January 1990

Imperfection for Realistic Image Synthesis

Welton Becket
University of Pennsylvania

Norman I. Badler
University of Pennsylvania, badler@seas.upenn.edu

Reprinted from *The Journal of Visualization and Computer Animation*, Volume 1, Issue 1, January 1990, pages 26-32.

Imperfection for Realistic Image Synthesis

Abstract
The precision of image synthesis techniques for rendering naturalistic scenes often works contrary to the realism of the everyday world. Pristine, crystalline, uniform and perfect may describe the most idealized computer images: the surfaces are smooth, neat and crisp in appearance. Efforts to produce realism have recently focused on light and the interaction of light with surfaces. The radiosity methods have shown that proper treatment of light is often critical to the proper visual effect in an image. Even the best of these images is nearly surrealistic in its precision, and thus belies its synthetic origins.

Keywords
texture, texture specification, procedural texture, texture generation, fractals reflectance models, imperfections rule-based systems, natural language

Comments
Reprinted from *The Journal of Visualization and Computer Animation*, Volume 1, Issue 1, January 1990, pages 26-32.

RESEARCH PAPER

THE JOURNAL OF VISUALIZATION AND COMPUTER ANIMATION VOL 1: 26–32 (1990)

Welton Becket
Norman I. Badler

Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104–6389, U.S.A.

Imperfection for Realistic Image Synthesis

SUMMARY **Creating objects with surface imperfections is accomplished through texture specification and generation techniques. Based on fractal subdivision techniques and relatively simple distribution models, a wide class of surface imperfections may be generated, combined and rendered. The surface effects include scratches, splotches, smudges, corrosion, mould, stains and rust. A rule-based system is used to position the various surface imperfections on the texture map, and a simple natural language interface is used to specify the kinds of imperfections and their generative parameters through adverbs and prepositional phrases. Results along some of the imperfection dimensions are illustrated.**

KEY WORDS: Texture Texture specification Procedural texture Texture generation Fractals Reflectance models Imperfections Rule-based systems Natural language

INTRODUCTION

The precision of image synthesis techniques for rendering naturalistic scenes often works contrary to the realism of the everyday world. Pristine, crystalline, uniform and perfect may describe the most idealized computer images: the surfaces are smooth, neat and crisp in appearance. Efforts to produce realism have recently focused on light and the interaction of light with surfaces.[1–5] The radiosity methods have shown that proper treatment of light is often critical to the proper visual effect in an image. Even the best of these images is nearly surrealistic in its precision, and thus belies its synthetic origins.

A careful examination of the 'real' world, however, yields more dimensions to the quest for realism than lighting models. Image synthesis techniques also require geometric models and surface reflectance models that incorporate accurate physical material properties and their response to light.[6] These are critical to the *microscopic* (that is, pixel level) generation of correct light reflectance. We believe, however, that the present state of modelling fails to account for an important *macroscopic* dimension of realism in computer synthesized images: *real objects are dirty*.

In this paper we describe a model for adding imperfect textures to a surface to achieve a more 'natural' appearance: the surfaces will be 'damaged' by rust, mould, stains, scratches, smudges and so on. The microscopic level, for example, can tell us the wavelength-dependent reflection coefficients of rust, but not model the shape or location of the rust itself. Of course, a skilled artist can render such naturalistic details manually; our challenge is to provide a system for specifying and modifying such textures symbolically. By-passing manual drawing accomplishes several goals:

1. Textures are created algorithmically from simple specifications. In fact, we use a simple 'natural language' command syntax to specify the texture parameters and their values qualitatively.
2. Textures may be formally characterized by various generative processes, such as fractal or Gaussian distributions, and combination operations.
3. Textures can be made dependent upon the geometry of the surface; for example, scratches and rust may be more prevalent near exposed edges.
4. Textures are synthesized prior to rendering rather than applied during a post-rendering $2\frac{1}{2}$D compositing step.
5. Textures may be created by less artistically-skilled individuals.

Naturally, some of these issues are addressed by procedural textures anyway.[7–10] Our contribution in this paper lies in defining the kinds of procedures and their parameters and combinations that create corroded, dirty or otherwise blemished surfaces.

First we discuss an overall approach to modelling imperfection. This paper will limit the kinds of surface features considered to those which do not intrinsically deform the surface geometry. Then we present the procedural models that formalize and realize certain classes of these phenomena. Control over the entire texture generation process is through a rule-based system with a simple natural language interface allowing adjective and adverb modifications of texture parameters. Some examples illustrate the systematic variation of the parameters on a simple object. Finally, future efforts and extensions are discussed.

AN APPROACH TO MODELLING DIRTINESS

Dirtiness in the sense we will use is: *any deterioration of some preconceived, idealized notion of perfection corresponding visually to the result of human or natural processes.** We use *dirty*, *blemished* and *imperfect* as interchangeable, and call the set of all processes satisfying the above property *entropatic processes*.

Our desire is to make more realistic scenes employing the effect of dirtiness and not to simulate entropatic processes unless their implementation is necessary for realism or most efficient.

Ideal modelling of object imperfection involves intricate, time consuming, complex simulation of moving particles, free liquids and common actions of humans within a given closed environment over time. An approach using pure simulation is unsatisfactory because:

1. Simulation of humans and natural processes over time at a depth that could adequately cause even the simplest of imperfections is not practical, and especially with respect to humans, not currently possible.
2. Complete simulation of real world processes would be overcomputation because most imperfections are viewed from distances where exact details of distribution or local appearance are indistinct.

For these two reasons we propose imperfection modelling through a rule-based approach where user-specified rules guide and modify low-level, localized, general simulation of blemish instances. This significantly reduces application complexity by replacing complicated environmental simulation by general observation in rule form. The only simulation is then low-level generalized imperfection modelling.

The approach involves two steps:

1. *Blemish instance modelling*—finding some technique to model a localized instance or concept of a blemish.
2. *Blemish placement*—designing rules that place or control distribution and local simulation of instances. This process constructs relevant statistical parameters for local simulation given simple object information such as shape and composition and specific contextual information such as the use of the object or location of adjoining objects.

