Pattern formation through minimalist biologically inspired cellular simulation

Supplementary material

Marcelo de Gomensoro Malheiros*
UNIVATES, Brazil
Universidade Federal do Rio Grande do Sul, Brazil

Marcelo Walter†
Institute of Informatics
Universidade Federal do Rio Grande do Sul, Brazil

1 Application

Our application is called **Pattern Explorer**. Source code for the complete implementation and all experiment files are publicly available at:

`http://github.com/mgmalheiros/pattern-explorer`

The simulation runs in real time, reading commands for an experiment from an ASCII text file (with **`.pex`** extension) and then opening a graphical window. Figure 1 shows the user interface for the application.

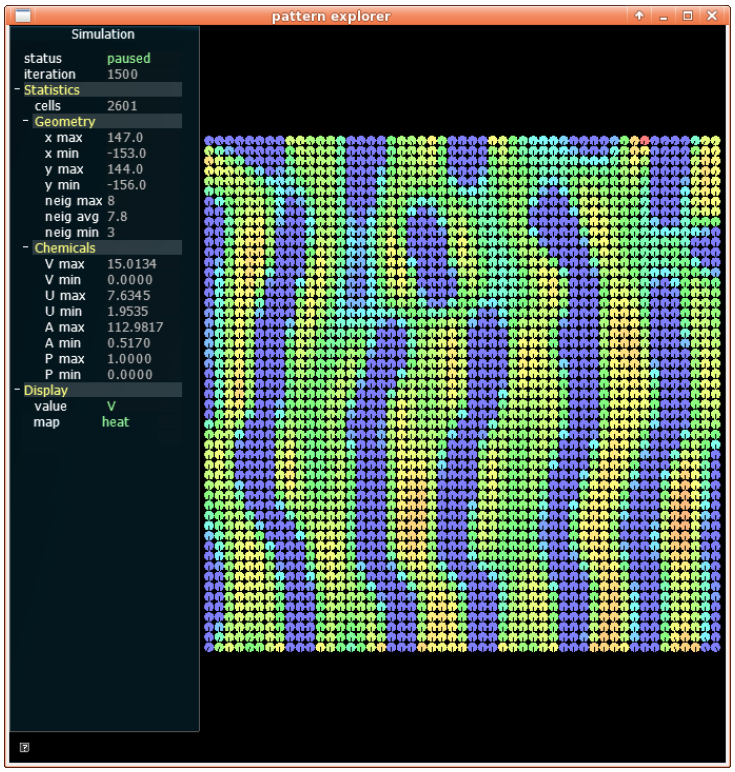

Figure 1: The interface for Pattern Explorer.

On the right, there is a graphical canvas where cells are drawn as filled circles, each with a line segment indicating their current orientation. On the left, there is an information panel that also enables some parameters to be manually changed. In this panel,

*e-mail: mgm@univates.br
†e-mail: marcelo.walter@inf.ufrgs.br

the simulation status (running or paused) is shown, besides the current iteration. Collapsible subpanels show several statistics for the simulations, including the number of cells. The Geometry panel shows maximum and minimum x and y coordinates, maximum and minimum cell radii, and also maximum, average and minimum number of neighbors. The Chemicals panel shows maximum and minimum concentrations for all defined chemicals. There is also a Display panel, controlling the color coding of cells, which maps the concentrations for a given chemical to a heat color map.

Additional control is possible through keyboard shortcuts and direct interaction using the mouse on the canvas, like panning, zooming or individual cell selection.

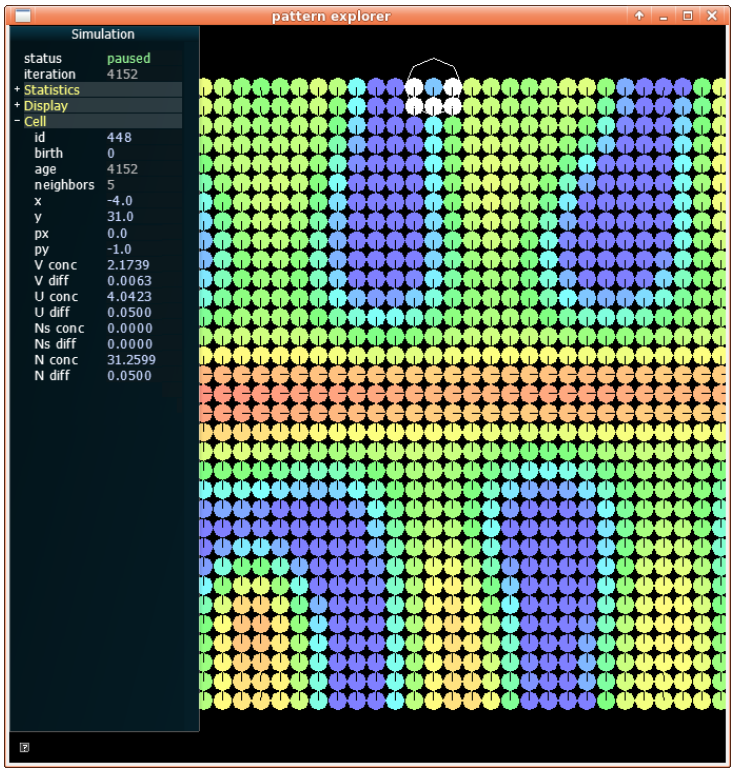

Figure 2: Individual cell information in Pattern Explorer.

