Cellular Automata

Vit Krajcik
krajcik@decef.elf.stuba.sk

Faculty of Electrical Engineering and Information Technologies
Slovak University of Technology
Bratislava, Slovak Republic

Abstract

This paper deals with one approach designing complex biomorphical shapes by means of the evolutional processes, which are represented by the cellular automata with different types of cells. The paper describes a method for the definition the grammar and simulation of cellular array (transformation functions for reproduction, aging and dying), set up the initial state of cellular automata, graphic interpretation of cell states, and some experimental results.

Keywords: Cellular automata, evolutiaonary process, grammar

1. Introduction

Cellular automata have a broad spectrum of applications - from computer games, to the simulation physical, biological, and social precesses.

For example, there are applications in archeology, where cellular automata is used for the re-construction of migration and evolution of the old cultures. In this case the simulation helps to predict the location of archaeological sites [3].

Theoretical base of cellular automatta statet John von Neumann, but the practical experiments were able when the computers became able to compute and visualize simulations. Cellular automata require large memory.

This paper is part of the undergraduate course at the Department of Computer Science and Technology. The main goal of this project is to create a tool for automatic data base generation in the modelling and rendering complicated shapes.

2. Cellular automata

Cellular automata in general, is the set of cells placed in a regular ortogonal n-dimensional grid. Single cell can reproduce, depending on its genetic code (rewriting rules), its state, and the state of neighbouring cells.

General definition of the automata is following.

It is ordered base **(A,S,U,p,v)**, where **A** is set of input symbols, **S** is set of states ,**U** is set of output symbols, **p** and **v** are transformations.

```
p : S x A -> S
v : S -> U
```

Transformation **p** is transition function and

Transformation **v** is output function.

Output function **v** assigns output symbol for each state. Transition function assigns change state to next state.Transition functions are defined in a discrete time intervals t:

```
s(t+1) = p( s(t), a(t) )
u(t) = v( s(t) )
```

where **s(t+1), s(t)** is from set of **S**
a(t) is from set of **A**
u(t) is from set of **U**

This paper deals with two-dimension orhogonal cellular arry.

2.1 Neighbouring cells

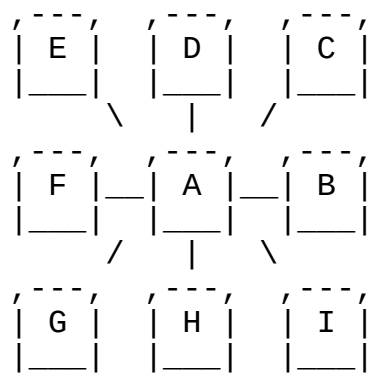
Two cells are neighbouring, if their absolute distance is less than 2

Example:

In matrix array neighbours of cell with coordinates A(X,Y) are:

B(X+1, Y)
C(X+1,Y+1)
D(X ,Y+1)
E(X-1,Y+1)
F(X-1, Y)
G(X-1,Y-1)
H(X ,Y-1)
I(X+1,Y-1)

d = sqrt(sqr(X - (x+1)) + sqr(y - (y+1)))
d = sqrt(1 + 1)
d = 1,4142136 ...



Each cell have eight neighbours.

2.2 States of the cells

State of cell is represented by the integer number.State 0 is reserved for dead cell, and other states are aviable for use. Generally we can define arbitrary number of states for different cell behaviour, but if computer allow less colors, then different cells will have the same color on the computer monitor. Graphical representation of the cellular automata described in this paper is limited by 256 colors of the standard VESA video modes.

2.3 Graphical interpretation of the states

Each state of the cell is represented by one color. The color is defined by three components: RED, GREEN, BLUE The value of colors for the particular state is calculated from the color values of the first and last state of the cell by the linear interpolation.

Until now, five types of graphic cell interpretations were implemented:

1. point
2. filled square
3. filled square with variable size
4. filled circle
5. filled circle with variable size.

Size is of the object is proportaionl to the screen resolution and the size of cellular array.

