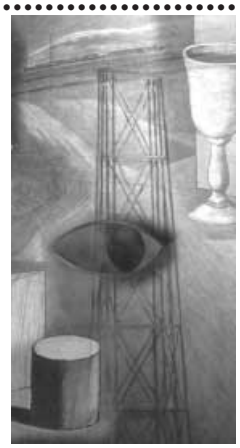
THE JOURNAL OF VISUALIZATION AND COMPUTER ANIMATION
J. Visual. Comput. Animat. **10,** 143–158 (1999)

3D Surface Cellular Automata and Their Applications



By Stéphane Gobron and Norishige Chiba*

This paper describes the generation and rendering of three-dimensional (3D) surface cellular automata (CA). Our model's main advantage is that it gives direct texturing simulation based on the actual shape of any triangulated input object. We first introduce general CA concepts and summarize works in the literature. We then describe our 3D surface CA method, emphasizing how it avoids potential problems in data structure and rendering steps. We then detail, two examples of specific 3D surface CA with their respective cell structures and corresponding computer graphics images. Copyright © 1999 John Wiley & Sons, Ltd.

Received April 1998; Revised February 1999

KEY WORDS: computer graphics; cellular automata; surface propagation; Voronoi diagram; patina; corrosion

Introduction

Purpose

In less than 15 years, computer graphics (CG) has become incredibly impressive. CG worlds have become 'too perfect'. In fact, compared with the actual world, with all its small details and imperfections, the clean CG worlds appear extremely uncomfortable and unrealistic. Texture-mapping techniques were devised to give CG a realistic, precise appearance, but have many limitations, such as: they are time consuming; difficulties occur in mapping orientation; scaling and resolution problems arise; and edges are often inconsistent. CG requires automatic textural effects that do not exhibit these limitations.

More recently, models for simulating dust accumulation,[1] appearance changed by water flows,[2] metallic patina,[3] and paint peeling[4] have shown very interesting results in generating realistic texturing. These techniques generated excellent results, but require a special data structure for each model, making it almost impossible to simultaneously render dust accumulation, patina painting and corrosion in the texture of a single object.

Finding no reports in the CG literature providing a general model, we propose an original 3D surface solution. Our model directly simulates texturing based on the actual shape of any triangulated input object. This natural approach avoids classical two-dimensional (2D) array-mapping problems.

What key points link the previously cited models? What do all objects have in common?

Cell and Cellular Automata

In all sciences, researchers have referred to Nature to find answers:

Everything in Nature—animate or inanimate—results from a set of interacting processes, e.g. the food chain or photosynthesis associated with respiration. Individual phenomena at first appear to exhibit complex behaviour. Further analysis shows that this behaviour results from a large number of simple interactions with limited capabilities. This makes Nature a complex set of homogeneous systems.

This important observation is the basis our work: Our model consists of a homogeneous set of interacting cells with limited capabilities, called cellular automata,[5] reviewed in Reference 6. On this strong basis, all flat and 3D texturing models—respectively applications 1 and 2—are available at a reasonable cost.

Background

Although the term 'cellular automaton' is rare in the CG literature, its theory has long been widely used in science,

*Correspondence to: S. Gobron, Faculty of Engineering, CG Department, Iwate University, Ueda 4-3-5, Morioka 020-8551, Japan. E-mail: stephane@cis.iwate-u.ac.jp

Copyright © 1999 John Wiley & Sons, Ltd. CCC 1049–8907/99/030143–16$17.50

and it is considered a vital tool in most simulation fields.

Here are some general references concerning CA: Reference 7—formal CA grammar/pure theoretical approach; References 5 and 8—theoretical computer engineering; References 9–11—CA image processing/fast ray tracing.

Previous work directly or indirectly addressing CA in CG can be divided into seven domains: texturing, fractals, semi dependent particle systems, 2D and 2·5D surface generation, 3D domain, texture generation using cellular interaction, and pure CA:

- At SIGGRAPH '92, Fowler *et al.* presented a very interesting article on seashell textures.[12]
- An impressive number of articles and books about the fractal domain have been published, e.g. References 13 and 14, or, for a theoretical basis, Reference 15 and, more recently, Reference 16.
- Particle systems have been widely explored in CG, especially for flow (liquid, gas) simulation. Owing to their natural structure, particle interactions, all are indirectly related to CA—particularly in the work of Takai *et al.*[17]
- Three subfields concern 2D or 2·5D surfaces. The first is terrain generation. Nagashima[18] and Chiba *et al.*[19] proposed erosion models for mountain scenery. The second is dynamic liquid surface simulation. Examples are a simple but efficient 2·5D liquid surface simulation by Kass and Miller[20] and, more recently, a simulation by Chen *et al.*[21] making possible real-time terrain modification. The third subfield is the non-photorealistic domain, concerning which an article was presented at SIGGRAPH '97 by Curtis *et al.*[22] in which the problem of water colour painting was treated by paper liquid absorption and pigment sedimentation.
- Real 3D CA publications are much more difficult to find. More or less in this domain, CG rendering of natural phenomena yields interesting articles on topics such as flames,[23] clay,[24] gas[25] and liquid[26] simulations.
- Much nearer to our domain is texture generation using cellular interactions. Even if it follows no regular grid, after stabilization, cells map a targeted surface quite regularly. However, intercommunication remains a very difficult problem to handle. Note also that our model requests no costly recursive particle stabilization. We recommend two works in this domain, one by Turk[27] on 2D (surface) textures and the other by Fleischer *et al.*[28] on 3D textures such as fur and thorn.
- We found only one reference addressing the CA problem directly in CG—a report by Thalmann[29] published in the mid-1980s that is highly innovative. The CA model is very simplistic, however, and the paper is mainly implementation oriented.

Surprisingly, we were unable to find anything on 3D CA texturing, so this paper can be considered as a first step into this unexplored but specific domain.

Overview

The paper is organized as follows. The next section outlines CA concepts, explaining why specific choices were needed to define our CA and showing our model's data structure.

The third section details intercellular communications—communications inside the input triangle and at the edge. It also describes our proposal's main drawback, exploring potential problems at triangle edges, quantifying them and showing how they can be reduced or even avoided.

The fourth section describes our rendering model consisting of a simple light model and a more complex rendering data structure.

The fifth section discusses two examples of CA. The first example proposes an original 3D surface Voronoi diagram construction based on regular probabilistic growth. This simple, intuitive model demonstrates whether our general model works well. The second example deals with a more complex 3D surface CA—copper corrosion and patina simulation. For both applications we include comments and diagrams on specific cell data and functions, possible structure and rendering problems, rendering statistics and CG results.

We conclude with a relatively long—but not exhaustive—list of potential research and expansion of this work.

Cellular Automata Model

In our CA model, CG applications consist of one or many objects to which an independent CA is applied. Each object is subdivided into triangles, which, for convenience, we call input triangles. Each input triangle is divided into a grid of identical cells.

General CA Structure

In conventional CG the basic structure of an object is defined as a set of more or less complex polygons. No

Copyright © 1999 John Wiley & Sons, Ltd.

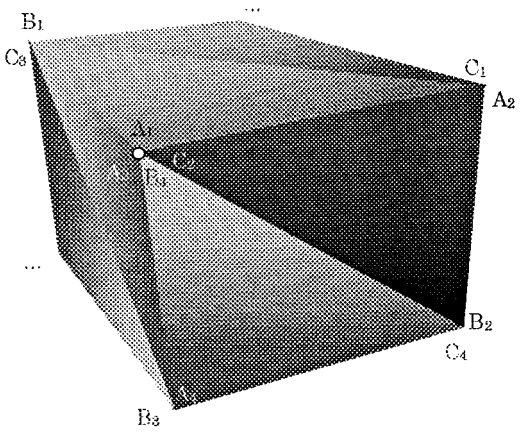


Figure 1. A cube, an example of a basic input object, defined by a set of triangles. The A_i, B_i and C_i vertices are described in the subsection on Triangle Division and Cell Properties. (Notice that this figure is related to Figure 20)

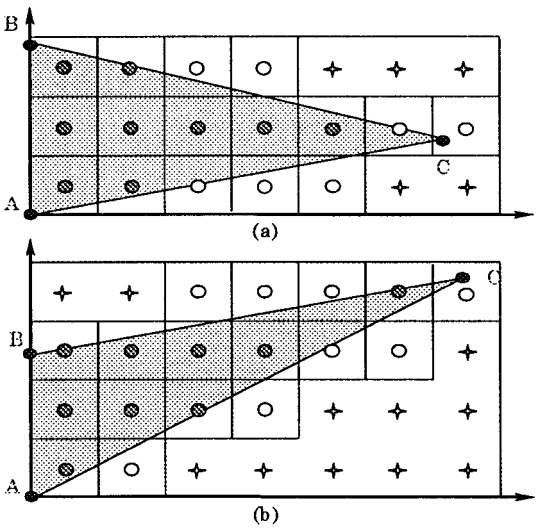


Figure 3. The only two possible cases for the projected triangles

matter how complex a polygon, it can always be subdivided into non null triangles. To prevent having a CA greatly increase the natural complexity of the object structure, we assume that input objects are in the form of an oriented triangular mesh (Figures 1 and 2a).

