Image Effects with Cellular Automata

Siniša Petrić

***Abstract*—This paper presents some techniques for creating various artistic effects on digital photography using the concept of cellular automata. All examples in this paper are created by "Image Infector" program, which is a plugin for Pixopedia 24 image editor and painter (www.sigmapi-design.com).**

***Index Terms*—cellular, automata, image, effects, artistic, growth, color, infection**

I. Introduction

USING cellular automaton is not a new concept in digital imaging. It is used for creating various patterns, for simulating ink dispersion[3] and for image filtering[2]. Cellular automaton growth is controlled by predefined rule, usually controlled by program, or by some CF grammar (L-system)[1]. The rule describes how the cell will interact with its neighborhood. Once the automaton is started it will work on its own according to the rule specified.

System described in this paper is little bit different than ordinary approach, in the way that rules are not completely specified by language, but are partially specified by means of pixels values inside 8-bit grayscale image which is used as a container for automaton rules. Another big difference is that ordinary cellular automaton changes cell state regarding its neighborhood, while this system propagates cell to its neighborhood (free slot), requiring less iterations to obtain desired results and having some similarities with particle system. With this method, we are not constrained to a single rule, rather, we can mix various automaton rules on the single image. The container size must be the same as original image (photography) size.

II. Image Infector

IMAGE INFECTOR is Pixopedia 24 plugin that uses 8-bit grayscale image (container mask) to hold automata rules. Image Infector controls automaton by means of 3 groups of parameter: seeding rule, automaton rule and infection color. The first group controls the way of spreading initial cells population over our mask.

The second group is used to specify the direction of cells growth. Here we have options: we can specify random direction, preferred direction, strict direction and mixture of all of them. Depending on these parameters grayscale image is seeded with initial cells with the values that correspond to desired local rule by means of seed value

As the value of pixel in 8-bit grayscale image is one byte, theoretically we can specify 255 different rules, but as byte value 0x00 means "free slot", while value 0xff represents "fixed cell", it leaves as with 253 different rules that we can specified. All cells with values greater then 0x00 and less then 0xff are assumed to be active. Current Image Infector version (1.1) described in this paper uses only 19 different rules.
The third group is used to control output pixel color used to infect original image (photography).

III. Seeding

THIS section describes seeding mechanism, id est. how initial cells population is distributed over our grayscale mask. Seeding section is not completely separated from rules, as seed elements also hold crucial rule information. However, under seeding topic, we will cover only various types of seeds distribution. Currently, there are 7 seeding styles (*SeedStyle* variable):

1) Random seeds: the cells are distributed randomly across the mask

2) Grid seeds: equidistant cells seeding. All cells are separated from each other by fixed distance.

3) Contour seeding: if some 8-bit grayscale contour of original image is created (option inside Pixopedia 24), contour pixels, which usually have value 0xff are modified with selected automaton rule values. Contour mask is used as a container mask.

4) Seed with dithered original image (Bayer): original image is converted to grayscale and Bayer dithering algorithm is applied with level=2 and noise=0.

5) Seed with dithered original image (Jarvis-Judice-Ninke): same as above, with different dithering algorithm.

6) Double seed high intensity areas: original image is converted to BW image using threshold value of 128 (0x80) and seeding is performed in two passed. On the first pass, mask is seeded uniformly and on the second pass, only areas with values greater or equal to 128 are seeded again.

7) Double seed low intensity areas: same as above, but with second pass seeding values below 128.

Another important variable in this group is *SeedWindow* variable, used to calculate the number of seeds (*SeedNum*). *SeedNum* is calculated using formula:

$$SeedNum = \frac{ImageWidth * ImageHieght}{SeedWindow^2}$$

The last parameter in this section is Boolean variable Fixed-Contour, which, if set to true, set values on image contour to 0xff in order to preserve image edges.

IV. Automaton rules

RULES are the crucial part of Image Infector, as growth of seeded cells is controlled by these rules. Regardless of rule chosen, infection mechanism, generally works by scanning container mask and looking for a non-fixed cell.

