Title:
Cellular Automata Based Representation of 3D Patterns

Authors:
Jae K. Shin, jkshin@yu.ac.kr, Yeungnam University

Keywords:
Cellular Automata, 3D printing, F-code, Primitives, Pattern representation

DOI: 10.14733/cadconfP.2017.405-409

Introduction:

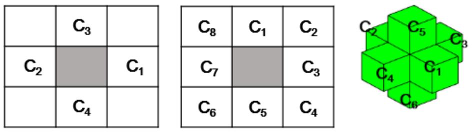
Typical CAD systems are based on a set of well-defined graphic primitives such as, for example, points and lines for 2D and cube, cylinder and sphere for 3D applications. It will be clear that the more diverse patterns we can get with the more diverse primitive sets we can use. Motivated from an increasing emphasis on an aesthetic design of commercial products and with an increased power of 3D printing that can fabricate complex shapes immediately from the CAD models, the present study experiments on a new way of representing 2D/3D patterns using Cellular Automata (CA). CA has long been used as a means for studying complex systems [7]. Wolfram expected that CA can explain the simple rules the complex natural patterns emerge. For example, it was found that complex natural patterns like snow crystal and skin pattern of sea shells can be explained in terms of simple rules of CA [3]. However, obtaining complex patterns using CA in general is not an easy task [1]. In the present paper, the author uses a special kind of CA [4,5] and suggests a systematic way of applying it as a general tool for representing complex patterns.

Pattern Representation Using CA:

In general, Cellular Automat (CA) is defined on a tessellation system composed of uniform cells. Commonly used cell types include square, honeycomb and triangles. For the present paper, square type is assumed without loss of generality. A two-dimensional canvas can be defined as an X by Y square cells. A cell state can be expressed in terms of a digits, 0, 1, 2, … . The number representing a cell state will be called the *color code* for the cell. Color code 0 will be used for an empty cell. Different set of color codes defined for a canvas will determine different patterns of the system.

In CA, a cell can change its state depending on the cell states of its neighborhood. The two most common types of neighborhoods are the *von Neumann neighborhood* and the *Moore neighborhood* (See Fig.1). An updating rule or function determines the state of a cell in the next time step as a function of the neighborhoods' cell states of the current time step. In 2D, the Moore neighborhood is composed of B=8 cells. Thus, a transition rule can be written as $c=f(r(\mathbf{C}))$. Here c denotes the value of the transition rule and $\mathbf{C}=(C_1,C_2,C_3,,,C_B)$ represents the cell states for the neighborhood. It is generally assumed that the transition rule applies for all the cells with non-trivial neighborhood. For the present study, the rule applies only for the empty cells with non-trivial neighborhood, which will be called surface cells. Once a cell is determined its state with nonzero color code, it does not change its states.

Typically, the rule for updating the state of cells is the same for each cell and does not change over time, and is applied to the whole grid or canvas simultaneously. In such a synchronous updating, all the surface cells in the system are subject to change. For the present study, I use a kind of asynchronous rule firing scheme conveniently named as one-rule firing CA [4,5]. In the one-rule firing scheme, only the subset of surface cells whose rule value is the minimum among all the surface cells at the current time step are subject to change.



(a) (b) (c)

Fig. 1: (a) 2D von Neumann, (c) 2D Moore, (c) 3D von Neumann neighborhood

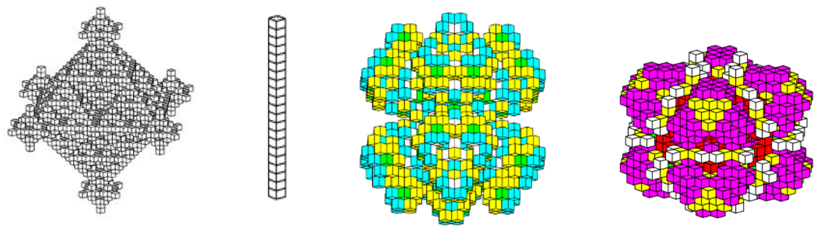
	r	1	2	3	4	5	6	7	8
Rule Set 1	c=f(r)	1	1	1	1	0	0	0	0
Rule Set 2	c=f(r)	1	0	0	0	0	0	0	1

Tab. 1: Example rule sets for a totalistic rule scheme.

