node.

We also want to expand the range of images we can render in crack style. It might be possible to use networks of very fine cracks to suggest shading and texture, and thus to produce more nuanced images.

We are interested in further exploring what can be done with Voronoi regions on the weighted-edge graph. Although the utility of the framework as a general texture synthesis tool may be limited, there may be additional phenomena, such as lightning, which can benefit from being treated by a similar technique.

References

[1] Raanan Fattal and Dani Lischinski. Target-driven smoke animation. *ACM Trans. Graph.*, 23(3):441–448, 2004.

[2] Pavol Federl. *Modeling Fracture Formation on Growing Surfaces*. PhD thesis, University of Calgary, 2002.

[3] Aaron Hertzmann, Charles E. Jacobs, Nuria Oliver, Brian Curless, and David H. Salesin. Image analogies. *Proceedings of SIGGRAPH 2001*, pages 327–340, August 2001.

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

[5] Koichi Hirota, Yasuyuki Tanoue, and Toyohisa Kaneko. Simulation of three-dimensional cracks. *The Visual Computer*, 7(16):371–378, 2000.

[6] Oleg Mazarek, Claude Martins, and John Amanatides. Animating exploding objects. In *Proceedings of Graphics Interface '99*, pages 211–218, June 1999.

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

[8] Michael Neff and Eugene Fiume. A visual model for blast waves and fracture. In *Proceedings of Graphics Interface '99*, pages 193–202, June 1999.

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

[10] James O'Brien and Jessica Hodgins. Graphical modeling and animation of brittle fracture. In *Computer Graphics (SIGGRAPH '99 Proceedings)*, volume 33, pages 137–146, August 1999.

[11] Joseph O'Rourke. *Computational Geometry in C*. Cambridge University Press, Cambridge, 1990.

[12] Saty Raghavachary. Fracture generation on polygonal meshes using Voronoi polygons. In *SIGGRAPH 2002 Sketches*, pages 187–187. ACM Press, 2002.

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

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

[15] Steven P. Worley. A cellular texture basis function. In *Proceedings of SIGGRAPH 96*, pages 291–294, New Orleans, Louisiana, August 1996.

[16] Brian Wyvill, Kees van Overveld, and Sheelagh Carpendale. Rendering cracks in batik. In *Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering*, pages 61–69. ACM Press, 2004.