Example: Coffee stains

Coffee stains on tables generally stem from either:

(a) drips, or
(b) coffee that escaped the top and resides on the outside of the cup.

A stain instance of either type *can* be modelled with concentric rings of fractal Brownian motion (described below) defining colour gradations. The second type requires

*Note that the constraint to a physical process eliminates noise as a permissible imperfection.

1049–8907/90/010026–07$05.00

Received 30 January 1990

specialized smearing transformations corresponding to the results of moving the cup around.

Each stain instance of either type can be modelled locally by varying statistical parameters and considering surface material. To determine the type of stain to create the system can use the simple heuristic that we normally cause cup stains around the perimeter of a workspace or around eating areas and we cause drip stains near where we obtain coffee.

BACKGROUND FOR BLEMISH INSTANCE MODELLING

For simplicity the blemish types discussed in this paper are ones which from standard human distances appear two-dimensional. The two-dimensional imperfections comprise a large portion of our common notion of imperfection and have a very simple and potent application technique—texture mapping.

Although other approaches to modelling are possible and in many cases necessary, our example models are achieved through 2D fractal techniques and simple Gaussian or random distribution functions. These modelling techniques with appropriate interpretation are powerful, yet simple to describe and implement. The high-level approach is easily extendable and more complex modelling techniques can be incorporated. Aside from Gaussian or random distributions, the two techniques used are as follows.

Rule-guided aggregation

The Witten–Sander method of *diffusion limited aggregation*[1] can be generalized to what is perhaps best thought of as *rule-guided aggregation* in order to model other natural phenomena.

Diffusion limited aggregation constructs tree-like aggregates by simulating the diffusion of randomly moving particles in a 'sticky' environment. One or more particles are defined as sticky origins; during the diffusion simulation whenever a particle collides with a sticky origin or a stuck particle there is a chance that particle will also stick. Higher probabilities of sticking yield bushier trees.

If the moving particles are replaced by growth of the aggregation and the sticking probability replaced by a growth probability based on any set of growth rules considering such things as distance from the growth centre and position of other particles, interesting blemishes like rust and complex stains can appear, and many others are possible (see Figure 1).

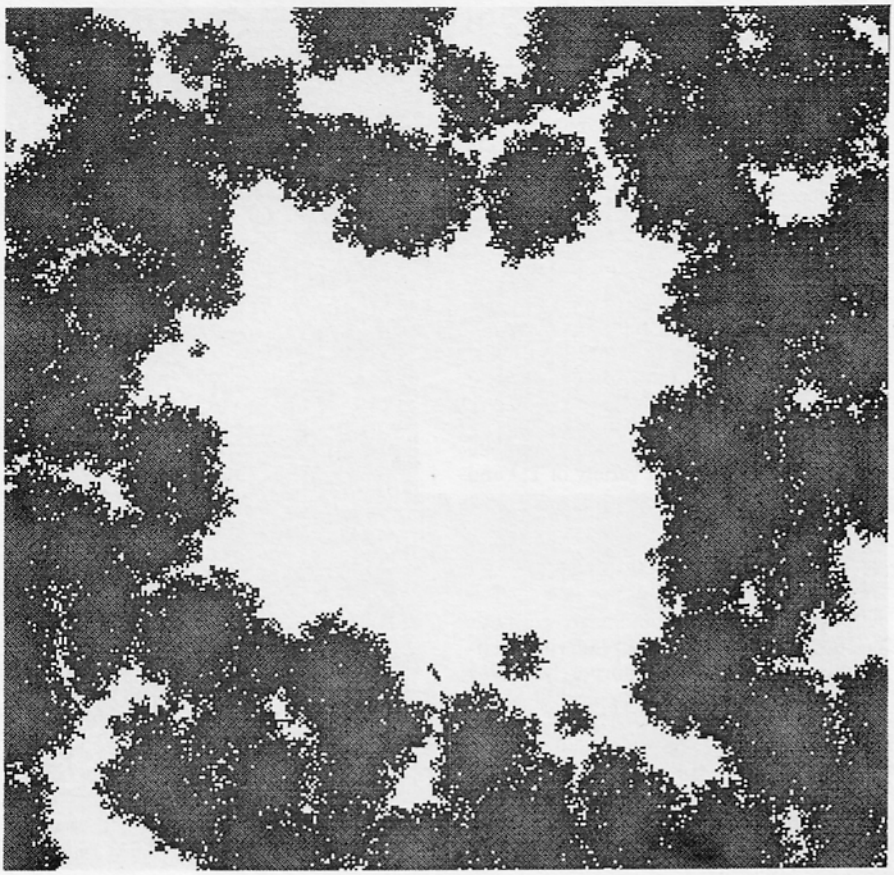

Figure 1. Rule-based aggregation of particles around the edges and around randomly placed seeds. This is the basis of our rust model

2D fractal subdivision

Two-dimensional fractal techniques generate a 2D array of values which, when post-processed and interpreted appropriately, can achieve blemishes exhibiting fractal boundaries or densities.

