Vol. 38 No. 6 Transactions of Information Processing Society of Japan June 1997

Regular Paper

A Comprehensive System for Texture Synthesis Developed from L-Systems

MIN-LU DAI† and KAZUMASA OZAWA†

This paper presents a comprehensive system for texture synthesis, developed from the L-systems, which is composed of three independent procedures *generator*, *gene* and *operator*. The proposed system separates description of primitives from distribution of them. This makes it possible to direct a dominant local orientation of primitives and delimit the region containing primitives. In addition, we extend PPBFFG (the planar parallel binary fission/fusion grammars) into multi-fission/fusion and stochastic systems. Basic structures of some textures have been analyzed. We also show several synthesized textures using the proposed system.

1. Introduction

Texture is summarized as surface characteristics of objects. It is widely used in engineering, natural science, art, animation and design. Especially, besides geometry, texture plays a critical role in simulating real objects in computer graphics. A picture, which is textureless, tends to appear barren, unrealistic and boring.

Textures are classified into three types; strongly ordered, weakly ordered and disordered textures [1]. In strongly ordered textures, spatial interactions between primitives are so considerablely regular that the textures can be described by structural approaches. Weakly ordered textures have weak spatial interactions between primitives, and can be adequately characterized by frequencies of primitive types appearing in a small range. Textures in which spatial distribution of primitives appears at random are regarded as disordered textures.

There exist two main types of texture descriptions; statistical and syntactic methods [2),3)]. The statistical method induces the different primitive-sized properties of a texture which is useful for simulating textures with small sized primitives. The syntatic methods (including hybrid methods given by combination of statistical and syntatic ones) describe textures using a grammar which represents a set of productions to fit primitives with a larger variety of properties corresponding to tonal properties.

We now have a lot of techiques for synthesizing natural textures [4)~13)]. However, most of them could only be used to construct a particular kind of textures and few experiments have been done to attempt other different textures.

Texture is composed of various kinds of primitives. Basically, a texture synthesizing system should be able to describe many kinds of primitives. However, it appears existing methods seldom meet such demand. In addtion, those methods focus mainly on parameters involved in the synthesizing process: Once the parameters are determined, there would be no chance to change their assignment at later stages.

By relying on our TSL-systems (Texture Synthesis L-systems) [14], we have taken several small steps toward comprehensively solving this problem. TSL-systems belong to parallel rewriting systems, of which productions are applied in parallel and simultanneously replace all letters in a given word. Since texture consists of mutually related primitives, a parallel rewriting system will be an effective model for generating the spatial distribution of primitives. A TSL-system is specified by the primitive and region L-systems. The primitive L-systems defines primitives described by functions, procedures or coordinate transformations, etc. The region L-system starts from simply geometric models and then generates more and more complexly geometric models for delimiting the region of developing primitives.

TSL-systems are, however, not sufficient for synthesizing texture. In texture syntheses, sometimes, dominant orientation of primitives should be considered. TSL-systems still do not show how to direct a dominant local orientation of primitives. To remedy this shortcoming, a newly modified approach CSTS (a Comprehensive System for Texture Synthesis), different from our previous paper [14], is proposed. CSTS

† Department of Engineering Informatics, Osaka Electro-Communication University

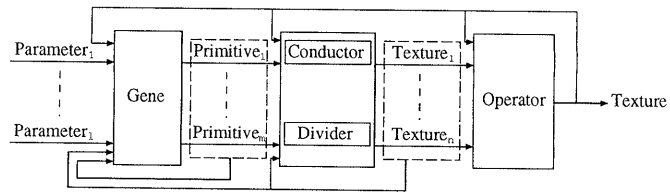


Fig. 1 The basic structure of CTSS.

employs a *conductor* to guide the direction of primitive generation. Furthermore, CSTS separates description of primitives from building process of spatial relationship between primitives so that a complicated model can be defined by several sub-models. In order to obtain a more general appliction to texture synthesis, our new approach allows modifing some parameters to control a final texture. Our approach is developed from L-systems [15] which is an effective method to simulate natural phenomena. The grammar of CSTS is context-sensitive, parametric and stochastic.

2. Texture Synthesis Using CSTS

The central concept of CSTS is that a complicated model can be represented by some simpler sub-models. In CSTS, a texture can be described by three sub-procedures. Namely,

- *Generator*, through which spatial relationships between primitives can be created,
- *Gene*, which defines a primitive,
- *Operator*, which combines some textures into a new one.

Figure 1 shows the overall system architecture. Three sub-procedures are independent with each other. Simple textures can directly be generated by them. On the other hand, these sub-procedures are also embeded (nested). A more complicated texture can be constructed by combination of the sub-procedures.

2.1 Generator

Generator is subdivided into *conductor* and *divider*. *Conductor* is used to direct a dominant local orientation of primitives. *Divider* acts to delimit regions for development of primitives.

Conductor is an extension of L-systems. It uses context-sensitive, stochastic and parametric grammars to guide the developing direction of a primitive. We use special symbols, given later, to represent primitives and productions to control movement of primitives. Conductor is different from L-systems in the following two points:

- Parameters in a productions are not only a set of real numbers, but also vectors, functions and procedures.
- Rewriting models not only are limited to "teaching the turtle" [15], but also are linked with some information on orientation in sequentially rewriting models.