The transition function *c*=f(r(**C**)) can be defined in many ways. For the present study, I use a weighted-sum rule defined as the following: $r=\sum_{i=1}^{B} w_i C_i$. When all of the weighting factors are 1, namely w_i=1 for i=1 to *B*, we get the totalistic rule set [1]. For a two-state CA (C_i=0 or 1) with the Moore neighborhood, we can have 8 different rule numbers *r* ranging from 1 to 8. For example, two set of rules are shown in Table 1. Rule set 1 consists of 8 individual rules of f(1)=f(2)=f(3)=f(4)=1 and f(5)=f(6)=f(7)=f(8)=0. When we use multiple colors or use different weights, the size of a rule set will become very large.

Diverse patterns can be generated using the one-rule firing CA. To grow a pattern, we basically need two kind of data: an initial pattern and a rule set. In many cases, we start from a single cell filled with a color code of 1. From the initial pattern, we apply the rules in a discrete time steps. If we have to define or store a pattern, we need to remember the rule set or the rule table. When the size of a rule-set is very large, we may have trouble in remembering the rule table. For complex rule scheme, the size of a rule set easily exceeds 10^4 [4].

In one-rule firing CA, however, the patterns can be easily stored in a series of codes (F-codes) that were actually fired. An example F-codes and corresponding 3D patterns are shown in Fig. 2. Starting from a seed pattern, you can use the F-code to grow-up the pattern, without a separate rule table. Each of the patterns shown in Fig.2 are grown up to the time step T=20 or 30 starting from an initial single seed with color code=1. The cases Fig. 2(a) and 2(b) shows that the same F-code, F=1, can result in different patterns, depending on the rule schemes used. The number of patterns that can be obtained from F-codes are almost an infinite. Simply listing a series of number will give an F-pattern.



(a) F=1 (b)F=1 (c) F=3502101 (d) F=601054044211

Fig. 2: Example F-patterns grown from an initial single seed of color code 1. (a) Totalistic, T=20, nE=1801, (b)W=[2,2,2,2,2,1], T=20, nE=21, (C) Totalistic, T=30, nE=1481, (d) Totalistic, T=30, nE=1467.

In addition to the initial patterns and rule table, we need more information for a pattern to be completely defined. Table 2 lists the information required to define a pattern completely. The type of tessellation, neighborhood topology and rule scheme are among the necessary information. In this study, we discuss the weighting factor W of the totalistic rule set in detail.

Elements	In this study	Examples/comments
•Tessellation types	•diamond	•Diamond, Hexagonal, Triangular, etc.
•Neighborhood topology	• Von Neumann	•2D, 3D

	• Moore	
•Rule Scheme	•Weighted-sum	•Symmetry of rules •Type of functions
•Initial seeds	•Single Seed	•Locations and colors
•F-codes		•Minimum rule firing
•Canvas size	•Infinite	
•Boundary conditions	•Open	•Aperiodic, periodic
•Repeating sequence		•Number of Mixing, Life(T)

Tab. 2: Definition of patterns.

Primitives and Mixing Sequence:
Although there are an enormous number of F-patterns, it is still a question if a designer can use the patterns systematically for designing something as he/she can do with a typical CAD system today. To generally answer the question is beyond the scope of the present study. As a first step for such an answer, I will show how simple patterns such as lines, rectangles and cubes can be represented in terms of the F-patterns. For the 3D patterns in this study, von Neumann neighborhood is used. The six neighborhood cells are numbered in the order of (+x,-x, +y, -y, +z, -z)(See Fig. 1(c)).

Straight lines can be generated using specially designed weights for the weighted sum rule. Starting from an initial seed of color code 1 and applying the pattern F1 in Table 3 repeatedly, we can get a pattern stretching in the positive x-direction. For the pattern F1, rule weight of W= [U, 1, U, U, U, U] and F-code of F=1 are used. The condition U>1 means that any value greater than 1 is sufficient for the value U. Figure 3 explains how the patterns grow with the application of the one-rule firing system. Similarly, we can define lines in the other main axes directions using different weights. Generating lines in an arbitrary direction may not be possible with a single F-code. It can be a limitation combined with the specific type of tessellation systems.