Syntax:

Cell : InitState : (R1,G1,B1)->(R2,G2,B2) ;

where **Cell** is name of the cell,
InitState is integer number from interval [0,MaxState]
R1,G1,B1 color of the state 1
R2,G2,B2 color of the state MaxState

Example

A : 1 : (1,1,1)->(1,1,255) :

2.4 Specification of transition functions

Transition functions can be expressed by the rewriting rules in the form:

Cell CellList -> MutateCell NewCellList : Cell operator number : Probability;

where **Cell** is name of the actual cell,
CellListb is a list of neighbouring cells (cells which influence the transition)
MutateCell is a name of the cell, which will replace active (checked) cell,
NewCellList is a name list of the cells, which will be created at the neighbourghood of the checked cell. Their position will be chosen randomly.
Cell operator number is an extended condition for application of the transition function, where
Cell is an arbitrary name of the cell, which is from the neighbourhood of the checked cell
operator is one of following arithmetic operators: **< , > , =**
number is an integer number from the interval [0,MaxState]
Probability is an integer from the interval [0,100], and imagine probability of apply transition function in per cent.

Example:

A B C D -> A A A : A < 5 : 15 ;

Means that at the neighbourhood of cell A - under the condition that the neighbours are cells named B,C and D and at the same time the state of the arbitrary cell A is less than 5 - two new cells A will be created, with the probability of 0.15;

Next transition rule for defining the process of ageing cells is :

Cell : +-number : Cell operator number : InitState ;

where **Cell** is a name of the cell,
+-number is an integer increment which will be added to the state of the cell after each iteration (new generation).
Cell operator number is extended condition where
Cell is a name of the checked cell,
operator is one of the following arithmetic operators: **< , > , =**
number is integer number from interval [0,MaxState+1]

This extended condition is usefull for the checking of color cycling. InitState is an integer number from interval [0,MaxState]. This is an inicial state of the cell.

Example:

A : +1 : A < 20 : 1 ;

Means increasing state of the cell A by one, if its state is less than 20. At the begining the state is 1.

2.5 Setting the initial state of the cell array

Initialization can be done in three ways:

1. Enumeration - list of cells with their X,Y coordinates and state numbers

Cell (X , Y) = InitState ;

Description see below.

Example:

A(10 , 24) = 10 ;
B(22 , 10) = 1 ;

2. Process - in this case rewriting parallel graphs in form of L-systems (see [1])

Syntax:

Cell (X , Y) = InitState -> EndCell = EndCellInitState : string ;

where **Cell** is name of the cell,
X, Y are coordinates,
InitState is integer number from interval [0,MaxState], initial state of each cell.
EndCell is the cell at the end of the line representing branch of the L-system.
EndCellInitState is an integer number from the interval [0,MaxState]
EndCell initial state of the cell at the end of the brach.

Example:

LEN = 5 ;
ALFA = 60 ;
A (50 , 10) = 1 -> A = 1 : ff(+f)(-f)f(+ff(+f)(-f)f)(-ff(+f)(-f)f)ff(+f)(-f)f .

means that the start (root) of the L-system has coordinates (50,10). Line consist from A cells. Each cell has an initial state equal to 1. Length of each brach is 5 units.

3. Initial state as a result of simulation.

This method allows to get the initial state of the cellular array from the result of simulation of another cellular automata.

3. Implementation and experimental results

Program is implemented in BORLAND C language. The process of simulation consists from three steps:

1. Computing a new generation of the cellular array

1.1 Application of rules for reproduction and mutation - scan array and try apply rule for each living cell (with the state > 0)

1.2 Application of rules for ageing - scan array and try apply rule for ageing

2. Storing cell array - changes are saved on a disk

3. Display - display cellular arry of the new generation on the monitor

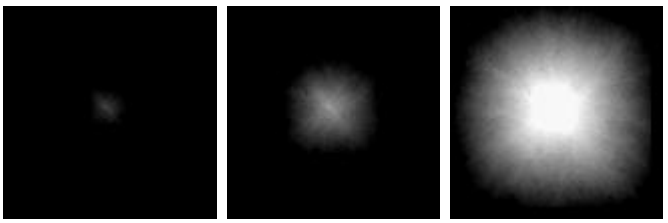
Example 1.:

Simple grammar for reproduction cells.