In Figure 2 is shown an additional panel, called Cell, which displays the current state for a selected cell. The cell is identified by a circle drawn around it, showing the range search radius, which is used to select nearest neighbors. The nearest cells are also show filled in white. The subpanel shows the cell id, birth, age, number of neighbors, x and y coordinates and polarity (described as a unit vector with components px and py), and the current concentrations and diffusion rates for all chemical reagents.

2 Text-based specification

Each experiment is comprised of a set of commands and rules, which are described by a simple ASCII text file. The following sections give a brief overview of the simulation language. This is exactly the same language used to generate the results presented in our paper.

We strived for simplicity, by having one statement per line and using a simple grammar, directed by English words. There is currently no explicit limit for the number of cells, chemicals or rules within a given experiment. Lines preceded by `//` are considered comments.

2.1 Simulation parameters

Global simulation parameters are declared by **`define`** commands.

The commands shown in Listing 1 define general properties. The **`define chemical`** command defines the name of a new chemical reagent, and optionally both an upper limit for its concentration and if its diffusion is anisotropic (in relation to the polarity of each cell). The **`define division_limit`** command sets the maximum number of neighbors a cell must have to be able to perform division. The **`define domain`** command defines either a square packed domain (that grows based on the number of cells) or a fixed domain with given width and height. The **`define time_step`** command alters the integration step for the diffusion calculation.

```
define chemical <string> [limit float] [anisotropic]
define division_limit int
define domain [float float | packed [area float]]
define time_step float
```

Listing 1: Simulation parameters.

2.2 Default values

There are **`use`** commands to define default values for several cell attributes, depicted in Listing 2. Such values are used when issuing a **`create`** command to generate a new group of cells. The attributes can be changed again, affecting new cells created later on.

The **`use chemical`** command defines the initial concentration of a given chemical (and respective variation) and its default diffusion rate (and variation). Further calls to this command simply change the default values to be used during initial cell creation. The command **`use polarity`** sets a default orientation for cells, specified in degrees. Eventually, a nonpolarized state can be specified with **`use polarity none`**. The **`use seed`** command sets the initial value for the random number generator. A value of zero means to use the current time (that is, a distinct random sequence is used each time the experiment is run).

```
use chemical <string> [conc float [dev float]] [diff float [dev float]]
use polarity [none | float [dev float]]
use seed int
```

Listing 2: Commands for setting default values.

We only allowed a prepattern to be set, by the creation of a group of cells at once and the explicit definition of particular cell concentrations. After the simulation starts, these commands cannot be issued.

2.3 Cell placement

The Listing 3 shows commands for direct cell placement.

The **`create cell`** command makes a single new cell at the given position. The **`create sqr_grid`** command defines a square grid of some number of columns and rows, centered at a given position. The **`create sqr_circle`** command defines a circle with given radius and made with cells following a square grid. The **`create hex_grid`** command creates a hexagonal grid of cells, and likewise, the **`create hex_circle`** creates a circle with cells following a hexagonal arrangement. Finally, the **`create shape`** command creates cells using positions given by a text file (each alphabetic character is a cell, any other character is empty space), following a square grid arrangement.

The optional **dev** parameter indicates the amplitude of deviation from the positions actually set, whereas the optional **fixed** parameter signals those cells to not be affected by the eventual collision. The **wrap** keyword is specific for the square grid and indicates that a typical toroidal wrapping should be performed on the square borders (to make the resulting pattern tileable, for example).

```
create cell float float [fixed]
create sqr_grid   int int [at float float] [dev float] [fixed] wrap
create sqr_circle int     [at float float] [dev float] [fixed]
create hex_grid   int int [at float float] [dev float] [fixed]
create hex_circle int     [at float float] [dev float] [fixed]
create shape "file.txt"   [at float float]             [fixed]
```

Listing 3: Commands for creating and positioning cells.

2.4 Explicit parameter setting

There are also explicit commands to alter the concentration, diffusion rate or polarity for specific cells. Such instructions are also restricted to the definition of the prepattern, and just provide a finer control for altering the cells placed by the **create** commands.

The **set cell** and **set cells** commands are shown in Listing 4, which modifies the cells associated with a given index or index range. The index is an integer that sequentially increases, and is given to each cell after its placement. The GUI makes possible the visual selection of cells and exhibits their corresponding indexes.

```
set cell int chemical <string> [conc     float [dev float]]
                               [diff     float [dev float]]
                               [polarity float [dev float]] [fixed]
set cells int to int chemical <string> [conc     float [dev float]]
                                       [diff     float [dev float]]
                                       [polarity float [dev float]] [fixed]
```

Listing 4: Commands for altering chemicals for particular cells.