To generate a CA on the surface, this object must be divided into a large set of *identical* cells. By definition[5] the cells must be identical in geometry and also in properties (limited capacities).

Triangle Division and Cell Properties

We now need to answer the following question: How can we subdivide an input triangle into a set of squares? This can be done by projecting all triangles onto the same grid with a size equal to our square cell. Our solution is represented in Figures 3a and 3b. For convenience the projection has to be in such a way that the three corners A, B and C are such that A is always (0·0, 0·0), B (0·0, B_y) and C (C_x, C_y) with C_y=0·0. Figure 1 also shows these points on a 3D simple input object.

Because the geometry of the cell is sometimes in the input triangle, sometimes partially in and sometimes completely out, we need to add some new hypotheses in defining a cell in our CA model:

- The neighbourhood of a cell consists only of its eight surrounding cells (four sides and four corners).

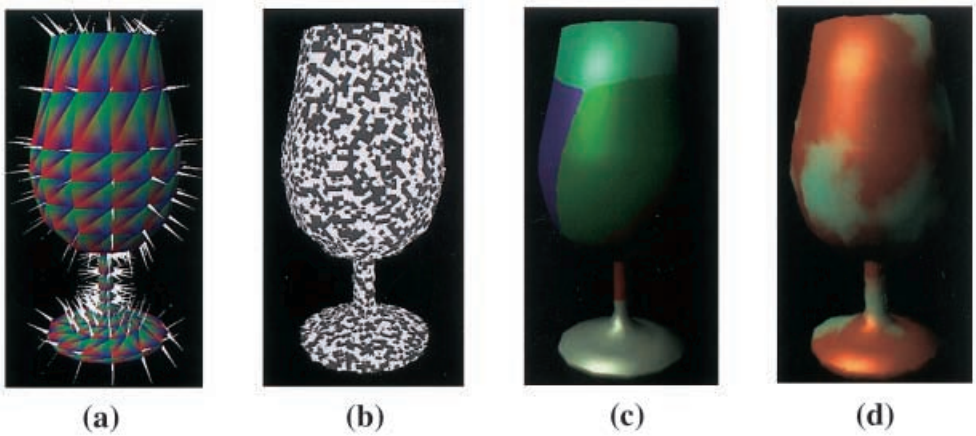


Figure 2. Same cup at different states: (a) and (b) data structures respectively—input object (subdivided into oriented triangles) with its orthogonal vertex vectors, and corresponding cellular structure (20 000 cells); applications—(c) Voronoi diagram and (d) copper patina

Copyright © 1999 John Wiley & Sons, Ltd.

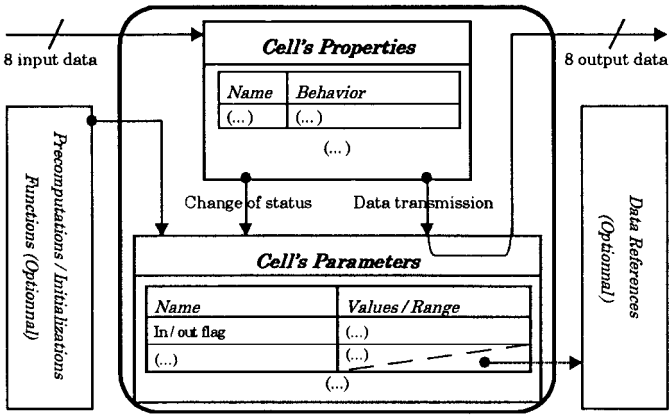


Figure 4. General structure of a cell

- A cell must have an *in/out* flag; the cell is considered as *active* in the CA when it is at least partially *in* the input triangle.
- Some cells can be completely out of the triangle (☆ in Figure 3). They must be considered as non-existent. In such cases we define the *in/out* flag as being *OUT-CELL*.
- Other very useful *in/out* flag positions are the *PART-IN* and *COMP-IN* positions. These states are only optional because they do not interact with the CA behaviour. They are, however, very useful in the rendering step. When the cell is *active*, its *-IN* flag position is determined by the geometric centre of the cell:
  - ■ *PART-IN* if outside the input triangle (○ in Figure 3);
  - ■ *COMP-IN* if inside the input triangle (◍ in Figure 3).

Figure 4 shows the general structure of a cell in our model.

As an example, Figure 5 shows the possible resulting projection of two neighbouring triangles, and Figure 2(b) shows the same projection's cellular structure but in 3D and using a real glass model (approximately 100 input triangles). In the following paragraphs, notice that Figures 5–7 follow the same pattern.

A major problem to solve is immediately apparent: from one triangle to another, some cells will not have eight neighbouring cells. How will the data propagation at edges be managed? The details of this problem are fully addressed in the following section.

Data Communication in CA

As shown, a cell has a limited capacity and can send and receive information only to and from its eight neighbours. How are data transmitted? What if the cell is at the edge of an input triangle? This section answers these questions and proposes a possible solution for data transmission through a cell set.

Communication Inside the Input triangle

In most cases, data transmission is within an input triangle. A special behaviour must be defined for each CA. (See the two examples of interior communication in the section on Applications and Results.)

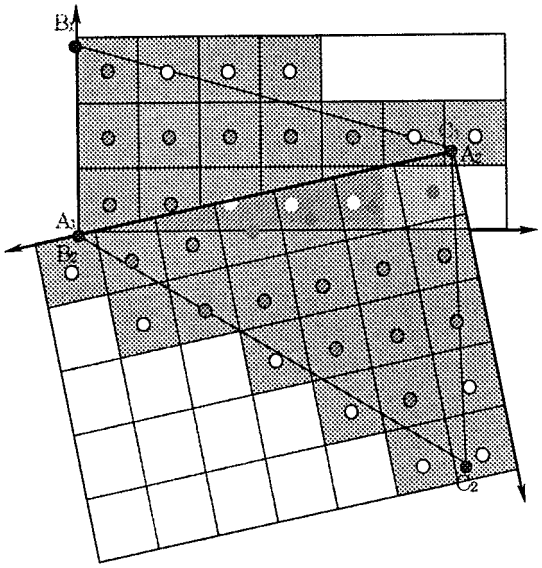


Figure 5. Two neighbouring triangles projected on the same grid size

Copyright © 1999 John Wiley & Sons, Ltd.

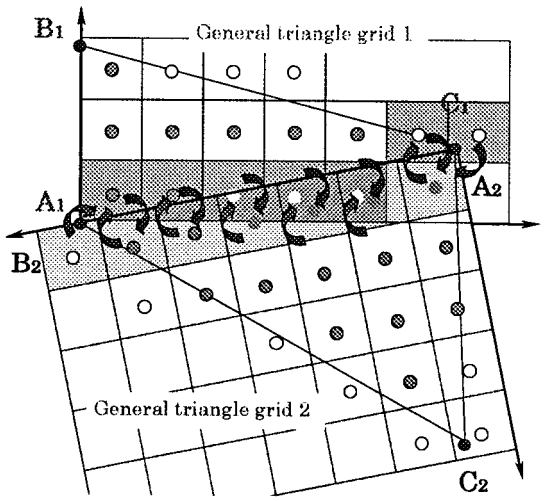


Figure 6. Edge data transmission

Edge Communication

The user cannot control information transmission based on two input triangles that have only one thing in common—one edge (Figure 6; 2D projection of two neighbouring input triangles). Edge data communication follows only one rule: edge cells are always in common. At the end of each turn, cell data from one edge are automatically transmitted to the neighbour's 'corresponding' cell.