When such cell is found and focused, cell propagation will take place in focused cell's neighborhood which can be visualized by 3x3 matrix:

$$N = \begin{bmatrix} NW & N & NE \\ W & X & E \\ SW & S & SE \end{bmatrix}$$

X is focused cell value, while surrounding cells are denoted according to their position from focused cell (northwest, north, northeast, etc...). Inside Image Infector, this wind rose matrix is implemented as a one row matrix with omitted middle element *X*, so that position of next infected cell can be easily calculated:

$$W = [NW, N, NE, W, E, SW, S, SE]$$

Wind rose matrix (array) W, depending on the rule selected, can correspond to single byte value, or to similar one row matrix (array). For instance, if preferred direction rule is chosen, wind rose array corresponds to:

$$R = [0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08]$$

single row byte matrix.

Choosing direction for next cell depends on the selected rule. If chosen new cell is occupied, automaton will try to find alternative free cells and will actually perform branching. Maximum number of branches is determined by variable *BranchNum* (number of branches when cell is occupied). If no free slot is found, automaton proceeds to next scanning step. Currently, there are 19 possible automaton rules, grouped in 3 sections:

1) Preferred direction: X takes its value from array $[0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08]$. Focused cell value will be used to calculate direction of propagation. It means that cell with value 0x01 will try to occupy nearest slot in the northeast direction, if value is 0x02 it will try to infect nearest free slot in the north, etc... If neighbor pixel in this direction is occupied, first free slot will be occupied and the new direction will be set (according to free slot position).

   Example: Let's assume that $BranchNum > 0$. While scanning container mask we have found the pixel with value X=0x02 and surrounding neighborhood is:

$$N = \begin{bmatrix} 0x01 & 0xff & 0x11 \\ 0x03 & X & 0x00 \\ 0x00 & 0x00 & 0x00 \end{bmatrix}$$

   This neighbor matrix corresponds to array:

$$W = [0x01, 0xff, 0x11, 0x03, 0x00, 0x00, 0x00, 0x00]$$

   and our rule array is:

$$R = [0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08]$$

   As $X = 0x02$, which is $R(2)$, automaton look at $W(2)$value, but $W(2) = 0xff$, which is fixed cell and this place can't be infected. Automaton will search for the first free place int he array, which is east. The resultant matrix is

$$N = \begin{bmatrix} 0x01 & 0xff & 0x11 \\ 0x03 & 0xff & 0x05 \\ 0x00 & 0x00 & 0x00 \end{bmatrix}$$

   Automaton will repeat infection until repeat number reaches $BranchNum$. You'll notice, that focused cell value is fixed to 0xff and east neighbor is set to corresponding byte value from R.

2) Strict direction: *X* takes its value from array:

$$R = [0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f, 0x10]$$

   and it works the similar way as preferred direction, but initial direction is carried to the free slot, id est: automaton tends to keep initial (strict) direction.

   Example: Let's assume that $BranchNum > 0$. While scanning container mask we have found the pixel with value $X = 0x0c$ and surrounding neighborhood is

$$N = \begin{bmatrix} 0x01 & 0xff & 0x11 \\ 0x03 & X & 0x00 \\ 0x00 & 0x00 & 0x00 \end{bmatrix}$$

   This neighbor matrix corresponds to array

$$W = [0x01, 0xff, 0x11, 0x03, 0x00, 0x00, 0x00, 0x00]$$

   and our rule array is:

$$R = [0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f, 0x10]$$

   As $X = 0x0c$, which is $R(4)$ automaton look at $W(4)$, but $W(4) = 0x03$ and can not be infected. Automaton will search for the first free place, which is east. This time, free slot is occupied with initial X value in order to force initial direction. The resultant matrix is

$$N = \begin{bmatrix} 0x01 & 0xff & 0x11 \\ 0x03 & 0xff & 0x0c \\ 0x00 & 0x00 & 0x00 \end{bmatrix}$$

   Automaton will repeat infection until repeat number reaches $BranchNum$.