The fractal dimension of the array can be

(a) interpreted directly as a mapping between two surface qualities (see Figure 2)
(b) cut at a thresholding value to a binary map giving filled regions of circular fractal Brownian motion (fBm)† (see Figure 3)

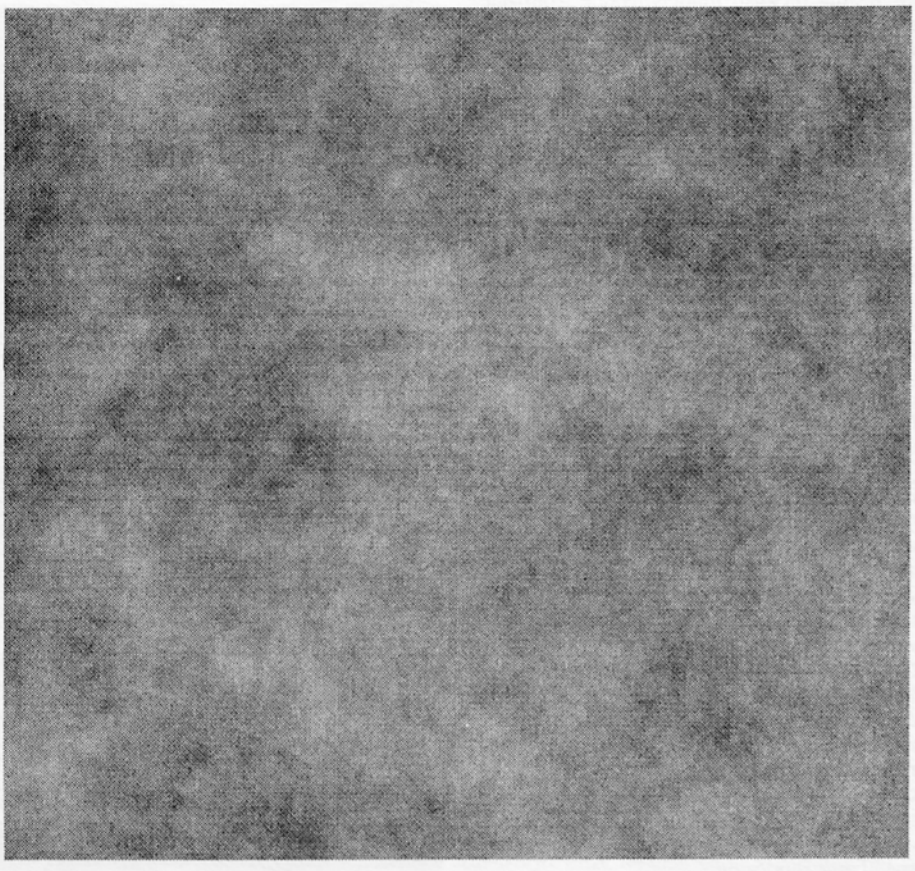

Figure 2. Interpretation of a 2D fractal as an interpolation of white and black. The fractal dimension is 2·5

†Closed 1D fractal rings.

(c) clipped to a range to form fBM rings (see Figure 4).

A particularly useful approach to interpretation of 2D fractals is to allow interpolation of attributes between pairs of contours. For example, with a fractal dimension normalized to [0·0, 1·0] the interval (0·0, 0·4] could be black, [0·4, 0·8] could be an interpolation from red to blue, and [0·8, 1·0] could be blue to green.

In our implementation we use subdivision techniques for their relative speed. The *creasing* associated with subdivision[11] is not as apparent in our use as when the fractal dimension is interpreted as a height field to create fractal mountains.

Dietmar Saupe's chapter in Reference 1 on algorithms for random fractals has code and good descriptions of a variety of 1D and 2D fractal techniques.

BLEMISH APPLICATION AND EVALUATION

The approach to realistic image synthesis shuld use the most effective rendering techniques available, even when purposely making imperfect objects. Although the most successful techniques to date are perhaps the non-diffuse environment radiosity method[2] and the hybrid radiosity and ray-tracing method, [3] only approaches to rendering with ray-tracing will be discussed. Since texture mapping is the underlying application technique our approach is extendible to mixed ray-tracing/radiosity methods.

Ray-tracing for testing

The highly reflective environments to which ray-tracing is naturally suited tend to accentuate dirtiness, making errors or possibilities clearer. For this reason we use ray-tracing for testing blemish models. Since from one test rendering to the next only surface attributes change, a problem-specific rendering acceleration method described below can be applied, which makes re-rendering on the account of attribute changes unnecessary, thus avoiding the enormous computational costs associated with ray-tracing.

Texture mapping techniques

At present we render dirtiness by using various models as texture maps on object surfaces. This avoids the computational burden of ray-tracing fractal densities, structured particle systems, and instantiations of object deformation methods without imposing unbearable constraints—both imperfections from normal human distances and the output of our example models are two-dimensional.

The type of texture map used is a generalized attribute map based on Cook's *shade trees.*[12] Associated with each mapping are the arguments 'from' and 'to', and a list of attributes. The 'from' and 'to' specify the source and target, either materials, other texture maps, or the host surface, between which the weights interpolate. The attribute list defines which surface attributes are modified (diffuse, specular, transparency, etc.). Texture maps of arbitrary depth are applied to a surface, and during rendering materials referenced are retrieved from a database of materials. This process keeps the texture maps separate from source or target and avoids writing all attributes into a file specifying rendering.

Modelling of thin coatings on both metals and non-metals requires weighted averaging