Symbolically, conductor is defined as an ordered quintuplet $G = \langle \mathrm{V}, \omega, \mathrm{P}, \mathrm{F}, \mu \rangle$.
Where

- V is an alphabet, V^* is a set of all words, V^+ is a set of all nonempty words.
- $\omega \in V^+$ is an initiator,
- $\mathrm{P} \subset V^* \times V^-$ is a set of productions,
- F is a set of formal parameters,
- μ is a probability distribution of productions.

According to the above definition, a production is given as follows:

$$\mathrm{Prob}: \mathrm{LC} \ll \mathrm{Pre}\ (\mathrm{Para}_p) \gg \mathrm{RC} : \mathrm{Cond} \longrightarrow \mathrm{Succ}\ (\mathrm{Para}_s).$$

Where Prob is a probability of applying the production, LC and RC are the left and right context, Cond is condition, Pre (Para$_p$) and Succ (Para$_s$) are the predecessor and successor with their parameters.

CSTS-1 illustrates an application of conductor.

CSTS-1

ω: A(X_0,Y_0).
P0: A(X,Y): Y $\geq$ TERMINAL $\longrightarrow$STOP.
P1: 0.5: A(X,Y) $\longrightarrow$B(P(W=3)).
P2: 0.5: A(X,Y) $\longrightarrow$B(P(W=4)).
P3: 0.33: B(P(W)) $\longrightarrow$ D(P(W),X+Δ,Y), A(X+Δ,Y+1).
P4: 0.33: B(P(W)) $\longrightarrow$ D(P(W),X,Y), A(X,Y+1).
P5: 0.33: B(P(W)) $\longrightarrow$ D(P(W),X-Δ,Y), A(X-Δ,Y+1).

Where, P(W), which is a vector, is width of a vein, D(P(W),X,Y) denotes the vein started from position(X,Y) with width=P(W). Produc-

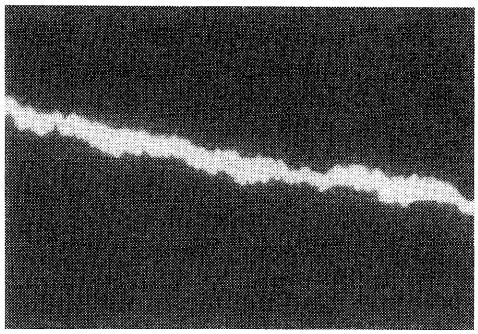


Fig. 2 A line generated by CTSS-1.

tions in CSTS-1 are classified into two groups. P1, P2 and P(W) can be regarded as description of primitives (*gene*) discussed later. Various primitives are given by selecting P1 and P2. P3, P4 and P5 are conductor productions which control movement of primitives. They are stochastic productions with the same probability. We start with an apex A. P(W) is a primitive generated by production A. Then we draw the first primitive by selecting P2, P3 and P4 at random. Note, in L-systems, strings produced at different rewriting levels must be reserved, i.e. the higher the rewriting level is, the longer the string is. In order to overcome this problem, CSTS-1 reserves only the current rewriting string. **Figure 2** presents a result produced by CSTS-1.

In some circumstances, development of primitives is limited to a region. Such a region is often irregular, polygonal and non-overlapped with other regions. This is a problem of dividing regions. By far, the most popular and widely used approaches for this problem are random processes [16),17)]. Such methods are so complicated that only a person who has scientific and engineering knowledge can use them. In addition, regions generated by these methods are restricted to rectangles which are not enough to simulate realistic textures. *Divider* in CSTS offers a method of dividing regions.

Divider is an extension of a map generating system [18)~20)] by which a complex geometric structure can be generated from a simple model. According to Tutte's definition [15)], descriptions and notations referred to in this paper are first summarized as follows:

Map is more strict but better implementable definition than *planar graph*. Difference of their definitions are that a graph is represented by vertices and edges, but a map, a connected

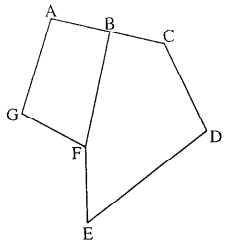


Fig. 3 A map. A, B, C, D, E, F and G are vertices; AB, BC, CD, DE, EF, FG and GA are edges; ABFG and BCDEF are two neighbors and BF is their common deges.

structure without islands or peninsulas, is composed of vertices, edges and regions. Each vertex is connected with one or more edges. Each edge has one or two vertices at its terminations. A region is bounded by finite sequences of edges. All edges and vertices lie on boundaries of regions. A region can have common edges with other regions. **Figure 3** illustrates definition of map.

Rewriting one map from another can be *region-controlled* or *edge-controlled* [21)]. In region-controlled map-rewriting systems, regions are considered to be the basic units and therefore they should be labeled and undergo *fission* and *fusion* described later. This way corresponds to node labeling in graph grammars. In edge-controlled map-rewriting systems, labelled edges are considered as the main control factors. This corresponds to edge labeling in graph generation.