Edge orientation is vital for maintaining correct data sequence during transfer, which is why we use vectors, not segments; that is, in this particular case the cells of *grid 1* are assigned to vector $\overrightarrow{C_1A_1}$ and the cells of *grid 2* to vector $\overrightarrow{A_2B_2}$.

Figure 7 shows how to find a corresponding cell (arrows). To transfer data, we look for the nearest projected cell of the opposite input triangle: When possible, we link each cell to the nearest point on the common segment using the projection of the cell's centre on the edge; otherwise, cells are linked to the nearest segment extremity corresponding cell.

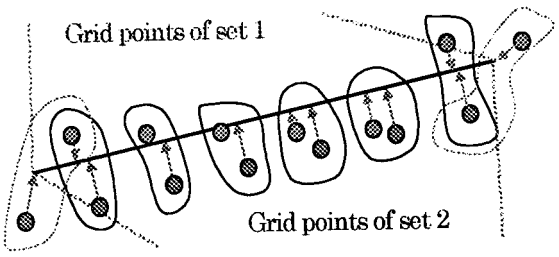


Figure 7. Edge data transmission

A set of pairs is generated automatically. To accelerate the computation of this algorithm, these sets of pairs can be precomputed and stored instead of being recomputed. Memory cost is relatively high (see the subsections on Rendering Statistics). Notice that at both ends an error occurs—some cells will receive information twice and others will only send data. This is discussed below.

Comments on Data Transfer Problems

An error is often generated during data transfer on edges. The problem previously described happens most frequently at extremities (input triangle vertices). Experiments and practical observations prove this effect negligible if the grid unit size is small enough.

This error is proportional to the difference in orientation of the two projected grids, which is why it follows the function 1-cos$x$ in the interval [0, $\pi/4$] (Figure 8). We obtain the average error

$$\varepsilon_{\text{average}} = \frac{1}{\pi/4}\int_0^{\pi/4}(1-\cos\alpha)\,d\alpha = 1-2\frac{\sqrt{2}}{\pi} \approx 9{\cdot}9\%$$

Clearly, a 10% error appears considerable, but we must not forget that this is in a scene where input triangles are purely randomized and does not concern the complete cellular model, but partial cells in a large minority. Note that if the input triangles are well organized (making the projected grid continuous), data transmission does not become redundant and therefore the problem disappears.

The second point concerns cell size. During initial experiments, very strange CA behaviours were observed when using very large cells—proportionally to input triangles. The following example explains how to avoid this:

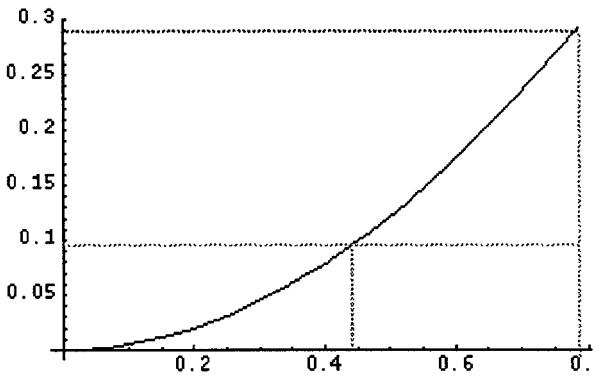


Figure 8. Function 1-cosx between 0 and π/4

Copyright © 1999 John Wiley & Sons, Ltd.

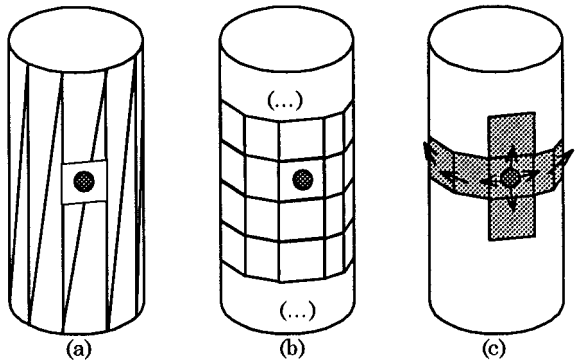


Figure 9. Worst case—long, thin triangles using too large a cell size

The worst scenario would be a scene composed entirely of a series of long, thin triangles, i.e., a very fine cylinder. Figure 9 shows the problem if we apply a very inappropriate CA such that the cell size will be equal to or bigger than the width of an input triangle.

(a) The first figure shows the approximation of a cylinder composed of long, thin triangles. The dot is the random initial seed and the surrounding square shows the cell's inappropriate size.
(b) Each triangle is decomposed into sets of square cells. To make the figure clear, each set of cells is simplified here. As two triangles are needed for each 'side' of the cylinder, in reality decomposition implies twice as many cells as are represented.
(c) After only one step, instead of having a state change in only cells neighbouring the seed, all the cells forming a ring around the cylinder appear to have a change of state. This is due to the fact that the input triangles share directly neighbouring cells.

The conclusion that can be drawn from these two points is that for each input triangle in the scene, if the cell size is smaller than all triangle sides, then only minor communication problems will occur (at input triangle vertices). We can also add that as the cell size decreases linearly, this phenomenon very quickly becomes negligible.

Figure 10 demonstrates graphically that using a sufficiently small grid, and even if the two grids have the worst difference in orientation, only very slightly visible problems occur.

Rendering Using OpenGL

Our light model is very simple. However, since a large number of 3D cells make scene representation quite complex, a special data structure for rendering is required. We also try to keep this rendering as simple as possible, remembering two important aspects:

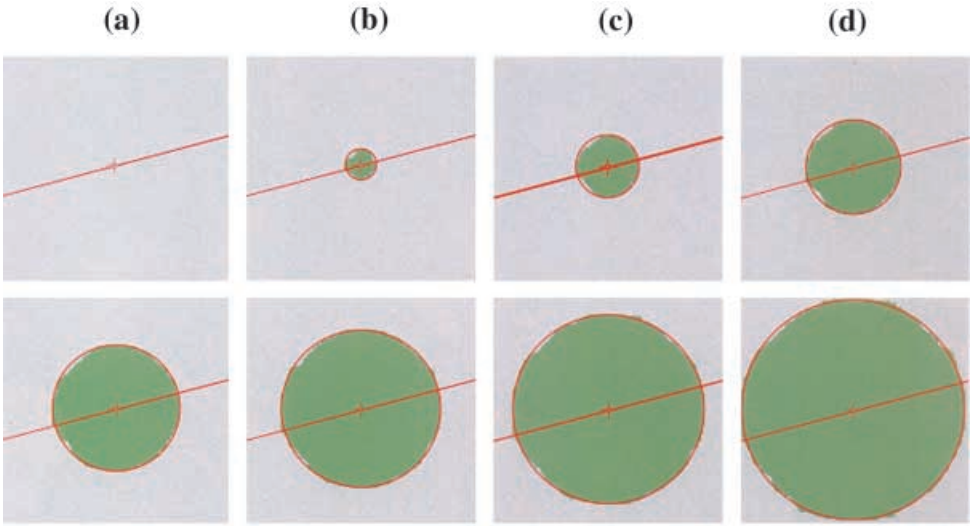


Figure 10. Experimental progression of the 'systematic propagation' for the Voronoi diagram applications (in green). As the initial seed point (top-left image) is very near the edge (line in red), the data are spreading into two input triangles at the same time. To show the quality of the generated circle, the ideal circle (also in red) is shown. The first green disc (a) generated by this technique has a radius of only 10 pixels and the surface is 87% completed, and the last green disc (h) has a radius of 70 pixels with 99.9% completion

Copyright © 1999 John Wiley & Sons, Ltd.

- To maintain greater flexibility we created our own light model.
- Data from each cell in the CA must be displayed so that no information is lost. We therefore use OpenGL™,[30] because it provides very fast and convenient rendering structures applicable for our CA: Triangle Strips (TS) and Triangle Fans (TF).

In the following subsection we describe the simple light model applied to each cell, and then present the special arrangement applied for OpenGL.