Generating a rectangle of size n x m (assuming cell size is unit square) with a single F-code will not be easy. However, it can be easily obtained mixing several F-patterns. If we want to generate a rectangle of size n by m, it can be generated using the patterns F1 and F3 in Tab. 3, for example. Staring from an initial seed, we can grow the pattern using F1 for n-1 time steps. This will give a line of length n in the +x direction. From time step n, we continue to grow the pattern using F2. Applying F2 for one-time step, we can get a new line of length n just above the previous line in the +y direction. Continuing F2 for m-1 time steps, we can get a rectangle of size n by m.

In the present method of using CA, the process of generating mixing simple patterns will be one of the key operations for building complex patterns. Simple patterns take the form of a single F-code. It can be compared to the primitives in the conventional CAD system. The number of typical 2D or 3D primitives is relatively small in commercial CAD packages. When the F-codes are used as primitives, the number of primitives in the present system will be almost an infinite. In a typical CAD system, there are many operations to build up complex objects out of the simple primitives. Boolean operations are among one of them. In the present system, the key operation is the time-wise mixing. From an initial seed, the patterns grow with the application of the F-codes differently designed for each of the time steps.

A pattern in the present study is generally defined with its primitives (F-code) and mixing sequence. The mixing sequence can be expressed with a list of numbers. For example, the mixing sequence for the rectangle of size n by m , as explained above, will be represented as S=[1 1 1 1…1 (m-1 times) 2 2 ..2 (n-1 times)]. In the sequence, the number 1 and 2 represents the primitive number for the case at hand. Or it can be simplified to $S=1^{n-1}2^{m-1}$. Similarly we can generate a prism of size n by m by p using primitive patterns F1, F3 and F4 in Table 3, with a mixing sequence of $S=1^{n-1}2^{m-1}3^{p-1}$.

Figure 5 shows an example of complex object obtained mixing two primitives. In Fig. 5, a complete list of the geometric data for the object is given. The list means to be self-complete. No other data is necessary for obtaining the object. The data exactly defines the location and color of every single voxel exactly.

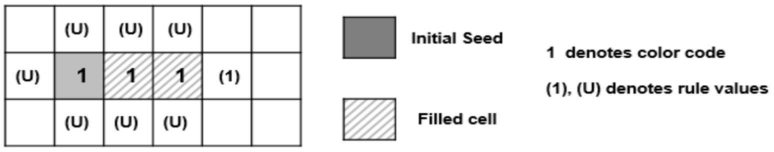


Fig. 3: Growing of a line pattern F1. Left figue shows the pattern after time step 2 and the rule values of the surface cells for the next time step. The cell with minimum rule numbe (r=1) will be fired next.

Pattern		weight	F-code	
F1	+x line	U,1,U,U,U,U	1	(U>1)
F2	-x lines	1,U,U,U,U,U	1	(U>1)
F3	+y line	U,U,U,1,U,U	1	(U>1)
F4	+z line	U,U,U,U,U,1	1	(U>1)
F5	(Fig. 5)	1,1,1,1,1,1	2520124	
F6	(Fig. 5)	1,1,1,1,1,1	5305	

Tab. 3: F-patterns used as a primitive in the present study.

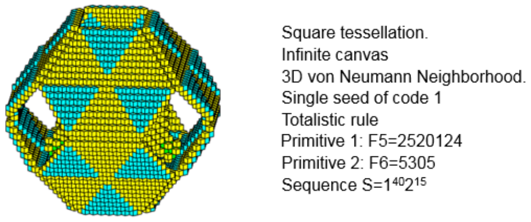


Fig. 5: A mixed pattern and its self-complete description.

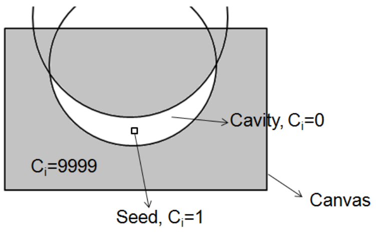


Fig. 6: Cavity generation.