Region manipulation, corresponding to generation of planar maps, has two types; i.e. splitting (*fission*) and merging (*fusion*). Splitting is to divide a region into serveral regions and merging is to combine serveral regions into a region. The first fission/fusion grammar PRDL-systems (the parallel region dividing L-systems) were proposed by Carlyle, et al. [22)]. This is based on planar parallel binary fission/fusion grammars (PPBFFG). PPBFFG are "binary" systems, i.e. each region can be just split into two regions and the only two regions can be merged into a new region in a rewriting step.

Since PPBFFG is a "binary" and "deterministic" system, textures with striking artificial regularity have been generated. To overcome this problem, we extend PPCFFG to multi-fission/fusion and stochastic systems.

Multi-fission/fusion is that a region can be split into several daughter regions, some of which can be merged into a new region in a

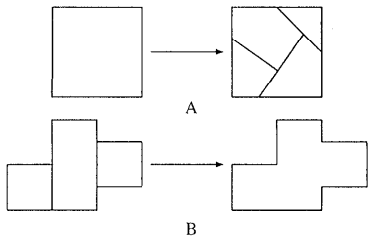


Fig. 4 Multi-fission and fusion productions. (A) A region is split into four sub-regions. (B) Three regions are merged into a new region.

rewriting step. Such an example is given by **Fig. 4**. Stochastic means that selection of productions is done stochastically. For simplicity, we do not give the formal definition of *divider*, but we present an example which explains its application. Wallboard is a simple texture which can be synthesized by using *divider* is given by the following CSTS-2.

CSTS-2

ω: M
P0: M : SubMapBuf=EMPTY & NowLevel≥ FinalLevel ⟶ Remove(TmpSubMap, SubMapBuf), M. END.
P1: M : SubMapBuf≠EMPTY & NowLevel≥ FinalLevel Pop(SubMapBuf, Map), Deformed(Map), M.
P2: M : SubMapBuf=EMPTY & NowLevel ⟨ FinalLevel⟶ NowLevel+1, Remove (TmpSubMap, SubMapBuf), M.
P3: M :⟶ Pop(SubMapBuf, Map), DivMap(Map), M.
P4: 0.20: DivMap(Map)⟶ SubMap(LA1), SubMap(LA2), SubMap (LA3).
P5: 0.20: DivMap(Map)⟶ SubMap(LB1), SubMap(LB2), SubMap (LB3).
P6: 0.20: DivMap(Map)⟶ SubMap(LC1), SubMap(LC2), SubMap (LC3).
P7: 0.20: DivMap(Map)⟶ SbMap(LD1), SubMap(LD2), SubMap (LD3).
P8: 0.20: DivMap(Map) ≪ Pop(SubMapBuf, NeighbourMap) : NeighbourMap ∪ Map ⟶ SubMap(LN1), SubMap(LN2), SubMap(LN3), SubMap(LN4).
P9: SubMap(Map)⟶ Push(TmpRegion,SubMap).
P10:0.33: Deformed(Map)⟶ Affin(Map,Triangle), Push(TmpRegion,Triangle).
P11:0.33: Deformed(Map)⟶ Affin(Map,Rectangle), Push(TmpRegion,Rectangle).
P12:0.33: Deformed(Map)⟶ Affin(Map,Triangle), Push(TmpRegion,Triangle).

Productions in CSTS-2 are classified into three levels: Productions P0, P1, P2 and P3 in the first level control the map rewriting procedure. Productions P4, P5, P6, P7, P8 and P9 in the second level generate a random distribuction of regions with different sizes. Productions P10, P11 and P12 in the third level generate various polygons, which are difficult to be represented in common mathematical and procedurable ways. Map rewriting in CSTS-2 is divided into *total map rewriting* and *sub-map rewriting*. Total map rewriting P2 is rewriting a map with the original size. Sub-map rewriting P3 is rewriting a map obtained from the original map. Both are different levels in map rewriting.

Rewriting a map starts from production ω, where an initial map is a rectangle. NowLevel and FinalLevel are a counter of a rewriting level and a final rewriting level, respectively. When NowLevel is larger than FinalLevel, the map rewriting is terminated by P0. And then, P1 starts P10, P11 and P12, which generate various polygons, in the third level.

SubMapBuf is a sub-map buffer, all sub-maps rewritten by sub-map rewriting, and TmpSubMap is also a buffer for saving sub-maps generated by merging and splitting. When SubMapBuf is empty and NowLevel is less than or equal to FinalLevel, total map rewriting of the current level is terminated. P2 increases NowLevel, removes sub-maps in TmpSubMap into SubMapBuf and starts a new total map rewriting. Consequently, P3 pops a sub-map from SubMapBuf and starts a sub-map rewriting.

Productions in the second level are selected at random. They have the same production probability 0.20. P4, P5, P6 and P7 are fission productions which split a map into three submaps. P8 is a fission/ fusion production. Introduction of fission/fusion, similar to the feedback function in the control theory, can create more natural distribution for dividing regions.

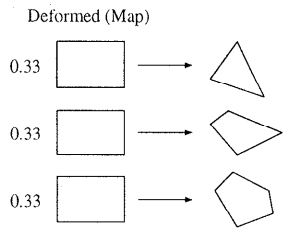


Fig. 5 Productions Deform.