Simple Light Model

In this light model our resulting images prove that it is possible to have both good results and a very simple approach. Our light model uses the classic ambient, diffuse and specular properties. We have

$$C_{[RGB]}=(K_a+K_d(\&wL \,.\, \&wN_c)+K_s(\&wR \,.\, \&wE)^n]C_{m\,[RGB]}$$

Where

- the ambient term K_a is simply a constant;
- the diffuse term K_d is multiplied by the dot product of the light vector $\&wL$ and the cellular normal vector $\&wN_c$;
- the specular term follows Phong's specular model[31] using the coefficient K_s, and where $\&wR$ is the light reflection vector, $\&wE$ is the camera (or 'eye') orientation and n is the specular power of the material;
- finally, C_m represents the initial material colour.

In the second example of CA, presented later, we will see that some special applications require multilayered rendering. In such cases the appearance of the surface is the result of light interactions within layers.[32] A special rendering model is then required to produce good-quality images. As this problem is not directly linked with our CA model, it will be treated as an example in the case of double layers (see the subsection on Corrosion and Patina Generation).

Triangle Strips and Triangle Fans

In this section we will show why and how we used OpenGL Triangle Strips (TS) and Triangle Fans (TF) as data structures for rendering.

On the one hand, as previously shown in Figure 3, the information contained in each cell is potentially 'situated' at the centre of each patch (rectangular cell). The possible geometry of the final colourings of one input triangle is then very different from its basic triangular geometry definition (three vertices).

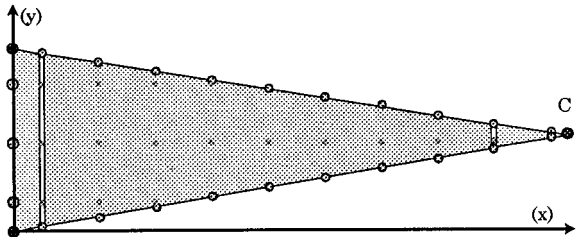


Figure 11. Three regions (yellow, green and blue) subdividing input triangles

On the other hand, for the data structure of the rendering, OpenGL provides very convenient and fast tools such as TS and TF, which are very helpful for rendering each input triangle with the cell's colouring.

Our goal in this section is now defined as reconstructing one input triangle with a set of triangle bands (TB=TS or/and TF) in such a way that TB vertices correspond as much as possible to the cell centre. To achieve this, the two main steps are as follows.

(a) Define the contour of the triangle by the three vertices A, B and C and also three series of intermediate points. These intermediate points are generated by the intersection of the centred cell-grid line and the three triangle edges. As the corresponding cells of these points are only partially in the triangle, their colourings are computed with a convolution method.
(b) Divide each triangle into three regions (Figure 11).

- The first band is always defined as TB and its size is half a grid unit.
- The intermediate region contains a set of similar TB (identical thickness).
- The last region depends on the geometric position of the point C. If $C \,.\, x$ is smaller than or equal to the half cell, then this region is only one TF; otherwise it has to be decomposed into one TB and one TF.

The TB construction complexity depends on the triangle orientation. Two types of triangles can be categorized.

- Figure 12(a) presents a 'long triangle' ($C \,.\, x \geq B \,.\, y$). This type of triangle does not generate any particular problem; the first TB on the left and intermediate regions will always be composed of TS, and the last region will be a TF and sometimes (depending on $C \,.\, x$) a TS.
- We can see that the reconstitution of the input triangle for the 'thin triangle' (when $C \,.\, x < B \,.\, y$) is less obvious

Copyright © 1999 John Wiley & Sons, Ltd.

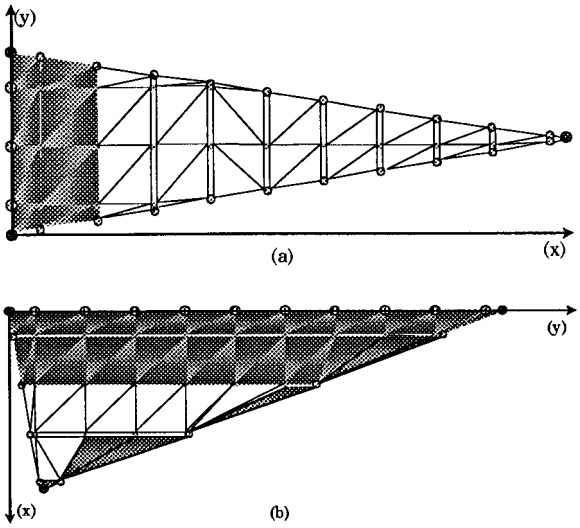


Figure 12. Two possible types of input triangle to recompose

(Figure 12(b); notice that the axes are not presented in the same order). Actually, when the two sides' difference in TB point number is greater than one, additional TF are needed (one at the top and/or bottom) to complete the middle TS.

Applications and Results

To show the graphical capabilities and especially the strong potential variety of applications that our model is able to produce, two different applications are presented in detail.

The first introduces a simple cellular structure for generating an original solution of the 3D surface Voronoi diagram.

The second application describes a more consistent CA where each cell has a limited but potentially complex behaviour.

All results (CA data propagation and graphical) were computed and rendered on an SGI Indigo2™ workstation with 175 MHz CPU R10000 and 128 MB RAM.

3D Surface Voronoi Diagram

One original application of our system is the automatic generation of 3D surface Voronoi diagrams (VD) based on regular probabilistic growth (Figure 13).

We have found this application particularly appropriate: first, it has the advantage of being well known, simple to model and intuitively easy to understand; but even better,

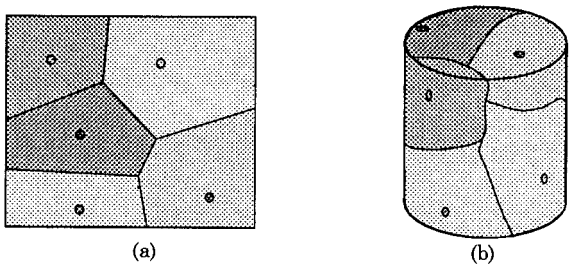


Figure 13. (a) 2D classical VD and (b) possible cylindrical VD

as its propagation needs to be regular, it naturally tests if our 3D surface cellular model is working well. For a complete explanation of the subject we recommend Preparata and Shamos's book.[33]

We will show first the internal data communication model, then the very simple cell structure, and finally we will make some brief comments on the resulting CG images.

VD internal data communication model. The principle is very simple. First, an initial number of seeds are randomly selected before any CA computations, and no new seed will be created later. Then we assign to each seed a different flag (which can be interpreted in the rendering step as a different colour). As the progression must be regular, the resistance of all cells is set to null. Probably most difficult is that the change of states of the cell must be set such that a 'regular' circle is obtained. For this purpose our method is as follows.

Propagation is systematic (100%) on the four direct sides, and a certain percentage P on the diagonal must be set. We intuitively show, using geometry (Figure 14), how we determined this diagonal probability. Note that d stands for the corresponding distance in red.

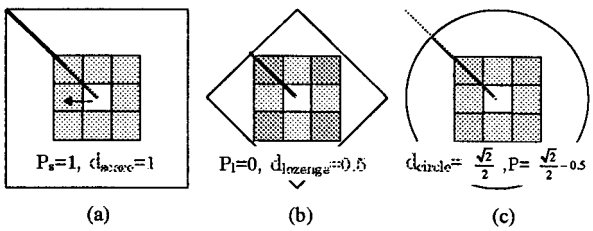


Figure 14. With two known diagonal percentages (P_s and P_l), two basic geometric shapes can be generated: a square and a regular lozenge. Then, assuming distances (d_s, d_l and d_c) and percentages are linearly related, we find a good approximation of the circle diagonal percentage: (a) a square is obtained if P is set to 100% (blue squares); (b) we obtain a regular lozenge if P=0% (orange); (c) then, to obtain a circle, the green percentage must be:

$$P = d_{circle} - d_{lozenge} = \sqrt{2}/2 - 1/2 \approx 20{\cdot}71\%$$

Copyright © 1999 John Wiley & Sons, Ltd.