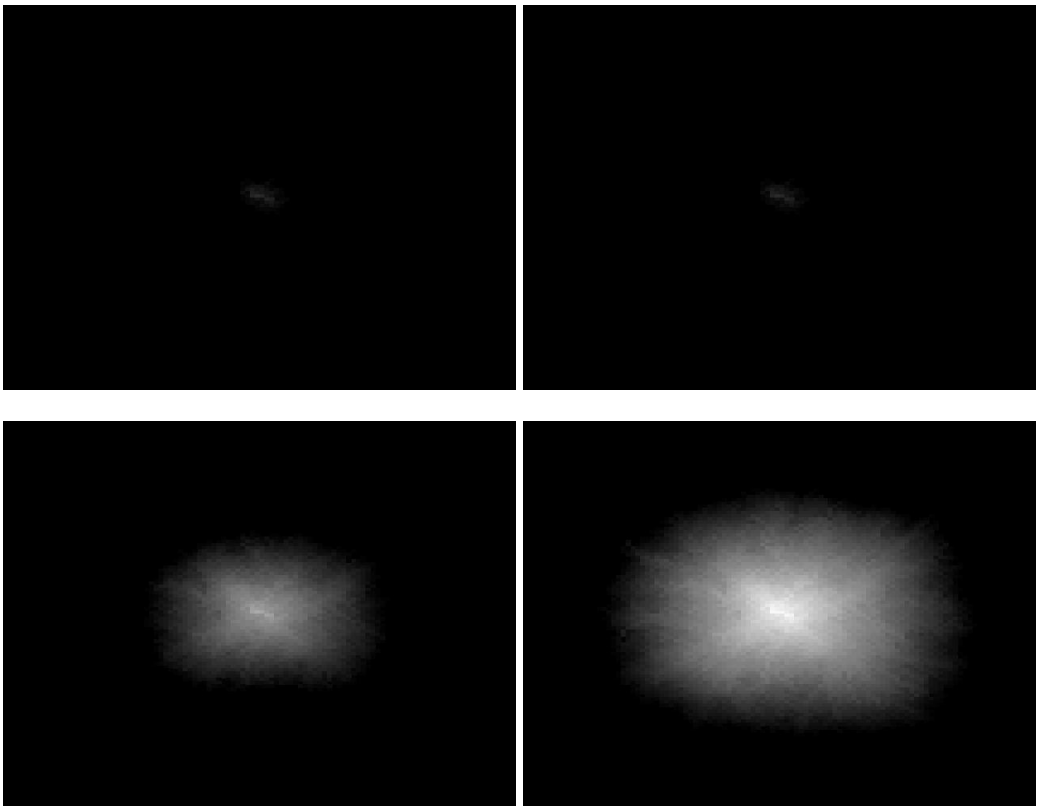
Grammar:

```
DIM
N=64;
X=100;
Y=100;
f=5;
a=60.
VAR
A : 1 : (1,1,1)->(63,63,63) .
AGE
A : +1 : A < 63 .
STARTUP
A ( 50 50 ) = 1 .
BEGIN
A -> A A : A < 63 : 50 .
END
```

Sample images after the simulation (after 10,40,100 iterations)



Pixel cells 1-3



Bar cells 1-4

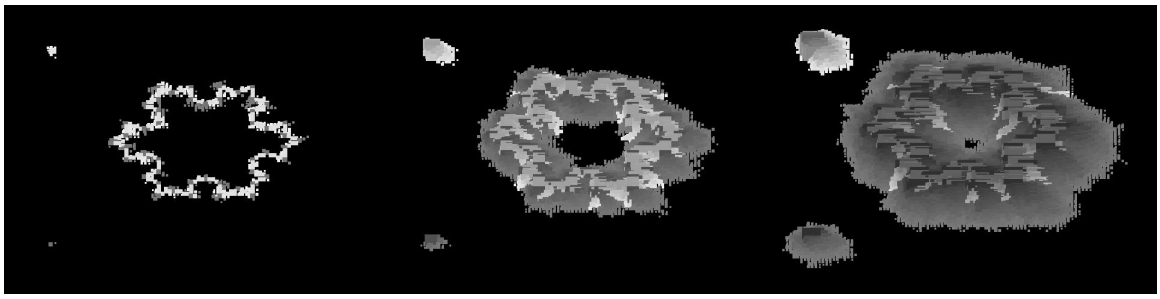
Example 2.