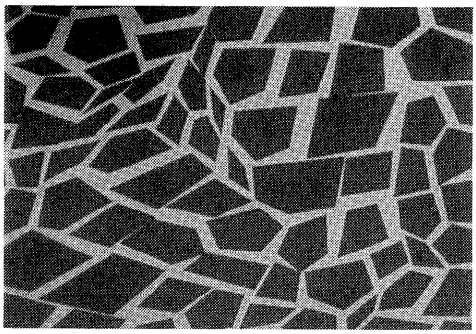
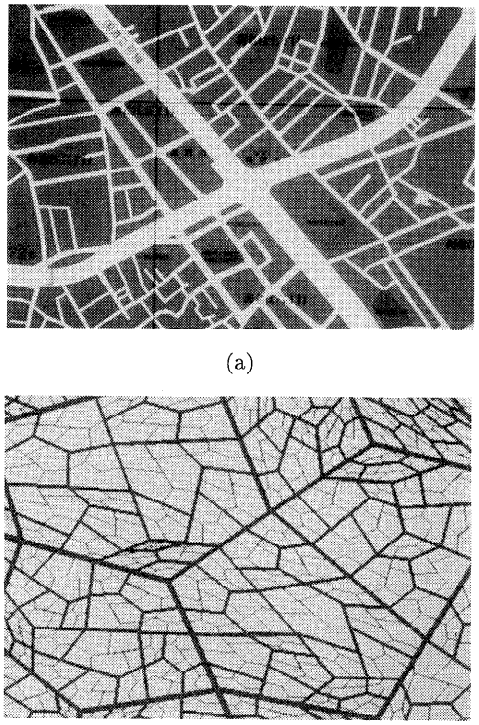


Fig. 6 The synthesized wallboard texture with CTSS-2.

The symbol $\ll$ in P8 represents DivMap and Pop to be contex-sensitive. NeighbourBoundary ∪ Boundary represents that two sub-maps are neighbored. P7 first merges two sub-maps into a new one and then splitting this new one into six sub-maps aga. Sub-maps generated by the second level are pushed into TmpSubMap. P10, P11 and P12 in the third level are also stochastic productions. Their production probility is 0.33. Productions in the third level are shown by **Fig. 5**. **Figure 6** is a model generated by CSTS-2.

In CSTS-2, polygons are generated by two kinds of productions DivMap and DeformMap. If we combine DivMap and DeformMap into one kind of productions, *Voronoi polygons*[23),24)] which are also useful for texture segmentation[25)] could be constructed. As seen in the other examples described later, Voronoi polygons provide a way of texture synthesis.

Figure 7 demonstrates execution of DivDeformMap, where map rewriting is carried out at four levels and the line thickness is halved at each level to emphasize the higher levels. We make use of parameters to produce an effect of fusible function (merging or fusion). A parameter is a position of a vertex. Since position of the vertex is individual for each sub-map rewriting, natural distribution of sub-maps can also be obtained.



(a)

(b)

Fig. 7 A structure of Voronoi polygons generated by CTSS, where FinalLevel=4 and the line thickness is halved at each level.

2.2 Describing Primitives

Primitives of a natural texture are generally irregular; structural errors, distortions, deformations or even structural variations are often seen. This means no strict production can be used to simulate a real texture. For syntactic description of textures, non-deterministic or stochastic grammars should be incorporated into the description grammars. Furthermore, introduction of special parameters can reduce the number of productions and computational complexity. *Gene*, which describes properties of a primitive, is given based on context-sensitive, stochastic and parametric L-systems. According to the concept of L-systems, we can use a group of productions to produce different kinds of primitives. Gene can represent color, size, shape, direction or their combination each of which is regarded as a primitive. This provides a powerful tool for texture synthesis.

Here, an example of gene is given by the following CSTS-3, omitting the formal definition (for a more detailed discussion, see Ref. 15)):

CSTS-3

#define P(X_s,Y_s) the start point
#define P(X_e,Y_e) the terminal point
#define P(X_c,Y_c) the current point
#define P($X_c = X_s, Y_c = Y_s$) the initial point.
ω: A.
P0: A : $(X_s = X_e) \mid \left|\frac{Y_e - Y_s}{X_e - X_s}\right| \geq 1 \longrightarrow$ B.

P1: A : $(X_s = X_e) \mid \left|\frac{Y_e - Y_s}{X_e - X_s}\right| < 1 \longrightarrow$ C.

P2: B : $Y_c > Y_e \longrightarrow$ END.
P3: B : 0.33: $\longrightarrow$ Draw(P($X_c + \Delta X$, Y_c, Width,Color), $Y_c = Y_c + 1$.
P4: B : 0.33: $\longrightarrow$ Draw(P(X_c, Y_c, Width,Color), $Y_c = Y_c + 1$.
P5: B : 0.33: $\longrightarrow$ Draw(P(X_c - ΔX, Y_c, Width,Color), $Y_c = Y_c + 1$.
P6: C : $X_c > X_e \longrightarrow$ END.
P7: C : 0.33: $\longrightarrow$ Draw(P(X_c, $Y_c + \Delta Y$, Width,Color), $X_c = X_c + 1$.
P8: C : 0.33: $\longrightarrow$ Draw(P(X_c, Y_c, Width,Color), $X_c = X_c + 1$.
P9: C : 0.33: $\longrightarrow$ Draw(P(X_c, Y_c - ΔY, Width,Color), $X_c = X_c + 1$.