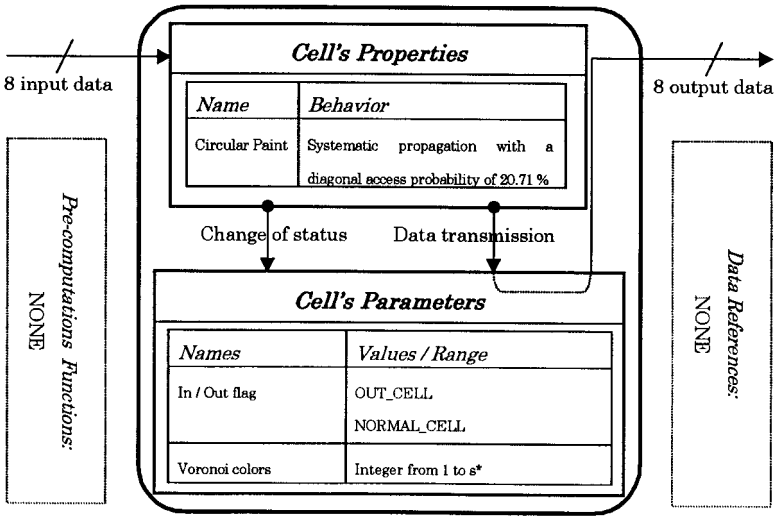


*Figure 15. Voronoi diagram cell structure (*s, number of seeds)*

3D surface Voronoi diagram results. Figure 15 summarizes the applied cell structure.

Resulting CG example of images. With these parameters an optimal circular propagation is obtained through the CA (refer to Figure 10). The entire surface is assigned very rapidly to one of the initial seed flags (Figure 16).

Figure 17 shows the regular progression in a very simple object composed of less than 100 input triangles but having a total cell number greater than one million. An important observation is that the generation exhibits a typical 2D CA quadratic growth, providing a very short computational time (Figure 16). We can also notice that this progression is very similar to the exponential law of population growth.[34] However, the logistic curve has no central symmetry, showing a facility for the seed to grow at the beginning (we will see later that this is not always the case).

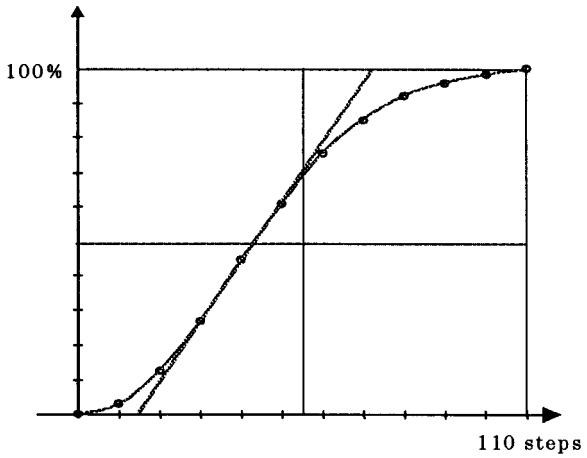


Figure 16. Corresponding data spreading of Figure 17

In 2D it can be demonstrated[33] that only a few points connected by segments define a 2D Voronoi diagram. In 3D the problem is not that simple. Our model shows (Figure 2(c)) also that a 3D Voronoi diagram has to be defined as a set of 3D points connected by a curvilinear segment—which can turn out to be impossible to derive due to sharp changes of direction.

Rendering statistics

- input triangles: 50.
- Number of cells: approximately 250 000.
- Total computational time (CA and rendering): less than 10 min.
- Number of CA steps: 110.

Corrosion and Patina Generation

One typical example of surface propagation is corrosion.[35] A cursory investigation will show that the corrosion is only at the object surface and generates random spot patterns; but in fact, after a longer period of observation, we realize that the corrosion not only propagates on the surface but also inside the material itself. Furthermore, these spots are not purely randomized, but more or less follow the path of water flow.[36] Secondly, the corrosion always makes regular circles if the region is continuously subjected to contamination. We may also notice that even if the spread of corrosion appears to be identical, for each metal a different kind of pattern exists.[37]

Copyright © 1999 John Wiley & Sons, Ltd.

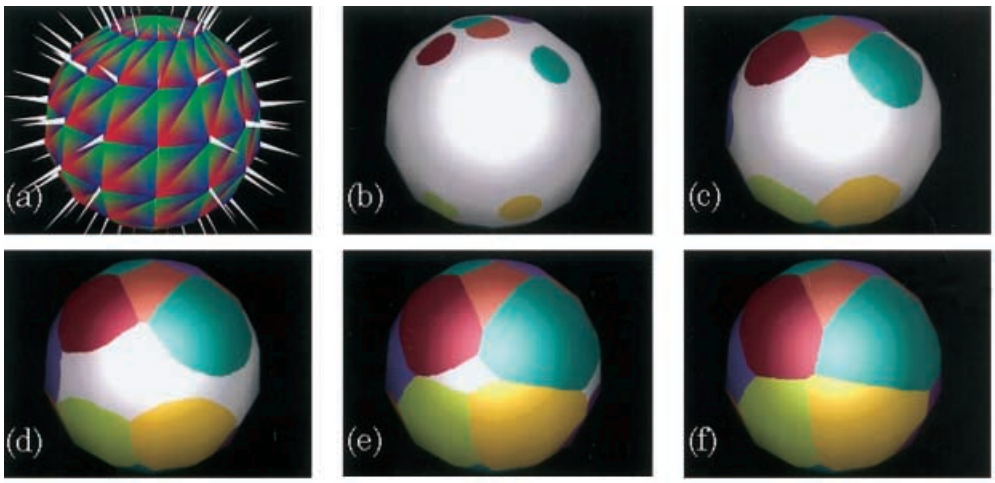


Figure 17. 3D surface Voronoi diagram: (a) image represents the input list of input triangles with their normal vertex vectors in white; (b)–(f) images show the systematic circular propagation. More than 250 000 cells are represented and the total computational time was less than 10 min

For the description of this model we will present first the corrosion cell structure, and then the special type of internal data communication. This will be followed by the proposed solutions for rendering problems and finally a description of the resulting images on a copper cup.

Corrosion cell structure. Figure 18 summarizes in detail the cell structure needed for simulating corrosion with our CA model.

The main difference from the previous model is apparent, particularly the fact that in this model both previous computations and material reference fields are needed to complete the definition of the cell's parameters.

Only one property (defining the cell's behaviour) is specified. We call this *strategic data propagation* and it is described in the following subsection.

Internal communication for corrosion CA model. The 'strategic' (attack/defence) cellular behaviour uses as support the natural grid points of the data structure and a simple rule of strategy. This model has been found very appropriate[3] for simulating corrosion.

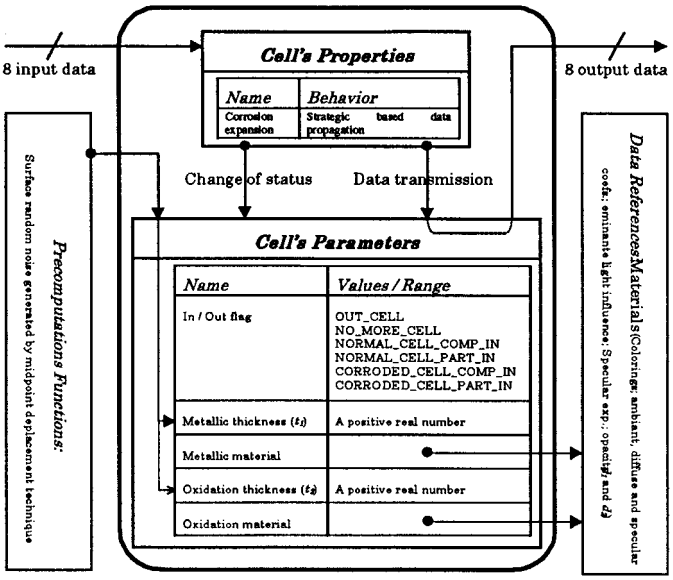


Figure 18. Details of the corrosion CA cell structure

Copyright © 1999 John Wiley & Sons, Ltd.