3) Calculated direction: this group of rules are not represented by array corresponding to wind rose matrix, but rather by a scalar value:
   a) Random: this rule is represented with byte value 0x11 and it means that all directions from the wind rose are equally possible, direction is randomly

chosen. If this place is not occupied, value 0x11 is distributed to the new cell position. Old cell position is set to "fixed value". Again, if randomly chosen position is occupied, automaton will try to branch cell to free positions.

b) High: rule is represented with byte value 0x12. This value tells automaton to search original image neighboring pixel with the lowest intensity value (regarding the focused cell position). If such value is found automaton will populate container mask with the same vale 0x12, if not branching takes place.

c) Low: similar as above. Value is 0x13 and tells automaton to search for highest intensity value neighboring cell (around the current focused cell position).

Previously described rules are not in exclusive relations. Values from selected rules are added to "seed sack". Values from seed sack are randomly picked in seeding process. There are 2 more set of parameters associated to this section:

1) Growth semaphore: represented by 3x3 Boolean matrix with omitted middle element. Growth semaphore tells infector engine which directions are forbidden and which are allowed. For instance, semaphore:

$$S = \begin{bmatrix} 1 & 0 & 1 \\ 0 & & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

means that directions NW, NE, SW and SE are allowed, while all others are forbidden. Growth semaphore can be very useful when combined with calculated rules.

2) Additional branch controls: a set of four parameters that controls branching behavior:
   a) Fixed number: *BranchNum* is fixed. Each active cell found will have equal retry chance.
   b) Randomize branch number: *BranchNum* is randomized each time the active cell is found. New *BranchNum* value lies in interval $[0, BranchNum]$.
   c) Brighter=more branches: *BranchNum* is calculated using OI Pixel intensity (luminance can be used as well). New *BranchNum* value is obtained using formula: $BranchNum = \frac{intensity}{255} + 1$. When this option is used, OI pixels with high intensity will have more retries then those with low intensity.
   d) Darken=more branches: *BranchNum* is calculated from reverse intensity value. Similar to precious option, but this time OI pixels with low intensity will have more retries then pixels with high intensity.

V. Algorithm

THE ALGORITHM is actually very simple: We will use 3 images to perform this operation. One is original full color (24-bit) image (photography), which we will denote as OI, and two 8-bit masks *LM* and *RM*. Position at i-th row and j-th column of an image is denoted as (i, j). Value of pixel at this position is denoted as $OI\,(i, j)$. Maximum number of retries is denoted as *BranchNum* and maximal number of iterations *MaxIter*. Constant *LastRuleValue* is set to number of rules represented as byte value.

1) Create two 8-bit masks of the same size as original photography.

2) Populate *LM* with initial cells according to seeding method and set seed values according to automaton rule selected.

3) Copy *LM* to *RM*.

4) Scan through *LM* and *OI* (*i = 0 to ImageHeight, j = 0 to ImageWidth*)

5) If $(LM\,(i, j) > 0x00)$ AND $(LM\,(i, j) \leq LastRuleValue)$ find free slot according to $LM\,(i, j)$value (check *W* array vs. *R* array). Get free slot position $\left(i^f, j^f\right)$and set $OI\left(i^f, j^f\right) = OI\,(i, j)$. Set $LM\,(i, j) = 0xff$ and set $RM\left(i^f, j^f\right)$ to new mask value. If slot is occupied, try to branch cell *BranchNum* times.

6) Copy *RM* to *LM*.

7) Repeat steps from 4. to 6. until maximum number of iterations (*MaxIter*) is reached, or all cells are set to fixed (*0xff*).

You can notice that in step 5) we set new mask value in *RM* and not in *LM*. This is done for a good reason. If we put the same mask value in LM, uncontrollable branching would occur, as there is probability that this value would be available in next scanning step. Because of that, routine that searches for a free slot (getFreeSlot) checks mask values towards *RM*. Using two masks, ensures the full control over automata behavior.

VI. Infection color

OUTPUTpixel color *OI* described in previous section can be modified by additional options (*InfectionStyle*). The default option in Image Infector (use original pixel value) can be overridden by selecting another method that modifies *OI* value:

1) Use original pixel value: this is exactly the method described in step 5 of algorithm: $OI\left(i^f, j^f\right) = OI\,(i, j)$. Initial pixel color is actually propagated through the image.