CSTS-3 is to draw a line: its length is $\mid Y_s - Y_e \mid$ or $\mid X_s - X_e \mid$ and its direction is represented by Eq. (1) or Eq. (2). Productions B and C represent two kinds of Draw (primitive), respectively. When

$$\left|\frac{Y_e - Y_s}{X_e - X_s}\right| \geq 1, \tag{1}$$

the shift of a line is ΔX. When

$$\left|\frac{Y_e - Y_s}{X_e - X_s}\right| < 1, \tag{2}$$

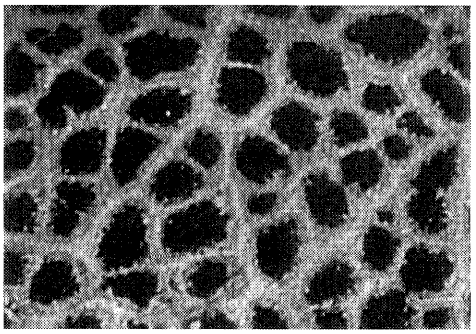
the shift of a line is ΔY.

Productions in B or C have the same production probability. If we change this parameter, direction and shape of the line could be changed.

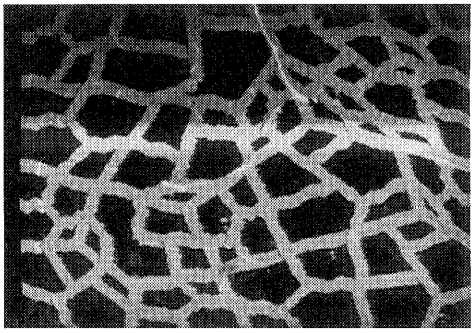
We can combine CSTS-2 and CSTS-3 into a complete model of texture synthesis. An interesting category of textures that resemble leaf veins, roadmaps, blood vessel and nervous systems and so on, can be established. **Figure 8** shows a synthesized texture. We can also adjust the value of Width and Color so that more colorful results may be generated.

According to CSTS-2, we use gene to represent development of color in a sub-map, i.e. color is considered as a primitive. And then the final level of map rewriting is increased to 5. We obtain a synthesized marble texture shown in **Fig. 9**.

In CSTS, hierachical textures can be generated by embeded structures. Some specific pattern (a texture) obtained from lower primitives may be considered as a new primitive at



(a)



(b)

Fig. 8 The wing of an insect synthesized by combination of CTSS-2 and CTSS-3, where FinalLevel=3 and the line thichness is equal at each level.

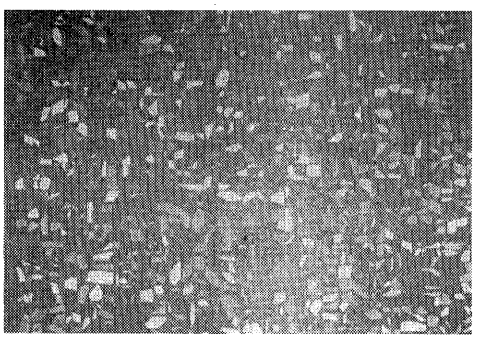


Fig. 9 A synthesized marble texture where TerminalNum=6 and colors of polygons are of three types.

a higher describing level. And then the new pattern (a texture) constructed by a new primitive can be also regarded as a primitive at a succeeding higher describing level.

In addition, other ways of texture synthesis can be combined with CSTS by means of gene. **Figure 10** is a synthesized texture with two levels given by the above mentioned method. A basic primitive (gene) at the first level is a 20×8 matrix F constructed as follows:

$$f_{ij} = Const_{ij} + d. \quad (3)$$

Where $i \in [1, 20], j \in [1, 8]$, $Const_{ij}$ is a constant and d is a random value reducing artificial effect. Starting from Eq. (3), we construct the first level texture by conductor, which is a curved strip, according to the concept of CSTS-1. Following the first level, the curved strip is considered as a primitive (gene) at the second level and then a complete texture can be established by CSTS.

Figure 11 is another example of hierachical textures. Here, a basic primitives is a line. The length, width, direction and color of the line is variable. First, we construct 500 lines in different positions and obtain the first level texture. This texture is regarded as a primitive in the second level. Consequently, we rewrite the new primitive (*total map rewriting*, FinalLevel=4). Finally, another marble texture is generated.

Figure 12 (a) is an embeded model of multi-primitive levels. Figure 12 (b) shows the CSTS's structure. GENEH1 defines a curve in which curvature may be changed. GencH2 describes the second level primitive which is composed by four curves generated from GENEH1. CONDUCTORH1 guides the movement of GENEH2 in the horizontal direction which forms the third level primitive. CONDUCTORH2 constructs the spatial distribution of CONDUCTORH1. GENEV1, GENEV2, CONDUCTORV1 and CONDUCTORV2 establish the model in the vertical direction similar to that in the horizontal direction.

2.3 Combining Different Textures

Operator provides an additional operation of CSTS which includes logical operation, arithmetic operation, matrix transformation, filtering, opaque overlapping and their combination. Note, when generating simple textures, this function will not be needed.