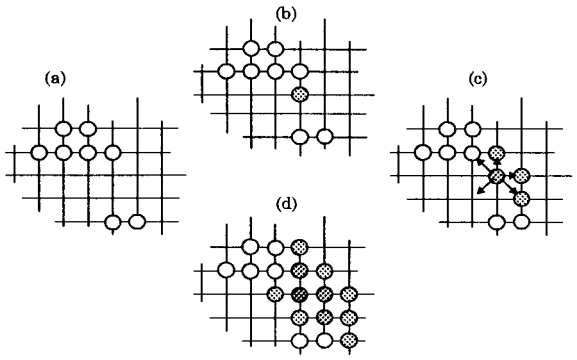


Figure 19. Surface propagation strategy—main steps

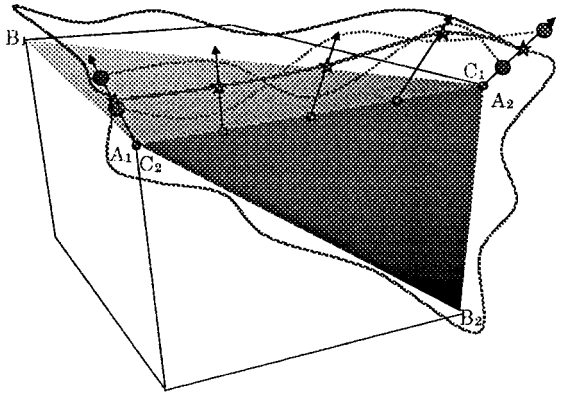


Figure 20. Input triangle edge noise problem (Figure 1)

Figure 19 summarizes the main steps of the procedure.

(a) The first step often consists of determining resistant regions called 'strong cells'. The selection of these regions can be randomized or can be based on a precomputed surface noise. For each CA a different choice is possible. In the case of corrosion a more sophisticated technique would consist of making the selection proportional to a precomputed general water path mask. As this forms part of our current research, the method will be explored in a future paper.

(b) The second step consists of finding new potential seeds. Two parameters control this step: the percentage of new seeds per unit of time and the maximum number of new seeds.

(c) The third step determines the data transfer from one cell to another or to many others (up to a maximum of eight). Data have a 'chance' of being transmitted. This probability of state change can be controlled by the following parameters:

- the 'age' of the offensive cell (*older is stronger*);
- potential help of the defence ('strong' material cells);
- the maximum number of accessed cells can vary from one to seven (and not eight);
- to make pseudo spot regions, we adjust the eight cardinal directions using a 3×3 Gaussian probability matrix.

The arrows shown in Figure 19(c), describe one of the possible data transmissions resulting here (as an example) in five data transfer attempts and only three successes.

The combination of these three steps allows many interesting effects.

(d) This figure shows one of the following possible steps where we notice the rapid propagation in the potentially weak regions.

For mathematical details of the transition model see Appendices 1 and 2.

Problems/solutions for rendering corrosion

Structure problem: edge thickness correspondence. When the corrosion is very thin, we can simplify the rendering by assuming that its thickness is null. However, when the object happens to be very corroded, the thickness of the metallic layer can seriously decrease (but we do not consider the holes generated). Moreover, the patina accumulation creates some strong visual effects. That is why we decided not to approximate the thickness.

To achieve this, the main problem is once again on the edges: the thickness of one edge input triangle is not always identical to that of its direct neighbour. Figure 20 presents the problem with two input triangles (red and blue). The lines represent the final 3D surface. At the edges we can see the real surface shown as dotted lines. The average of these dotted lines (stars) generates the future edge to render (dotted line).

A criticism of this technique is that the possible high frequency noise of the input triangles will have a strong loss of variance, which upon close inspection reveals unnatural edges. However, this occurs only very rarely.

Rendering problem: light through two layers. As we stated previously, we try to keep a simple and fast rendering method. In the case of corrosion a patina layer rises to the top of the metallic structure. Obviously, this implies a more complex light behaviour.

Because of the high quality of its results, the Kubelka–Monk[32] light-through-layers general model has enjoyed much success in recent CG publications.[3,6] However, owing to the presence of two 'unknown' parameters (K, S),

Copyright © 1999 John Wiley & Sons, Ltd.

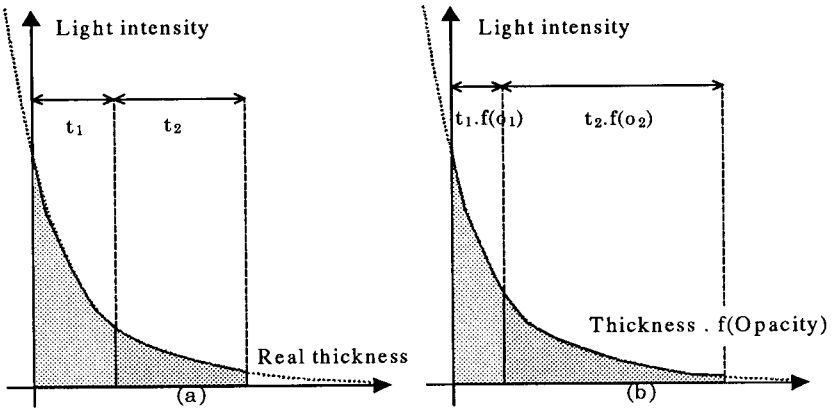


Figure 21. Curve of e^{-x} function showing (a) only the thickness of the two layers and (b) multiplied by a function of their respective opacities

this method is not very convenient. We will show that a general and complex model is not always necessary to achieve natural results.[38]

A good question to consider is: *what is the special behaviour of light when it goes through layers of material?* A simple approach is to answer: *its intensity decreases exponentially, depending on the material's opacity*.

Therefore the colour resulting from the light going through two layers is only a function of the respective thickness, opacity and material colour using the e^{-x} function (Figure 21).

An important point is that we must first modify the original material colours of the two layers according to the light model, and then compute the final colour by our light-through-layers method. This appears to be crucial for the image quality if the metal is naturally very shiny (most metals are) and the patina thickness is very small (also true in most cases).

Figure 21 shows the light intensity dependent upon the two-layer thickness. In (a) the two layers have the same density, but (b) shows the same layers with different densities. We can easily derive—using surfaces interpreted by integrals—the following mathematical formula for the cellular colour using double layers:

$$c_{\mathrm{rgb}} = \frac{l_{c_1}\int_0^{t_1 f_{o_1}} e^{-x}dx + l_{c_2}\int_{t_1 d_1}^{t_1 f_{o_1}+t_2 f_{o_2}} e^{-x}dx}{\int_0^{t_1 f_{o_1}+t_2 f_{o_2}} e^{-x}dx}$$

where l_{c_i} is the result of the light function $l(x)$ on a material colour c, with thickness t and the function $f_o = 1/(1-o)$ (o standing for opacity) of the two layers. After integration, development and regathering of the exponential term, we simply obtain

$$c_{\mathrm{rgb}} = \frac{l_{c_1} + (l_{c_1} - l_{c_2})e^{-t_1 f_{o1}} + l_{c_2} k}{1+k}$$

with $k = -e^{-(t_1 f_{o1} + t_2 f_{o2})}$.

This approach has the advantages of both being simple and giving good results, and coefficients are easy to find where layer thicknesses are known and one's material density is easily determined.

Corrosion CA graphical results. Figure 22 presents the progression of copper corrosion and its corresponding typical green patina over a cup composed of less than 450 input triangles and over 220 000 cells.

Its progression seems also to be proportional to the exponential law of population growth, and the logistic curve also has no central symmetry (Figure 23). The first observation we make is that data saturation occurred many times faster (the systematic propagation used for a 3D Voronoi diagram was about nine times faster). The second observation is that in this case the seeds have very strong propagation difficulties at the beginning.

Rendering statistics

- Input triangles: approximately 450.
- Number of cells: approximately 220 000.
- Total computational time (CA and rendering): less than 20 min.
- Number of CA steps (up to Figure 22(f)): 950.

Implementation of Other Models

We can easily transfer all models that work on a 2D regular grid onto a 3D surface cellular model. Again, the main problem is in designing the cell data structure and functions.

Copyright © 1999 John Wiley & Sons, Ltd.