Grammar:

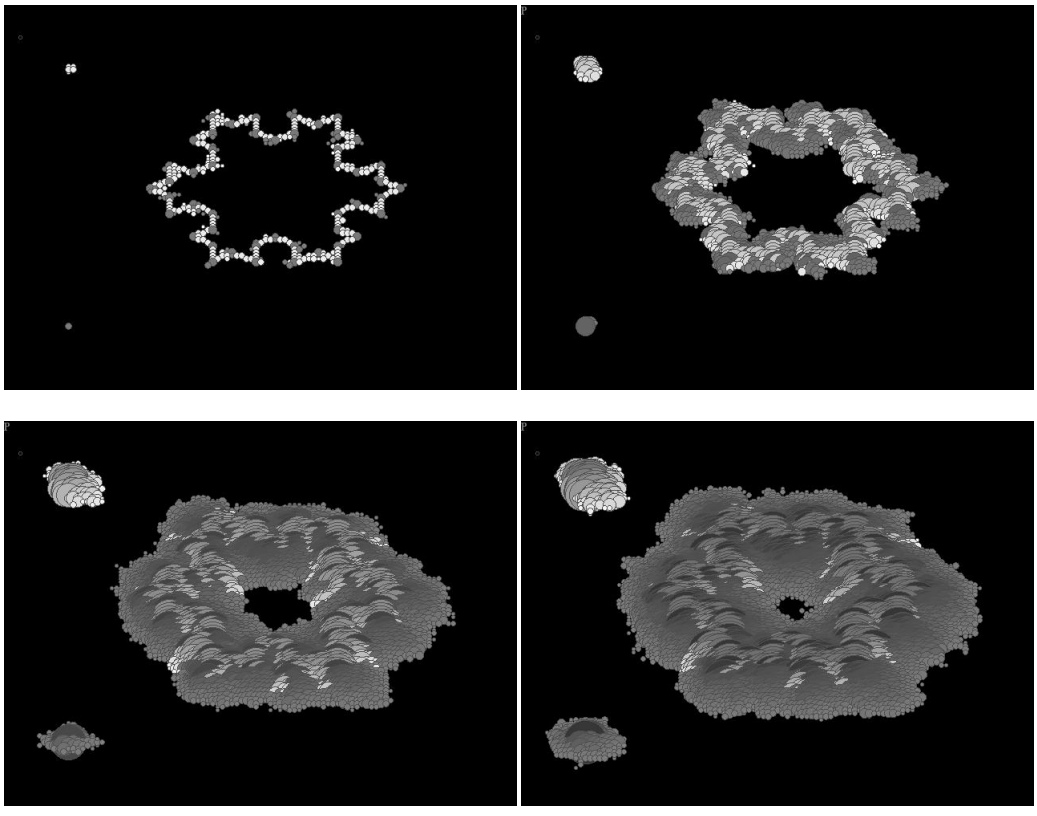
```
DIM
N=64;
X=100;
Y=100;
f=5;
a=60.
VAR
A : 1 : (33,33,33)->(10,10,10) ;
B : 1 : (63,1,63)->(1,1,1) ;
C : 1 : (63,1,63)->(1,1,1) ;
D : 1 : (63,63,63)->(20,20,20) .
AGE
A : +1 : A < 63 ;
D : +1 : D < 63 .
STARTUP
A ( 10 10 ) = 1 ;
```

D (10 90) = 1 ;
D (40 30) = 1 -> A = 2 : f+f--f+f+f+f--f+f--f+f--f+f+f+f--f+f--f+f--f+f+f+f--f+f--f+f--
f+f+f+f--f+f--f+f--f+f+f+f--f+f--f+f--f+f+f+f--f+f .
BEGIN
A -> A A : A < 60 : 20 ;
D -> D D : D < 60 : 10 .
END

Sample images after the simulation (after 10,40,100 iterations)