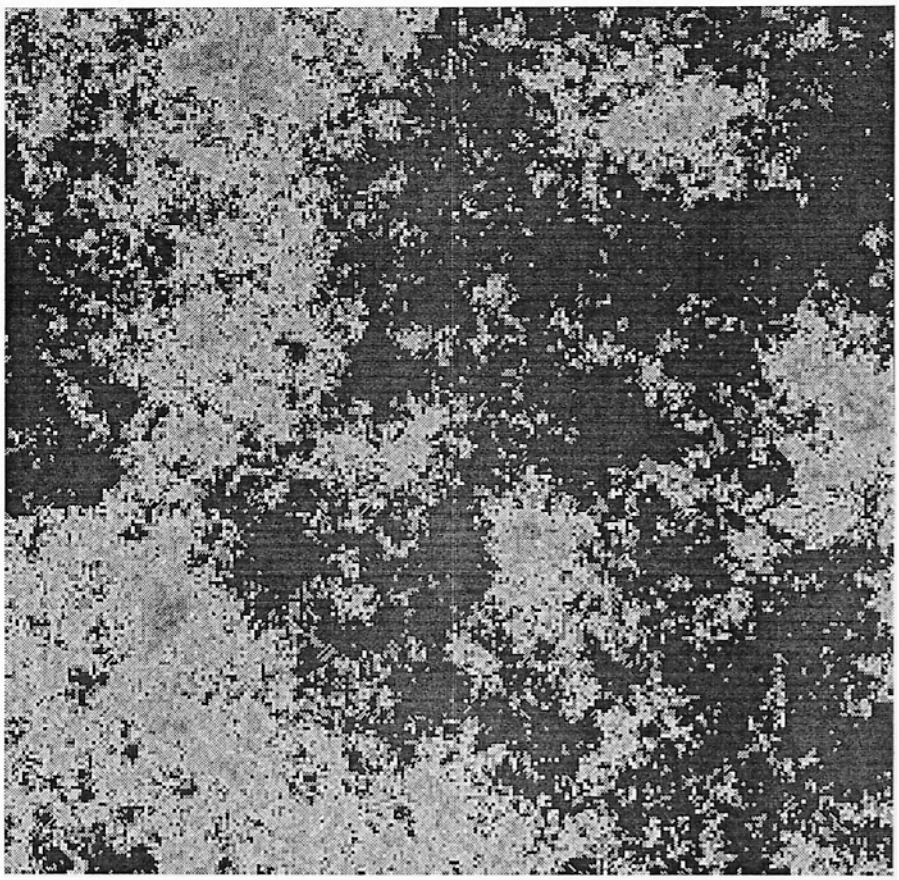

Figure 3. Fractal interpretation where [0·0, 0·5] is black and [0·5, 1·0] is an interpolation from white to black. The fractal dimension is 2·9

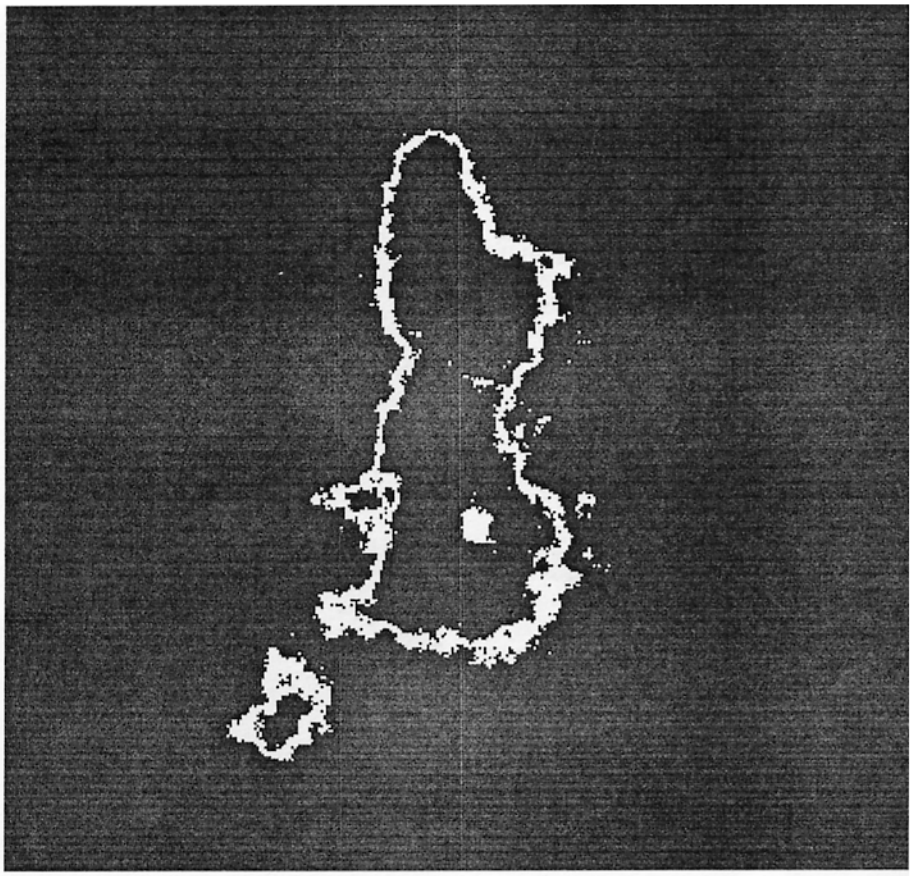

Figure 4. Fractal where [0·0, 0·65] and [0·7, 1·0] are black and [0·65, 0·7] is white. The fractal dimension is 0·75