Let A_k and B_k be synthesized textures, respectively. Let C_k be a newly combined texture. Here, we define *operator* as follows:

$$C_k = A_k \bigcirc B_k, \quad (4)$$

where $\bigcirc$ is the symbolical notation of *operator*.

Figure 13 is a synthesized textile texture which is composed by two different textures. We construct a basic textile texture and its worn marked texture with CSTS, respectively. And then two different textures are combined into a new one.

Operator can also be given by a polymomial expression. Symbolically, we have

$$C_k = \sum_{k=1}^{N} A_k \bigcirc B_k, \quad (5)$$

where N is the number of textures. The complete listing of CSTS for operator is not to be included in this paper, but two examples provide a good illustration.

Figure 14 (a) shows a texture of a leather handbag synthesized by CSTS. First, we decompose the texture into three independent textures (Figs. 14 (b), (c), and (d)), which can be established in such ways as mentioned, respectively. Then, we compose these three independent textures into a new one with the opaque overlapping operation. The idea given here provides a simple solution for constructing complicated texture models. **Figure 15** gives another example composed by the logical operation OR.

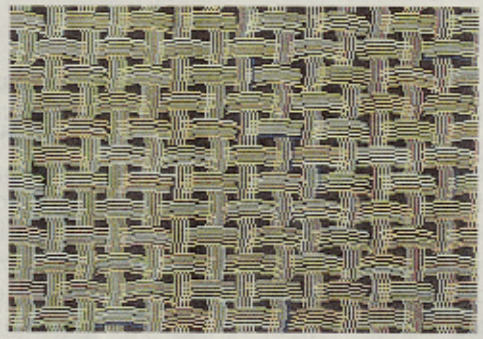
3. Conclusion

CSTS provides useful models for the simulation of natural textures. In CSTS, primitives of a texture and of spatial relationship between primitives are handled by different procedures, respectively. The final texture is controlled by the parameters at every rewriting level. Extented L-systems (*generator* and *gene*) provide simple and intuitive description of textures. CSTS is a parametric and stochastic system. Larger change of models can be achieved by selecting productions at random. On the other hand, smaller change can be implemented by adjusting parameters of productions.

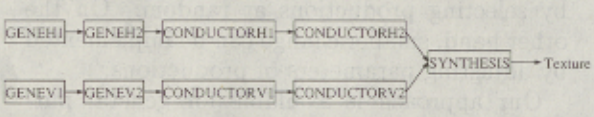
Our approach is available for general purposes: It can be used for various textures; hierachical and embedded structures can be used for more complex textures. As discussed, this has been verified by some examples. Note, there can be different primitives at different rewriting levels. Furthermore, different primitives can be concurrently developed at the same rewriting level. These can be achieved by selecting different productions at a rewriting level and at different levels, respectively. Such flexibility could

Fig. 10 A model of wax printing cloth constructed by the hierachial textures.

Fig. 11 Another model of marble. This is also an example for hierachial textures.



(a)

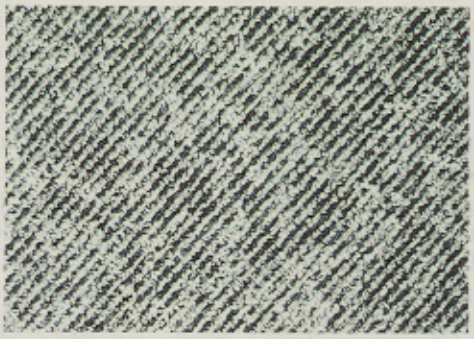


(b)

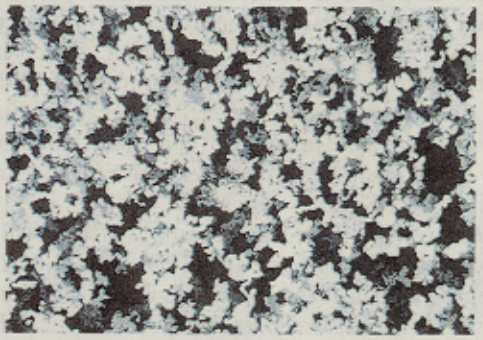
Fig. 12 A texture contructed by an embeded model. (a) A synthesized texture. (b) Explanation of the embeded model.

successfully be used to build models of many natural textures.

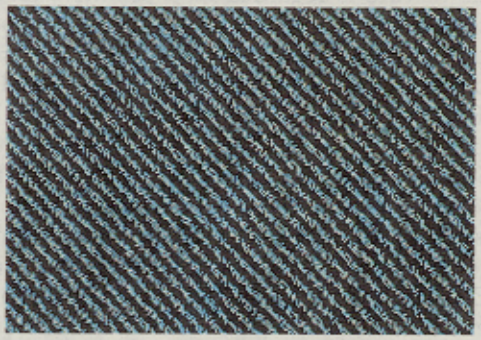
Though our method presented in this paper works well, there is room for futher improvement. Main points that we have to pay more



(a)



(b)