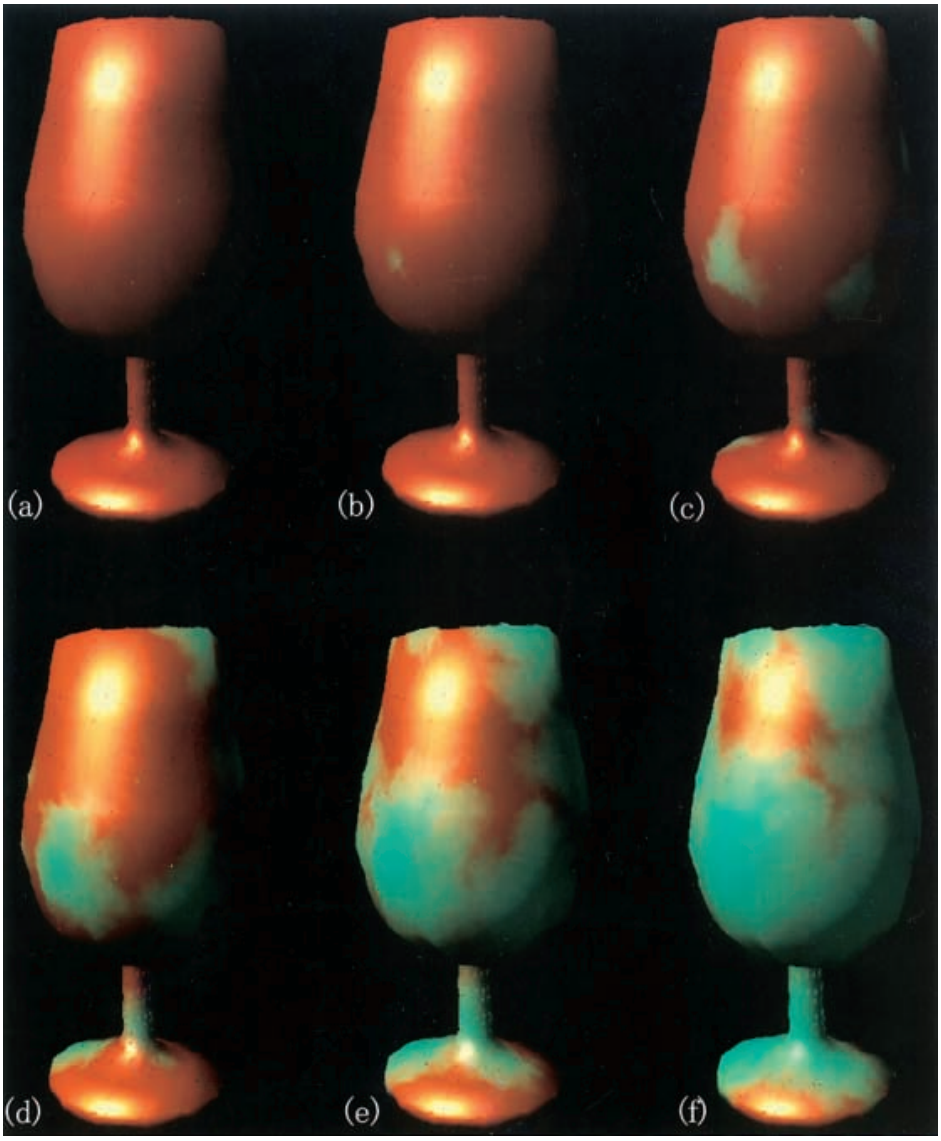


Figure 22. Propagation of typical green patina on a copper cup (220 000 cells, total computational time less than 20 min)

Conclusions and Future Work

We presented an efficient method for representing 3D surface cellular automata (CA). We explained the need for the input to be subdivided into triangles and square cell shapes. We presented data propagation through the CA, adequately solving the main problems: triangle edge-to-edge cell communications and potential orientation errors.

To prove the model's flexibility, two applications were described—along with possible weak points and how to avoid them. The first application showed how simple it is to make a good approximation of the Voronoi diagram on any type of 3D surface using an original probabilistic approach. Moreover, it permits us to verify that even if communication redundancy occurs, our CA model is consistent. The second application demonstrated that our CA model enables us to realize more advanced models, such as a simple but efficient, realistic corrosion model. For these two models a series of rendered CG images was presented and discussed.

Many interesting aspects still remain for further improving 3D CA.

A relatively direct task would be to search for an extension of our model for symmetrical and fractal

Copyright © 1999 John Wiley & Sons, Ltd.

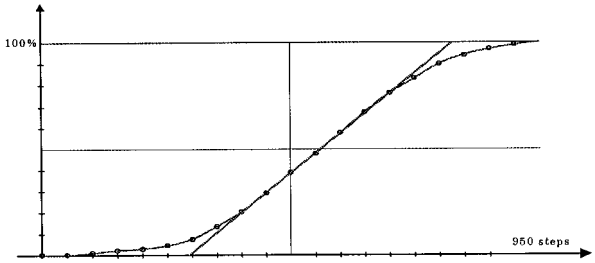


Figure 23. Percentage of cell change of state (using pure random strong cell selection) in a scene composed of one simple object

texturing. Automatic image mapping would seem a good extension of our model.

Another key area for further work is the CA data structure itself. At this point the data structure defines only one layer. An extension to n layers for each different set of cells would be an interesting research topic. The attribution of different layer behaviours, such as advanced corrosion, plastic and ceramic cracks, or paint peeling, would exponentially extend the palette choices for objects.

Finally, we can imagine linking this 3D surface CA model with non-grid based models, such as a water flow system, to generate (as Dorsey *et al.*[2] proposed) realistic corrosion and the corresponding patina paintings.

ACKNOWLEDGMENTS

We especially thank Norihiro Nasukawa for his system assistance and daily services. Thanks are also due to Corinne Kapel, Karen Mortin, Celia Takachi and Leroy Garciano for their proofreading of the draft.

Appendix 1. State Transition Model

Let us define C as the set of cells of any main triangle of the CA, C_x being one of its elements.

Corroded Cells

- Assume that the initial corroded cells are predetermined randomly or by the user.
- Let $S_{x,y}(t)$ be the y state parameter of any C_x cell at position x at any step time t:

$$\forall i | S_{i,\mathrm{b}}(t) = \text{corroded} \rightarrow \begin{cases} S_{i,\mathrm{mt}}(t+\Delta t) = S_{i,\mathrm{mt}}(t) - k_1 \\ S_{i,\mathrm{ot}}(t+\Delta t) = S_{i,\mathrm{ot}}(t) - k_2 \end{cases}$$

Considering

$$\forall i | S_{i,\mathrm{mt}}(t) \leqslant 0 \rightarrow S_{i,\mathrm{b}}(t) = \text{non-existent}$$

$$\forall i | S_{i,\mathrm{ot}}(t) > k_3 \rightarrow S_{i,\mathrm{ot}}(t) = k_3$$

with 'b' the cell behaviour, 'mt' the metallic thickness, 'ot' the oxide thickness and k_1 and k_2 user-defined constants.

Corrosion Propagation

- We assume that the corrosion is propagated by corroded cells:

$$\forall i | S_{i,\mathrm{b}}(t) = \text{corroded} \rightarrow \exists n_{\mathrm{vc}} | n_{\mathrm{vc}} := V(N(C_i))$$

with V the 'number of valid (existing and non-corroded) cells' function and N the 'neighbour of' function.
- Let n_{a} be the number of 'attacked' cells:

$$n_{\mathrm{a}} := f_1(A(S_{i,\mathrm{b}}(t)))$$

with A the 'Age of corroded cell' function (proportional to the oxide thickness) and f_1 a user-defined function, *e.g.* f_1 linearly increasing, with range $[1, n_{\mathrm{vc}}]$.
- Then

$$\left.\begin{array}{c} \forall C_i \in N(C_o) | i \in [1, n_{\mathrm{a}}] \\ \forall k \in N(C_i) \end{array}\right\}$$

$$\rightarrow S_{i,\mathrm{b}}(t+\Delta t) := f_2(C_i(t), set(C_k(t)), C_o(t))$$

f_2 determines whether or not the C_i cell behaviour has to be set to corroded. It first determines the C_i cell defence (depending on the neighbourhood cell, i.e. $set(C_k)$, behaviour) and then the C_o attack (proportional to its age). Finally, if the attack is greater than the defence, the C_i cell becomes corroded.