2) Soften using 3x3 kernel: pixel value is obtained by

convolving $OI\,(i,j)$ pixel neighbors using kernel

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 9 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

and matrix *N*. Resultant pixel is then divided by 3 and saved as $OI\left(i^f, j^f\right)$.

3) Average two pixels: a weighted average of two pixels. Random pixel$OI\left(i^{rand}, j^{rand}\right)$, in the neighborhood of $OI\,(i,j)$ is picked, new pixel value is calculated as:

$$OI\left(i^f, j^f\right) = OI\,(i,j)\,p + OI\left(i^{rand}, j^{rand}\right)q$$

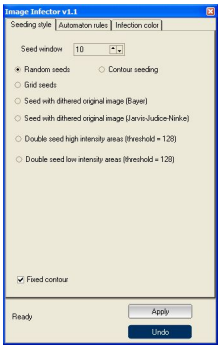
such that $p + q = 1$.

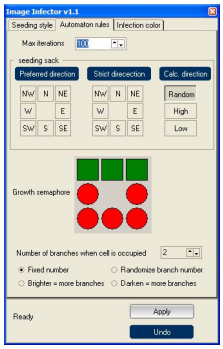
4) Custom kernel: pixel value is obtained by convolving $OI\,(i,j)$ pixel neighbors using specified kernel and matrix N, divided by div value, bias is added and new pixel value is saved at $OI\left(i^f, j^f\right)$.

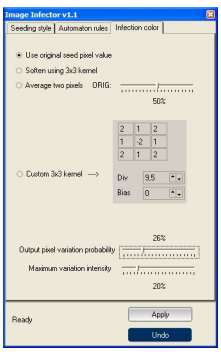
Regardless of selected option, output pixel color can be additionally modified by means of color variation (mutation). Two parameters control this option: *CVaryProb* and *CVaryIntensity*. *CVaryProb* controls probability of variation occurrence (in percentage), while *CVaryIntensity* controls maximum intensity of output color variation. Probability of variation occurrence is calculated as $p = \frac{CVaryProb}{100}$, while output pixel color is modified by formula: $OI\,(i,j) = OI\,(i,j) + CVaryIntensity - rand\,(CVaryIntensity * 2)$.

VII. Image Infector GUI

IMAGE Infector user interface follows previously described control groups: seeding, rules and infection color which are implemented as page tabs:







As you can see, groups explained in previous sections (seeding style, automaton rules and infection color) can be easily identified in the GUI. When all parameters are set, by clicking **Apply** button cellular automaton start working. To revert to original image, click **Undo** button.

VIII. Results

EXAMPLES in this paper are attached in their original size. To see full-sized images, use zooming tool, or you can simple copy and save images externally.

Figure 1. Portrait-original image

And now, some explanations about first few examples.

In **Figure 2**, random calc. rule is applied without contour fixing. You can notice jagged lines on the edges. Lighter areas are double seeded and semaphore is set to full green:

$$S = \begin{bmatrix} 1 & 1 & 1 \\ 1 & & 1 \\ 1 & 1 & 1 \end{bmatrix}$$ (Moore neighborhood).

In **Figure 3**, random calc. rule is applied with contour fixing. Contour is imported from Pixopedia 24 (Contour 7x7 filter) and edge is nicely preserved. Increasing variation probability to 40% and changing growth semaphore, a nice cork texture is obtained. Semaphore is set to:

$$S = \begin{bmatrix} 0 & 1 & 0 \\ 1 & & 1 \\ 0 & 1 & 0 \end{bmatrix}$$ (Von Neumann neighborhood)

to get combined horizontal and vertical spreading, without diagonal growth.

Figure 2. Random without contour
Seed style is set to "double seed high intensity", FixContour is OFF, SeedWindow=10, automaton rule set to Random with fixed size branches; BranchNum=2, MaxIter=1000, infection color to average two pixels(p=60%), CVaryProb=10% and CVaryIntensity=20%

Figure 3. Cork effect
Seed style is set to "double seed high intensity", FixContour is ON, SeedWindow=10, automaton rule set to Random with fixed size branches, BranchNum=3, MaxIter=1000, infection color to average two pixels(p=77%), CVaryProb=40% and CVaryIntensity=20%

In **Figure 4**, preferred direction rule is applied, with all directions in seed sack. Contour is OFF, and growth semaphore is set to :

$$S = \begin{bmatrix} 1 & 0 & 1 \\ 0 & & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

to force diagonal growth. With such parameters combination, a kind of glow effect is obtained.