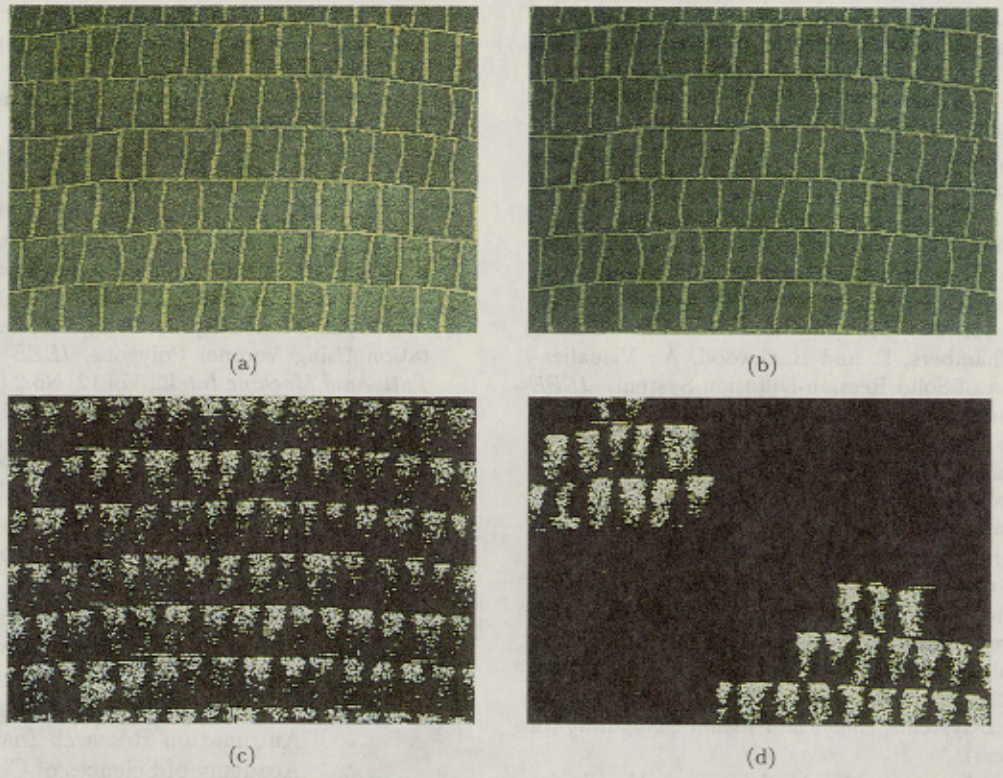
(c)

Fig. 13 A textile texture composed by two different textures. (a) A synthesized texture. (b) A worn sign. (c) A new cloth.

attentions include:

1. The proposed method is not suitable for representation of disordered textures. As mentioned above, development of a primitive is limited in a fixed region so that common calculation of primitives in several regions is impossible. Although context-sensitive grammar can partially avoid this effect, this still remains a principal deficiency.

2. Combining a structural analysis. Real textures being complicated and various, it is impossible to synthesize all types of textures



(a) (b) (c) (d)

Fig. 14 The synthesis leather handbag surface composed of three subtextures. (a) A synthesized texture. (b) Creation of spacial relationship; horizontal and vertical planar dividing. (c) The worn signs wear for each region. (d) The local worn effect for a group of regions.

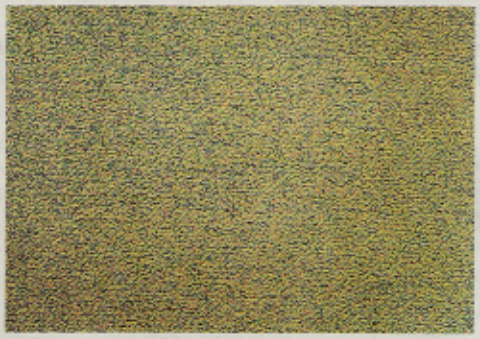


Fig. 15 A woolen cloth texture generated by *operator*.

by a general method. Scanned real world images stand on a principal source of real textures. An interesting way of texture synthesis may take place using primitives extracted from real textures. Tomita et al have given such an approach as is to be called the *analysis-by-synthesis* method [26]. It is a possible future task to combine our method with such a structural analysis.

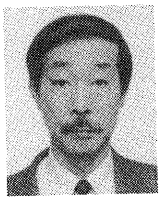
References

1) Reed, T.: A Review of Recent Texture Segmentation and Feature Extraction Techniques, *CVGIP: Image Understanding*, Vol.57, No.3, pp.359–372 (1993).
2) Lipkin, B.C. and Rosenfeld, A. (Eds.): *Picture Processing and Psychopictorics*, Academin, New York (1970).
3) Haralick, R.M.: Statistical and Structural Approaches to Texture, *Proc. IEEE*, Vol.67, No.5, pp.786–804 (1979).
4) Reed, T.: A Review of Recent Texture Segmentation and Feature Extraction Techniques, *CVGIP: Image Understanding*, Vol.57, No.3, pp.359–372 (1993).
5) Mandelbrot, B.: *Fractal: Form, Chance, and Dimension*, W.H. Freeman, San Fransisco (1977).
6) Peachy, D.R.: Solid Texturing of Complex Surfaces, *Computer Graphics (SIGGRAPH'85)*, Vol.19, No.3, pp.279–286 (1985).
7) Perln, K.: An Image Synthesizer, *Computer Graphics* (SIGGRAPH'85), Vol.19, No.3,