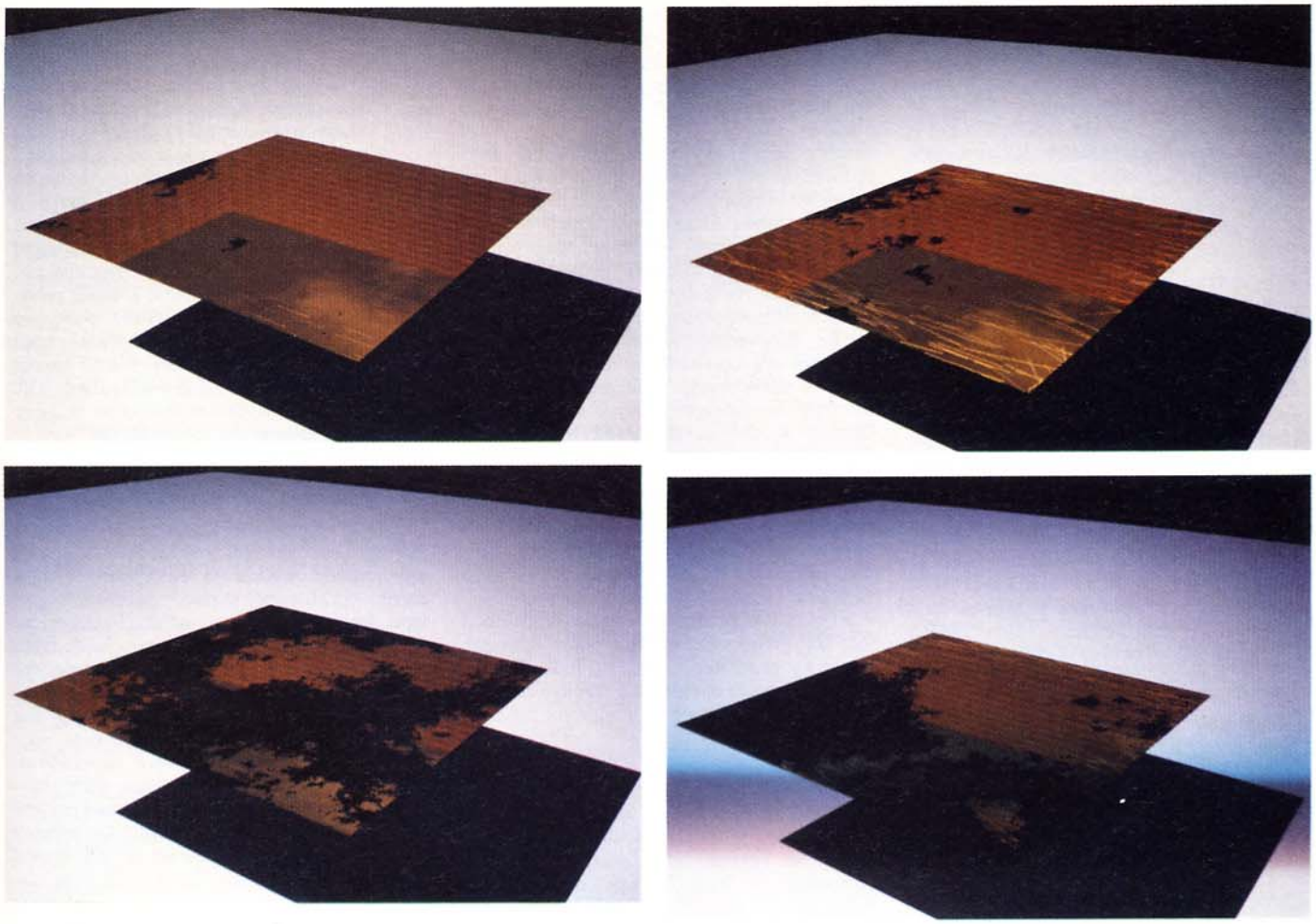

Figure 5. Planes in a room increasing from left to right in scratches and from top to bottom in tar-blotches

of the complex indices of refraction, the spectra, and the associated ambient, diffuse and specular terms for the materials in question. Storing averaging weights directly in the attribute map and specifying the above-mentioned attributes in the attribute list accomplishes this. The lighting model can then average the appropriate values at rendering time.

If the weight distribution for two attributes is unrelated then two maps are made. In many cases, however, the weights are proportional, and the addition of a scale factor to items of the attribute list avoids separate maps.

In all cases of semi-transparent coatings on materials it is appropriate to mix micro-facet distributions. If stochastic sampling[13,14] is an option, then a realistic solution is for the renderer, upon seeing a facet distribution flag, to perform two separate evaluations of the Monte Carlo integrals of incident light over a pixel, one with the facet distribution for the source and one for the target. The resulting values are then averaged with the weight defined in the texture map. During testing of a model, however, for execution speed's sake, using only one integral with an averaged facet distribution is perhaps better.

Maps of surface normal perturbations[15] can be used in addition to attribute maps to evince three-dimensionality when coating thickness is severe enough to demand it.

Storing shade trees

Storing symbolic representations of shade tress produced during ray-tracing rather than the resulting pixel values can substantially decrease re-rendering time, although incurring somewhat limiting increases in storage. Expressions such as

```
pixel 13: 0·77   *face8.color[u=0·2,v=0·51]
                 *face8.specular
                 + 0.23 * ambient
```

can be encoded into some relatively compact, binary format, independent of surface attributes. An instance of the shade tree file can then be rapidly displayed from a file containing surface attributes for each face. This makes testing different attributes and texture maps very fast.

This method has the additional benefit of not requiring the renderer to load a potentially large number of texture maps and permits image compilation in several passes when the number of texture maps or the complexity of a scene does not allow simultaneous loading.

The obvious drawback of storing shade trees, aside from the much larger file size, is its requirement than an environment be static with respect to attribute changes, ruling out both change of colour when using wavelength dependency and change of facet distribution when employing glossiness through stochastic sampling. The speed benefits, however, are enough to warrant such a move, especially while testing large numbers of images of the same scene.

A recent advancement and formalization of our simple notion is given in Reference 16.

IMPERFECTION MODELS

The imperfection models given are designed for realism at normal human distances and do not attempt to describe the close-up details. Our model construction emphasis has been on capturing the way an imperfection modifies a surface's attributes. Our fractal dimensions and distribution functions are found through visual experimentation. Correct and more realistic values can be obtained for our examples by

(a) collecting data about the distribution of instances on real objects
(b) measuring the fractal dimension of real blemish instances through box coverings.[11]

This approach is reasonable, since good modelling of an imperfection's attribute modification characteristics appears to dominate statistical details about physical shape or shape of the intensity distribution. A dried splotch can easily be recognized as such if the correct basic modelling is used, even if the fractal dimension is obviously wrong and the colour is unusual.

It is important to restate that the emphasis is *always* on how a model looks in final form at realistic distances and not how closely the simulation corresponds to the physical process.

The modelling approach

The high-level approach for synthesis of real-world dirty objects should certainly be through looking at actual objects. The appearance of objects, however, should not be

considered as a single unit, but rather as a composition of several *imperfection types* acting together. The acting classes can be isolated and approached independently since instances of classes in realistic quantities do not interact.

A workable approach for modelling a class is the following:

1. Decide whether there is some visible, *atomic unit* for the class, as with scratches or fingerprints, or whether the class is best described as only a distribution of intensity, as with mud or smudges.
2. If the class has an atomic unit then do one of the following:
   (a) If the atom is composed of a simple geometric primitive such as a line or a circle then goto step 4.
   (b) If the atom is some fractal boundary then model it either with rule-guided aggregation or with a circular fBm approach.
   (c) If the atom is an intensity distribution then model it with step 3.
   (d) If the atom is composed of a number of smaller primitives such as points or lines, then find a function, perhaps Gaussian or random, for modelling the distribution of subprimitives within the atom.