In **Figure 5**, "high" calculated direction rule is applied. Contour is OFF, and growth semaphore is set to:

$$S = \begin{bmatrix} 1 & 1 & 1 \\ 0 & & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

to force upward growth. Automaton is applied **twice** and with such settings a nice fur effect is created.

Figure 4. Glow effect
Seed style is set to “random seeds”, FixContour is OFF, SeedWindow=6, automaton rule set to Preferred Direction (all directions allowed), more branches on brighter areas, BranchNum=3, MaxIter=5000, infection color to average two pixels(p=83%), CVaryProb=0%

Figure 5. Fur effect
Seed style is set to “double seed high intensity”, FixContour is OFF, SeedWindow=8, automaton rule set to High, randomize branch number, BranchNum=6, MaxIter=5000, infection color to average two pixels(p=80%), CVaryProb=80%, CVaryIntensity=15%.

Figure 6. Sea-original image

Figure 7. Impressionist effect
Seed style is set to “contour seeding”, automaton rule set to Random, randomize branch number, BranchNum=2, MaxIter=1000, infection color to original pixel value, CVaryProb=0%. Semaphore is set to full green. NOTE: when “contour seeding” is used, SeedWindow value is not used.

Figure 8. Upward contour glow
Seed style is set to “contour seeding”, automaton rule set to Strict direction (south only), randomize branch number, BranchNum=2, MaxIter=1000, infection color set to average two pixels (p=50%), CVaryProb=0%. Semaphore is set to full green.

IX. Conclusion

CELLULAR automata can play an important role in digital imaging in obtaining various artistic effects. What type of seeding, rules, infection color and contour (edge) detector to use greatly depends on image structure and coloring. It requires a little bit of experimentation, but an experienced user can achieve interesting results. More

Figure 9. Fruits-Original image

Figure 10. Crayon effect
Seed style is set to “double seed low intensity”, FixContour is ON, SeedWindow=20, automaton rule set to Strict direction (NE,SW), more branches on bright areas, BranchNum=7, MaxIter=5000, infection color to default custom kernel, CVaryProb=70%, CVaryIntensity=20%. Semaphore is set to full green.

work however needs to be done. In the time this paper is written, Image Infector version is 1.1. Few improvements are already under development and we can put them as future goal:

- infected pixel combination with some predefined foreground color. Convert low seeded areas to some background color and so on.
- more seeding variations: we can also use external 8-bit image as a seeding template, the same way we’ve used image contour for seeding. The best way to get nice effects is to use simless background tiles (canvases) like we have in Pixopedia 24.
- new automaton rules: we can add rule that force cells growth toward some attractor on the input image, to

choose direction depending on color difference (closest color). Also, we can add a rule that converts certain are of an image into specific color (color eater), rules that follow certain curvature, cells with limited age, cells that exchange colors when meet each other, etc...

- combining cellular automata and particle system: this is actually the main goal, to combine cellular automata with particle system. As mentioned in introductory section, presented systems already has some elementary particle system attributes (direction rules). Further experimentation will eventually lead to merging PS idea with CA.

Obviously, there is a wide area of possible research regarding artistic effects with CA. After all, there is more then 230 bytes values available for new rules.

REFERENCES

[1] Vit Krajcik. Cellular automata. *Faculty of Electrical Engineering and Information Technologies Slovak University of Technology Bratislava, Slovak Republic*.

[2] Adriana Popovici and Dan Popovici. Cellular automata in image processing. *Departments of Computer Science and Mathematics, University of the West Timisoara*.

[3] Qing Zhang, Youetsu Sato, Jun ya Takahashi, Kazunobu Muraoka, and Norishige Chiba. Simple cellular automaton-based simulation of ink behaviour and its application to suibokuga-like 3d rendering of trees. *The Journal of Visualisation and Computer Animation*, 10:27–37, 1999.