<u>A Simple Application:</u>
The patterns in the previous section are grown in an infinite canvas, without boundary. We can introduce boundaries to generate interesting patterns. For example, we first generate a cavity of target shape. And next growing F-patterns inside the cavity, we get patterns that has the shape of the cavity but having different textures depending on the filling F-patterns. For an example, we molded a dish using a cavity formed between two spheres whose centers do not coincide. The method for generating a cavity for a 2D case is explained in Fig. 6. To generate a cavity in the form of a crescendo shown in Fig. 6, we use two circles. By filling the cells of canvas outside of the cavity with a big color code, say 9999, and inside with color code of 0, we get the desired cavity. Starting to grow a pattern at an initial seed of color code 1, we get patterns filled inside the cavity. Figure 7 shows the patterns obtained using different F-codes. Only one F-pattern is used for each of the dishes shown in Fig. 7. Of course, we could mix F-patterns if we wish. Observe that the dishes Fig. 7(c) and (d) differs in their F-codes at 6-th digit. In many cases, similar F-codes result in similar shapes, but not always.

It is already known that the patterns generated from CA can be fabricated using 3D printers [2, 6]. In Figure 8, we show some of the 3D printed patterns obtained in this study.

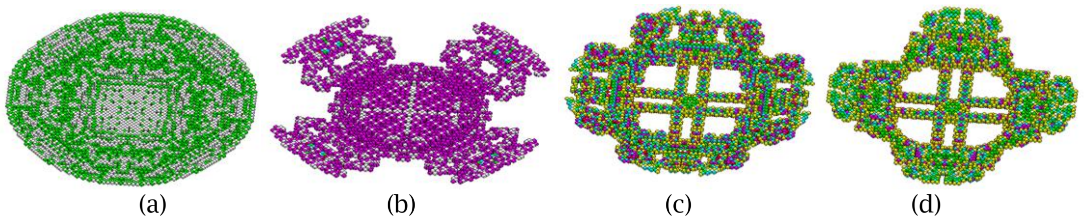


Fig. 7: Dishes generated by cavity filling, (a) F=212,T=93, nE*=10224 , (b) F=4110231620, T=166, nE=5204 (c) F=2350**6**142210001412, T=222, nE=5948, (d) F=2350**0**142210,T=160,nE=5248 (*nE means number of cells).

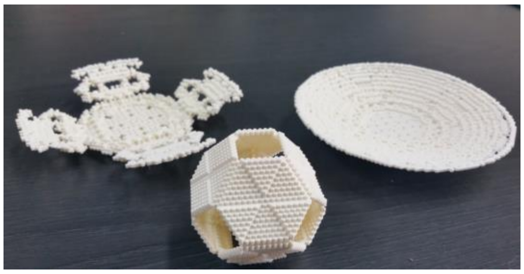


Fig. 8: 3D printed patterns.

Discussion and conclusion:

In the present study, we first applied F-codes for 3D F-patterns. Especially, we experimented on a possibility of using F-patterns as a primitive for a 3D CAD system. Wolfram coined a phrase 'a new kind of science' as a method of using CA as a tool for studying physics. The CA based description of geometries could be similarly viewed as 'a new kind of geometry'. It defines the geometry such as lines, squares and cubes quite in a different way from those in the classical geometry. It is expected that the pattern formation using the present system will have both advantages and disadvantages at the same time. It will be hard to get the F-code representation for a pattern already existing in designers mind. With almost an infinite number of primitives, the F-codes, it is expected the present system can get patterns beyond his/her imagination. It means that the system could be very promising as a creative or an aesthetic pattern design tool.

References

[1] De Garis, H.: Artificial embryology and cellular differentiation, Creative evolutionary systems, Edited P. Bentley and D. Corne.Chap.12, Morgan Kaufmann, 2002.

[2] Kanada, Y.: Self-orgnized 3D-printing patterns simulated by Cellular Automata, Kanada, Y., in *Y.* Suzuki and M. Hagiya, ed., Recent Advances in Natural Computing, 2016.

[3] Levy, S.: Artificial life, Vintage books, 1992.

[4] Shin, J.: Identifying patterns from one-rule firing cellular automata, Artificial Life 17(1), 2011, 21-32. https://doi: 10.1162/artl_a_00015

[5] Shin, J.: Application of cellular automata for a generative art system, Leonardo 49(5), 2016, 431-435. https://doi:10.1162/LEON_a_00964.

[6] Southwell, R.: The universe of 3D Cellular automata, https://www.youtube.com/watch?v=OxASD5xvgKI.

[7] Wolfram, S.: A new kind of science, Wolfram Media, 2002.