   Then goto step 4.
3. If the class is best described as an intensity distribution then model it possibly as either a fractal distribution approach or a growth of rule-guided aggregation. Use only loose approximations for fractal dimension, growth rules or distribution function.
4. Decide how the class modifies a surface's appearance by deciding which attribute flags to set in the attribute map or structure of maps to achieve the desired effect.
5. Find a region-dependent approach to writing the class instance or instances into maps based on simple positional noun phrases such as 'centre' or 'lower edge' (described in the Rule-base Section). A quick solution is to write fractal distributions directly into the map in the density distribution case, or to write many, randomly placed primitives in the atomic case, then subtract from weights in unwanted areas. A better approach for densities is to control the generating process by some statistical function, and a better approach for atomic classes is to write instances into map regions based on a distribution function.
6. Refine distribution functions, fractal dimensions and growth rules, where applicable.

Simple example models

Given here are, first, detailed accounts of both scratches and splotches to clarify the approach, then secondly, brief descriptions outlining the approach to other more involved imperfections.

Scratches

Scratches, being very easy to model in their simple form, are a good first example. They are simple geometric primitives, just lines, modifying surfaces either by increasing the facet-distribution value or by revealing what is underneath a composite surface.

Scratches on non-composite surfaces and light scratches on composite surfaces increase the facet distribution and the diffuse, and decrease specular by interpolation between *material-x* and *completely scratched material-x*. Scratches on composite surfaces that pierce the external layer turn one material into another.

Writing scratches into the attribute map reduces to writing lines whose lengths are Gaussian deviants of some given line size. Scratches tend to appear oriented towards some dominant axis, sometimes due to their actually occurring that way because abuse of an object is normally in the same manner, and sometimes because scratch visibility tends to be anisotropic. Correct modelling of scratches will most probably require consideration of the scratches' anisotropic behaviour through appropriate modifications to the lighting model as discussed by Kajiya.[17] Quick approximations, however, can exploit only the tendency for scratches to actually occur along a primary axis, accomplished easily by writing lines of Gaussian-deviant orientation from the given dominant axis. The intensities of the scratches should then vary as a Gaussian of a given intensity.

Regional dependency can be accounted for, as mentioned above, either through randomly generating scratches and subtracting non-scratched regions or through controlling the line generation through some function of attribute map position.

Splotches

Splotches of very viscous fluids or solutions, such as tar or mud, after having hit a surface, appear as several boundaried occurrences appearing to have fractal characteristics. Although the occurrences could be thought of as atomic units, it is better to model them as fractal densities, since there is normally a coherence between the splotches on a surface.

Modelling dried splotches is easily accomplished by interpreting a 2D normalized fractal of some approximate fractal dimension with:

> $[0{\cdot}0,n]$: non-existent (zero alpha value)
> $[n,1{\cdot}0]$: an interpolation from *surface-quality* to *dried-splotch*.

Higher values for n give sparser splotches.

Still wet splotches have fractal boundaried regions where some parts within the region are wet and some parts are dry.

A very opaque, drying splotch could be modelled by a single fractal generated texture-map as

> $[0{\cdot}0,n]$: non-existent
> $[n,1{\cdot}0]$: interpolation from the *dried-splotch* attribute into the *wet-splotch* attribute.

Other methods

(a) *Smudges and corrosion*: these can be modelled as fractal-based intensity distributions using the fractal dimension as an interpolation from the normal surface to the disturbed surface. Smudges are typically seen through their lowering specular. Corrosion interpolates from the original surface to the attribute of an oxide.

(b) *Mould*: a special case of mould, such as mould on cheese, can be just a Gaussian distribution of dots off central points chosen in randomly centred clusters.

(c) *Stains*: stains from a distance appear as just fractal boundaries, perhaps with areas close to the boundary darker than areas further inside. As mentioned above, coffee stains in particular, tend to have a dark border, which appears as a ring of fBm, and a light interior. Stains from normal or closer distances should probably use a low resolution fractal, spline interpolated between points on the boundary to make it smoother.

(d) *Rust*: rust on a surface can be modelled through rule-guided aggregation. Points near edges have a small probability of rusting, points in the interior have a very small probability, and points next to already rusted points have a high probability of rusting. The severity can be controlled by number of iterations.

RULE-BASED APPLICATION

Creating appropriate imperfection attribute maps for a scene implies application of several imperfection models to every surface of a scene, considering surface material, object type, face size, location within the face and severity of the various blemishes. Aside from consuming an imposing quantity of time, the process requires that a user defining a scene have substantial experience in setting the multitude of statistical and intensity parameters to the various low-level simulations. The knowledge required for generating attribute maps, however, can be defined in sets of context-sensitive rules, making the process eligible for control through a rule-based system.

Natural language interfacing

A powerful approach to the use of such a map-generating rule-based system is through a natural language interface. Natural human language is not only potent at visual description, but also, not surprisingly, a more natural expression medium than menu interfaces or context-free languages contrived for the occasion. An interface of this type can also disambiguate most of the context-sensitive aspects, simplifying the rule-base construction process by breaking it into two separate pieces, a high-level part handled by the parser which uses adjectives and adverbs to compute generalized statistical parameters, and a low-level part applying imperfection-specific rules using context information and provided or language-implied surface material and geometry information. The standard barrier to interfacing through natural language is the immense quantity of common knowledge required for most applications. By limiting the allowable context-sensitivity, however, natural language becomes a possible interfacing technique, although the resulting language is then a subset of English.

An input language

The parser could accept command constructions such as

> *Make a very rusty, shiny, copper cube that is slightly smudged near the center of the left face.*

Our commands consist of a verb, a noun part and an optional compound relative clause. The structure generated by a parser* is analysed by various rules considering the material, the object, the severity of the imperfection, and where the imperfection is to occur to create appropriate attribute map

structures. When any of the arguments to the map rules are missing the system can introduce partially randomized, relevant default values. If in the above example command no severity adverb like 'very' was supplied for rust, the system could use some slight Gaussian deviant of the average severity of rust applied to copper.