pp.287–296 (1985).
8) van Wijk, J.J.: Spot Noise: Texture Synthesis for Data Visualization, *Computer Graphics* (SIGGRAPH'91), Vol.25, No.4, pp.299–307 (1991).
9) Witkin, A. and Kass, M.: Rection-Diffusion Textures, *Computer Graphics* (SIGGRAPH'91), Vol.25, No.4, pp.299–307 (1991).
10) Turk, G.: Generating Textures on Arbitrary Surfaces Using Reaction-Diffusion, *Computer Graphics* (SIGGRAPH'91), Vol.25, No.4, pp.289–298 (1991).
11) Chambers, P. and Rockwood, A.: Visualization of Solid Rection-Diffusion Systems, *IEEE Computer Graphics and Applications*, Vol.15, No.5, pp.7–11 (1995).
12) Sakas, G.: *Fractal Geometry and Computer Graphics: Modeling Turbulent Gaseous Motion Using Time-Varying Fractal*, pp.173–194, Springer-Verlag, New York (1992).
13) Gröuer, E., Rau, R.T. and Straßer, W.: Modeling and Visualization of Knitwear, *IEEE Trans. Visualization and Computer Graphics*, Vol.1, No.4, pp.302–310 (1995).
14) Dai, M.L. and Ozawa, K.: Texture Synthesis by L-systems, *Image and Vision Computing* (in press).
15) Lindenmayer, A.: Mathematical Models for Cellular Interaction in Dcvclopment, Parts I and II, *Journal of Theoretical Biology*, Vol.18, pp.280–315 (1968).
16) Derin, H. and Elliot, H.: Model and Segmentation of Noise and Textured Imanges Using Gibbs Random Fields, *IEEE Trans. Patt. Anal Machine Intell.*, Vol.PAMI-9, No.1, pp.39–55 (1987).
17) Onural, L.: Generating Connected Textured Fractal Patterns Using Markov Random Fields, *IEEE Trans. Patt. Anal Machine Intell.*, Vol.13, No.8 (1991).
18) Tuttc, W.T.: *Graph Theory*, Addison-Wesley, Reading, MA (1984).
19) Claus, V., Ehrig, H. and Rozenberg, G. (Eds.): Graph-Grammars and their Application to Computer Science and Biology, *Lecture Notes in Computer Sciences*, pp.367–378, Spring-Verlag (1979).
20) Ehrig, H., Nagl, M. and Rozenberg, G. (Eds.): Graph Grammars and their Application to Computer Sciences, *Lecture Notes in Computer Science*, pp.288–296, Spring-Verlag (1983).
21) Nakamura, A., Lindenmayer, A. and Aizawa, K.: Some Systems for Map Generation, *The Book of L*, Rozenberg, G. and Salomaa, A. (Eds.), pp.323–332, Springer-Verlag (1986).
22) Carlyle, J.W., Grelibach, S.A. and Paz, A.: Planar Map Generation by Parallel Binary Fission/Fussion Grammars, *The Book of L*, Rozenberg, G. and Salomaa, A. (Eds.), pp.29–44, Spring-Verlag, Berlin (1986).
23) Voronoi, G.: Nouvelles applications des paramétres continus á la théorie des forms quadratiques. Deuxiëme mémoire: Recheches sur les paralléllоédres primitifs, *J. Reine Angew. Math.*, Vol.134, pp.198 287 (1908).
24) Shamos, M.I. and Hoey, D.: Closest-Point Problems, *Proc. 16th Annu. Symp. Fundations of Computer Sciencc*, pp.131–162 (1975).
25) Tüceryan, M. and Kjaw, A.: Texture Segmentation Using Voronoi Polygons, *IEEE Trans. Patt. Anal Machine Intell.*, Vol.12, No.2 (1990).
26) Tomita, F., Shirai, Y. and Tsuji, S.: Description of Textures by a Structural Analysis, *IEEE Trans. Patt. Anal Machine Intell.*, Vol.PAMI4, No.2 (1982).

(Received July 18, 1996)
(Accepted March 7, 1997)

Min-lu Dai received his B.S. degree from Talien University of Technology of China in 1982 and M.S. degree from Shenyang Automation Research Institute, Academy of Sciences of China in 1988. Since 1991, he has been working for Shibasoku Co. Ltd, Japan. During 1993–1996, he was a student of the Graduate School of Engineering, Osaka, Electro-Communication University, Neyagawa, Osaka, Japan. He is presently a Visiting Scientist at the university. His research interests include image processing and computer graphics.



Kazumasa Ozawa received the B.E., M.E. and Ph.D. degrees in electrical engineering from Osaka University, Japan, in 1966, 1969 and 1972, respectively. Since 1972, he has been with Osaka Electro-Communication University, Neyagawa, Osaka, Japan. He is presently a Professor of the Faculty of Information Science and Technology. His teaching and research interests include pattern recognition, computer graphics and computer applications in archaeology. He published four books on information theory and on computer applications in archaeology. He is a member of IPSJ, IEICE, IEEE, Pattern Recognition Society and British Machine Vision Association.