Variable sized bar cells 1-3

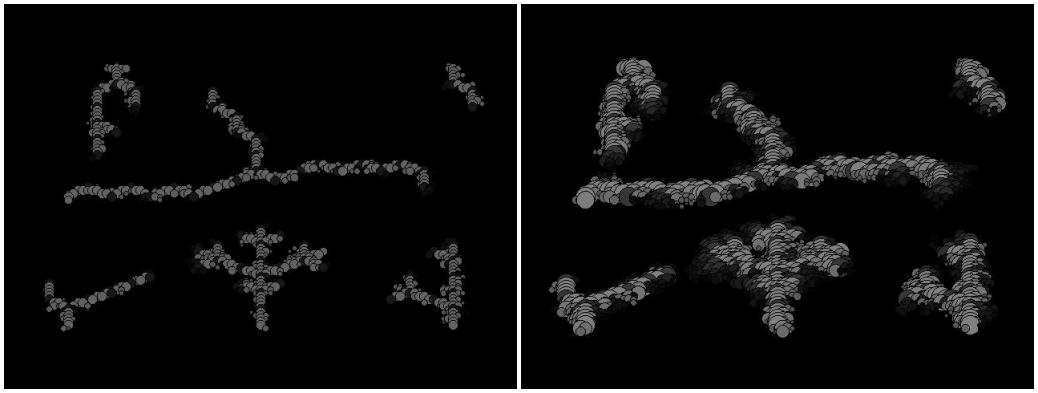


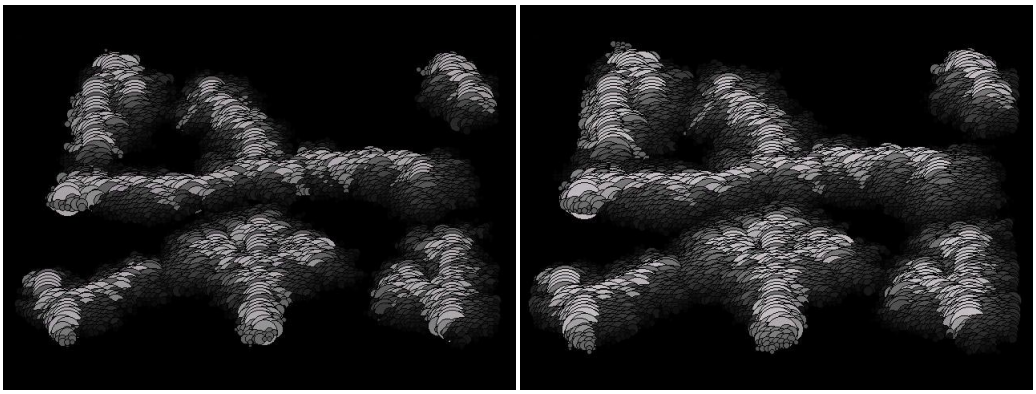
Variable sized ellipses 1-4

Example 3.

```
DIM
N=64;
X=100;
Y=100;
f=5;
a=60.
VAR
A : 1 : (1,1,1)->(43,43,43) ;
D : 1 : (20,20,20)->(63,60,63) ;
B : 1 : (1,1,1)->(1,1,1) ;
C : 1 : (1,1,1)->(1,1,1) .
AGE
A : +1 : A < 63 ;
D : +1 : D < 63 .
STARTUP
D ( 90 10 ) = 1 -> A = 1 : f(fff(+f)f)+ff(-f)+f ;
D ( 90 90 ) = 1 -> A = 1 : ---f+f-f ;
D ( 10 50 ) = 1 -> A = 1 : -f-f+f-f+f-f+fff(-f+ff-f+f-f+f-f-f)+ff+f-f+f-f ;
D ( 20 90 ) = 1 -> A = 1 : +++f(+f-f)-f+ff(ff)+f ;
D ( 10 10 ) = 1 -> A = 1 : f(+f-f)-ffff ;
D ( 50 10 ) = 1 -> A = 1 : ff(+f)(-f)f(+ff(+f)(-f)f)(-ff(+f)(-f)f)ff(+f)(-f)f .
BEGIN
A -> A A : A < 60 : 20 ;
D -> D D : D < 60 : 10 .
END
```

Sample images after the simulation (after 10,40,100 iterations)





Variable sized ellipses 1-4

4. Conclusion and future plans

This system allows the flexible way of defining behaviour rules and simulation of cellular automata. Results of the simulation are presented in the graphical form after each iteration step. The system allows different graphical cell states interpretations (by means of 2D shapes).

Future work will concentrate on the implementation of cellular automata in three-dimensional grid, and interpretation of 2D and 3D grid by means of three dimensional objects.

One of the alternatives is using metaballs and metaellipses, which will allow to model biomorphical shapes.

At the end I would like to thank associate professor Martin Sperka for some comments during the work on this project.

5. References

[1] Fristacky,N.,a kol.: Logicke systemy, Alfa, Bratislava, 1986.

[2] Zara,J.,a kol.: Pocitacova grafika - principy a algoritmy, Grada, Praha, 1992.

[3] Rise v pocitaci, 100+1, jun 1998, pp.6-7.