To keep the rule base manageable, knowledge called upon by sentences or any other chosen input language should be limited to:

1. attribute adjective parts directly modifying rendering such as 'very shiny' or 'bright red'
2. imperfection adjective parts for which there exists a model such as 'rusty' or 'extremely scratched'
3. complete object nouns such as 'teapot' or 'cube', and noun parts referencing subparts of those objects such as 'the tip of the spout' or 'the top face'
4. locational prepositional phrases referring to a noun or subnoun part such as 'very close to the edges' or 'near the top face'.

Allowing sentences such as

make a used-looking teapot

would call on too much external knowledge. But sentences such as

make a copper teapot that is stained around the spout, rusted and slightly scratched on the lid, and extremely scratched on the bottom

are interpretable if the rule-based system knows about the structure of a teapot and how stains, scratches, and rust are applied to each part.

Numerical interpretation of adverbs

The intensity or severity of a rendering or imperfection adjective and the specific interpretation of locational prepositional phrases are achieved through interpreting adverbs numerically. Adverbs are assigned numbers, as in the following for adverbs applied to imperfection adjectives:

adv(slight, 0·05)
adv(partially, 0·10)
adv(somewhat, 0·20)
adv(default, 0·25)
adv(moderatelly, 0·35)
adv(very, 0·55)
adv(extremely, 0·75)
adv(completely, 1·00)

The values are the estimated coverages of the imperfection. Adverbs modifying rendering qualities such as shininess or glossiness can take the same set of adverbs but with different values. Adverbs modifying colour would, however, have a different set of adverbs, perhaps 'dark', 'bright', 'light', and 'dull'.

Making attribute maps

The rules for each imperfection type can consider the following arguments: material type, object type, object size, location within the object, severity of imperfection, and contextual information. From these the rule-base can construct attribute maps and invoke the renderer in the following way:

1. *Material*—the material type may modify severity, but most importantly must be given to the renderer as either the 'from' or 'to' field for a map, or as an attribute inherent in the surface.
2. *Object type*—the object type is used to decide how many attribute maps to make and what kind of coherence is necessary between them, if any coherence at all.
3. *Object size*—used to scale size of blemishes. For example, scratches of the same severity will normally be the same size although the object size may vary, so scratches on a map for a small object should be larger than scratches for a large object if map size is constant.
4. *Location*—information about location is either encoded into some identifier to tell the map-generation function where within a map the imperfection is, or used by the system to choose faces on which to place blemishes.
5. *Severity*—the severity is used to give external functions fractal dimension, cut-off height, standard deviation, weight average or number of primitives.
6. *Contextual information*—contextual information for each imperfection can be such things as former or current locations of objects or some high-level indication (such as specifying the whole left-bottom of an object) of which faces are exposed to human-hands, humidity, or whatever is being considered. This information is used to determine overall placement and intensity and statistical parameters as a function of position.

EXAMPLES

The first example is an exaggeration to show what our models look like in extreme, enlarged cases. The images in Figure 5 increase in scratch severity along the left-to-right axis and increase in tar-splotch severity along the top-to-bottom axis. They were generated and rendered by the rule-based system from English commands. The system added a default environment of an open-ceiling room with fractal clouds mapped above it.

The actual sentences given to the system were for (tar-axis, scratch-axis):

(1,1): *make a very shiny, copper plane which is somewhat scratched near the edges and somewhat blotched.*
(1,2): *make a very shiny, copper plane which is moderately scratched near the edges and somewhat blotched.*
(2,1): *make a very shiny, copper plane which is somewhat scratched near the edges and moderately blotched.*
(2,2): *make a very shiny, copper plane which is moderately scratched near the edges and moderately blotched.*

The images in Figures 6 and 7 are composed of several objects generated by the rule-based system.* Figure 6 contains exaggerated blemishes and shows some subtle uses of imperfections such as the specular-reducing smudges on the floor. Figure 7 shows several extreme uses of imperfection.

FUTURE WORK

Future work will concentrate on including three-dimensional imperfections, increasing the number of imperfections, tuning the realism of the instance-models and the rule-base, and increasing the power of the language interface.

3D imperfections

Exciting possibilities for modelling surface deforming processes such as deterioration,[20] fracture due to inelastic deformation[21] or deformations from collisions exist and can be placed appropriately with a rule-based system as with two-dimensional phenomena.

More 2D imperfections

A predominant 2D imperfection is *dust*. Dust poses interesting distribution problems because the appearance of dust is primarily through its negative space where objects have recently been removed or where an object has been dragged along the surface. It involves reasoning over a geometric database constructed from the scene to determine the shape of objects specified as once having been on the object in question.

Enhanced realism

Finding exact fractal dimensions and distributions of existing blemish models can help the realism, but perhaps most important is to include a strong notion of actual, physical materials in the low-level instance modelling to give more realistic appearances to models for corrosion and thin-coating-based imperfections.

Language extensions

A good extension to the language interface, aside from allowing more complex sentence structure, is to incorporate simple pronoun reference that is either local to a sentence or relative to some well known, single, discourse focus (like a cube defined in a given session's first sentence). Adding such references ('more scratches' or 'less shiny', for example) will simplify the iterated expression of parameters while trying to create a particular image.

ACKNOWLEDGEMENTS

This research is partially supported by Lockheed Engineering and Management Services (NASA Johnson Space Center), NASA Ames Grant NAG-2-426, FMC Corporation, Martin-Marietta Denver Aerospace, NSF CER Grant MCS-82-19196, and ARO Grant DAAL03-89-C-0031 including participation by the U.S. Army Human Engineering Laboratory.