Appendix 2. General Algorithm for Our CA Model

1. Verify completeness of object (Figure 2(a)).
2. For each triangle, generate oriented projections on identical cell grid (Figure 2(b)).
3. Produce surface noise (if requested by cell definition).
4. Create 'strong' and 'weak' regions (if requested by cell definition).

Copyright © 1999 John Wiley & Sons, Ltd.

5. Define the individual cell behaviour definition for this surface CA (user).
6. Generate CA up to (user's choice):
   - no more possible data transfer through CA (Figure 17);
   - or a specific number of iterations (Figure 22);
7. Iterate one step of data transfer through the CA.
8. Record animation (*optional*).

References

1. S. C. Hsu and T.-T. Wong, 'Simulating of dust accumulation', *IEEE Computer Graphics and Applications*, **15**(1), 1 (1995).
2. J. Dorsey, H. K. Pedersen and P. Hanrahan, 'Flow and changes in appearance', *Computer Graphics Proceedings (SIGGRAPH '96)*, 411–420 (1996).
3. J. Dorsey and P. Hanrahan, 'Modeling and rendering of metallic patinas', *Computer Graphics Proceedings (SIGGRAPH '96)*, 387–396 (1996).
4. T.-T. Wong, W.-Y. Ng and P.-A. Heng, 'A geometry dependent texture generation framework for simulating surface imperfections', *EUROGRAPHICS '97*, 1997.
5. P. P. Chaudhuri, R. D. Chowdhury, S. Nandi and S. Chattopadhyay, *Additive Cellular Automata Theory and Applications VI*, IEEE Computer Society Press, New York, 1997.
6. E. F. Codd, *Cellular Automata*, Academic Press, New York, 1968.
7. A. Rosenfeld, *Picture Languages*, Academic Press, New York, 1979.
8. T. Toffoli and N. Margolus, *Cellular Automata Machines*, MIT Press, Cambridge, MA, 1987.
9. M. Ishii, H. Sato, K. Murakami, I. Ikesaka and H. Ishihata, 'Cellular array processor cap and applications', in *International Conference on Systolic Arrays*, IEEE Computer Society Press, New York, 1988, pp. 535–544.
10. K. Preston, M. J. Duff, S. Leviadi, P. E. Norgren and J. I. Toriwaki, 'Basics of cellular logic with applications in medical image processing', *Proceedings of the IEEE*, 67 (1979).
11. S. R. Sternberg, *Language and Architecture of Parallel Image Processing*, North-Holland, Amsterdam, 1980, p. 35.
12. R. Fowler, H. Meinhardt and P. Prusinkiewicz, 'Modeling seashells', *Computer Graphics Proceedings (SIGGRAPH '92)*, **26,** 379–387 (1992).
13. S. Ishizaka, Y. Kato, R. Takaki and H. Toriwaki, 'Science on form', *Proceedings of the First International Symposium for Science on Form*, Tokyo, 1985.
14. J. L. Encarnacao, H.-O. Peitgen, G. Sakas and G. Englert, *Fractal Geometry and Computer Graphics*, Springer, New York, 1991.
15. E. H. Stanley and N. Ostrowsky, *On Growth and Form*, Martinus Nijhoff, The Hague, 1986.
16. A. J. Kaandorp, *Fractal Modeling Growth and Form in Biology*, Springer, New York, 1994.
17. Y. Takai, K. Ecchu and N. K. Takai, 'A cellular automaton model of particle motions and its applications', *The Visual Computer*, **11,** 240–252 (1995).
18. K. Nagashima, 'Computer generation of eroded valley and mountain terrains', *The Visual Computer*, **13,** 456–464 (1997).
19. N. Chiba, K. Muraoka and K. Fujita, 'An erosion model based on velocity fields for the visual simulation of mountain scenery', *The Journal of Visualization and Computer Animation*, **9**, 185–194 (1998).
20. M. Kass and G. Miller, 'Rapid stable fluid dynamics for computer graphics', *Computer Graphics*, **24**(4), 49–57 (1990).
21. J. X. Chen, N. Da Vitiria Lobo, C. E. Hugues and J.-M. Moshell, 'Real-time fluid simulation in a dynamic virtual environment', *Proceedings of the IEEE*, **17**(3), 52–61 (1997).
22. C. J. Curtis, S. E. Anderson, J. E. Seims, K. W. Fleischer and D. H. Salesin, 'Computer-generated watercolor', *Computer Graphics Proceedings (SIGGRAPH '97)*, 421–430 (1997).
23. T. Agui, Y. Kohno, T. Nagao and M. Nakajima, 'Generation method of 3-dimensional flame images using cellular automata', *The Transactions of IEICE, D-II*, **J75,** 1747–1749 (1992).
24. H. Arata, Y. Takai and T. Yamamoto, 'Visual clay modeling based on cellular automata', *IPSJ CG Workshop*, **97**(124), 19–24 (1997) (in Japanese).
25. N. Foster and D. Metaxas, 'Modeling the motion of a hot, turbulent gas', *Computer Graphics Proceedings (SIGGRAPH '97)*, 181–188 (1997).
26. N. Foster and D. Metaxas, 'Realistic animation of liquids', *Graphics Interface '96*, 1996, pp. 204–212.
27. G. Turk, 'Generating texture for arbitrary surfaces using reaction–diffusion', *Computer Graphics Proceedings (SIGGRAPH '91)*, **25**(4), 289–298 (1996).
28. K. W. Fleischer, D. H. Laidlaw, B. L. Currin and A. H. Barr, 'Cellular texture generation', *Computer Graphics Proceedings (SIGGRAPH '95)*, pp. 239–248 (1995).
29. D. Thalmann, 'A "lifegame" approach to surface modeling and rendering', *The Visual Computer*, **2,** 384–390 (1986).
30. J. Neider, T. Davis and M. Woo, *OpenGL Programming Guide, the Official Guide to Learning OpenGL™*, Addison-Wesley, Reading, MA, 1993.
31. J. D. Foley and A. van Dam, *Fundamentals of Interactive Computer Graphics*, Addison-Wesley, Reading, MA, 1984, pp. 575–587.
32. P. Kubelka and F. Munk, 'Ein Beitrag zur Optik der Farbanstriche', *Zeitschrift für Technische Physik*, **12,** 593 (1931).
33. F. P. Preparata and I. M. Shamos, *Computation Geometry, an Introduction*, Springer, New York, 1985, pp. 198–227.
34. E. J. Purcell and D. Varberg, *Calculus with Analytic Geometry*, 5th edn, Prentice-Hall, Englewood Cliffs, NJ, 1987.
35. M. G. Fontana, *Corrosion Engineering*, 3rd edn, McGraw-Hill, New York, 1987.
36. S. Gobron and N. Chiba, 'Visual simulation of corrosion', *Workshop of Tohoku '97*, Morika, 1997, Ref. 97-3-9.
37. R. Hugues and M. Rowe, *The Coloring, Bronzing and Patination of Metals*, Watson-Guptill Publications, New York, 1991.
38. R. Barzel, 'Faking dynamics of ropes and springs', *IEEE Computer Graphics and Applications*, **17**(3), 31–39 (1997).

Authors' biographies:

Stephane Gobron is a doctoral student in the Department of Computer Science at Iwate University, Morioka, Japan. He received a BS in computer graphics and software engineering from the Florida Institute of Technology, FL, U.S.A. and a DEA

Copyright © 1999 John Wiley & Sons, Ltd.

(Master) in computer graphics and CS languages theories from the University of Nancy I, Loria, France, in 1994 and 1995 respectively. He worked as a computer graphics engineer in 1996. His research interests include computer graphics and cellular modelling.

Norishige Chiba is currently a Professor in the Department of Computer Science at Iwate University. His research interests include computer graphics, algorithm theory and science on form. He received a BE in electrical engineering from Iwate University and an ME and DE in information engineering from Tohoku University in 1975, 1981 and 1984, respectively. He worked at Nippon Business Consultant Co., Ltd from 1975 to 1978. He was a Research Associate in the Department of Communication Engineering at Tohoku University from 1984 to 1986, an Associate Professor of Computer Science at Sendai National College of Technology from 1986 to 1987 and an Associate Professor of Computer Science at Iwate University from 1987 to 1991. He is a member of IEICE Japan, IPS of Japan, IEEE and ACM.

Copyright © 1999 John Wiley & Sons, Ltd.