REFERENCES

1. Tomoyuki Nishita and Eihachiro Nakamae 'Continuous tone representation of three-dimensional objects taking account o shadows and interreflection', *Compute Graphics*, **19**, (3), 23–30 (1985).
2. David S. Immel, Michael F. Cohen an Donald P.Greenberg, 'A radiosity metho for non-diffuse environments', *Compute Graphics*, **20**, (4), 133–142 (1986).
3. James T. Kajiya, 'The rendering equation *Computer Graphics*, **20**, (4), 143–150 (1986)

*Our parser, written in Common Lisp, uses a chart-parsed combinatory categorial grammar to convert the commands into a functional semantics. Combinatory categorial grammar was chosen for its particularly nice properties with respect to co-ordination and extraction. For a good description of combinatory categorial grammar see Reference 18, for a description of chart parsing such a grammar see Reference 19.

*With the exception of text and graffiti.

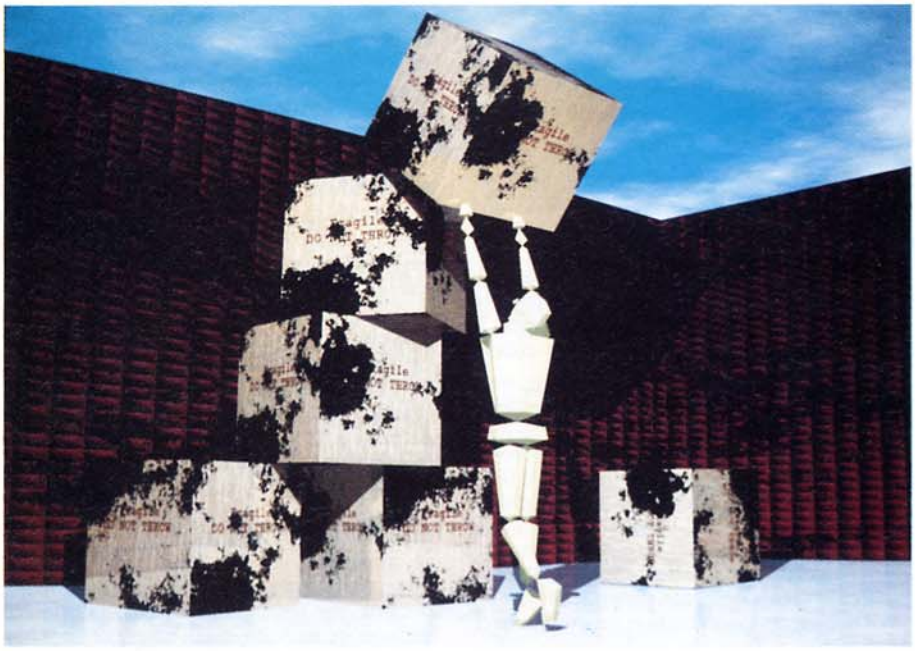

Figure 6. Image composed of several objects generated by the rule-based system

Figure 7. Another image composed of several objects generated by the rule-based system

4. Michael F. Cohen, Shenchang Eric Chen, John R. Wallace and Donald P. Greenberg, 'A progressive refinement approach to fast radiosity image generation', *Computer Graphics*, **22**, (4), 75–84 (1988).
5. Min-Zhi Shao, Qun-Sheng Peng and You-Dong Liang, 'A new radiosity approach by procedural refinements for realistic image synthesis', *Computer Graphics*, **22**, (4), 93–101 (1988).
6. Robert L. Cook and Kenneth E. Torrance, 'A reflectance model for computer graphics', *Computer Graphics*, **15**, (3), 307–316 (1981).
7. K. S. Fu and S. Y. Lu, 'Computer generation of texture using a syntactic approach', *Computer Graphics*, **12**, (3), 147–152 (1978).
8. Geoffrey Y. Gardner, 'Simulation of natural scenes using textured quadric surfaces', *Computer Graphics*, **18**, (3), 11–20 (1984).
9. Darwin R. Peachey, 'Solid texturing of complex surfaces', *Computer Graphics*, **19**, (3), 279–286 (1985).
10. Ken Perlin, 'A image synthesizer', *Computer Graphics*, **19**, (3), 287–296 (1985).
11. M. F. Barnsley, R. L. Devaney, B. B. Mandelbrot, H.-O. Peitgen, D. Saupe and R. F. Voss, *The Science of Fractal Images*, Springer-Verlag, 1988.
12. Robert L. Cook, 'Shade trees', *Computer Graphics*, **18**, (3), 223–231 (1984).
13. Robert L. Cook, Thomas Porter and Loren Carpenter, 'Distributed ray tracing', *Computer Graphics*, **18**, (3), 137–144 (1984).
14. Robert L. Cook, 'Stochastic sampling in computer graphics', *ACM Transactions on Graphics*, **5**, (1), 51–72 (1986).
15. James Blinn, 'Simulation of wrinkled surfaces', *Computer Graphics*, **12**, (3), 286–297 (1978).
16. Carlo H. Séquin and Eliot K. Smyrl, 'Parameterized ray tracing', *Computer Graphics*, **23**, (3), 253–262 (1989).
17. James T. Kajiya, 'Anisotropic reflection models', *Computer Graphics*, **19**, (3), 15–21 (1985).
18. Mark Steedman, 'Dependency and coordination in the grammar of Dutch and English', *Language*, **61**, 523–568 (1985).
19. Remo Pareschi and Mark Steedman, 'A lazy way to chart parse', *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, July 1987, pp. 81–88.
20. Ken Perlin and Eric M. Hoffert, 'Hypertexture', *Computer Graphics*, **23**, (3), 253–262 (1989).
21. Demetri Terzopoulos and Kurt Fleischer, 'Modeling inelastic deformation: viscosity, plasticity, fracture', *Computer Graphics*, **22**, (4), 93–101 (1988).
22. J. P. Lewis, 'Generalized stochastic subdivision', *ACM Transactions on Graphics*, **6**, (3), 167–190 (1987).
23. Robert Siegel and John R. Howell, *Thermal Radiation Heat Transfer*, McGraw-Hill, 1972.