2.5 Rules

Rules are initiated by the keyword **rule** and are comprised of two parts: the condition and the action.

The condition has an optional part that defines the initial and final iteration counts where such rule must apply, as depicted in Listing 5. These iteration counters are simply integer values that follow the **from** or the **until** keywords. The remaining of the condition is either comprised of the **always** keyword, a numerical comparison or the **probability** keyword. As expected, the **always** keyword is always true, whereas the numerical comparison must be evaluated each time to be known. The **probability** keyword is followed by a real number between zero and one, which is the probability of being true (a random number is drawn for each occurrence of this condition). The overall effect is that the associated action is performed only when the current iteration is within the specified interval and when the condition is shown to be true.

```
rule [from int] [until int] always ...
rule [from int] [until int] if <par0> == <par1> ...
rule [from int] [until int] if <par0> != <par1> ...
rule [from int] [until int] if <par0> <  <par1> ...
rule [from int] [until int] if <par0> <= <par1> ...
rule [from int] [until int] if <par0> >  <par1> ...
rule [from int] [until int] if <par0> >= <par1> ...
rule [from int] [until int] if <par0> in <par1> <par2> ...
rule [from int] [until int] probability <par0> ...
```

Listing 5: Types of possible conditions for a rule.

Currently, the comparison parameters, here indicated by **par0**, **par1** and **par2**, can be real values or refer to the current cell state. Therefore is possible to refer to the concentration or diffusion rate of a given reagent by its name or the name of a mapping (described later on). Moreover, it is also possible to use its birth iteration (**BIRTH**), its current age (**AGE**) or the number of close neighbors (**NEIGHBORS**).

Note that the conditions involving iteration counts are not strictly necessary, as a similar control mechanism could be set within each cell by a specific reagent that does not diffuse, and that is increased by a tiny amount each iteration. We have thus opted to include the explicit references to the iteration count and cell age for convenience when designing patterns, due to the necessity of timed actions.

Almost all actions map to the cell events described in the paper. Listing 6 gives a brief overview of each one. The **change** rule modifies a concentration or diffusion rate for the given chemical value, which optional variation. As said before, only the given amount to be added (or subtracted) to the current state is specified: it is not possible to absolutely set to a given level. The **react** action performs the reaction part of a reaction-diffusion equation on two reagents specified by the given strings. The optional **scale** parameter permits controlling the overall size of stationary waves by scaling the magnitude of the reaction for both reagents. The **divide** action can trigger a cell division, that actually happens after the current iteration is finished. Optionally can be defined an offset direction and variation (in degrees) relative to the current cell polarity.

```
... change <par0> float [dev float]
... react <string> <string> [scale <par0>] turing alpha <par1> beta <par2>
... divide [direction float [dev float]]
... map <par0> float float to <string> float float
... polarize <string>
... and
```

Listing 6: Types of possible actions for a rule.

The **map** action is just a convenient mechanism for converting real quantities. It does not have any direct effect on the simulation: it simply defines a computed value that is associated with a string. This string can then be used as a parameter in other conditions or actions. The idea is to linearly map the given real interval from **par0** to another interval. The computed values are always restricted to be in the output interval (that is, the value is clamped when over the lower or upper limits). The **polarize** action adjusts the current cell polarity to follow the local gradient of the given chemical name. Finally, the **and** keyword combines the condition for this rule to the condition of the following rule, making a logical "and". Therefore, both conditions must be true for the following action to be triggered.

```
... react <string> <string> [scale <par0>] cubic a <par1> b <par2> c <par3>
... react <string> <string> [scale <par0>] gm1 a <par1> b <par2> c <par3>
... react <string> <string> [scale <par0>] gray-scott f <par1> k <par2>
... react <string> <string> [scale <par0>] linear au <par1> bu <par2> cu <par3> du <par4> mu <par5>
                                               av <par6> bv <par7> cv <par8> dv <par9> mv <par10>
... move float [dev float]
```

Listing 7: Types of experimental actions.

Listing 7 shows the currently experimental actions, which are still being evaluated. The **cubic** type implements a single-reagent cubic equation that affects only the first reagent, being a continuous approximation for the discrete rules that generate reinforcement patterns. The **gm1**, **gray-scott** and **linear** implement other two-reagent RD equations, as in [6]. The **gm1** is a particular Gierer-Meinhardt model, whereas **linear** is the generalization of Turing's original linear RD system [10] (where **mu** and **mv** are the upper limits for the synthesis of the first and second reagents, respectively). The **move** action moves the cell along its polarity direction, by the given offset and respective deviation.

2.6 Utility commands

Listing 6 gives a brief overview of the utility commands implemented so far. Most are not directly related to the simulation itself, but they help on mapping concentration values to output colors, automate taking snapshots of the pattern or interrupt the simulation at a given iteration.

```
colormap select [heat | striped | gradient]
colormap slot int use color "<string>"
colormap slot int use rgb int int int
snap at int [...] [then exit]
snap from int repeat int step int [then exit]
stop at int
texture size int int
zoom level float
```

Listing 8: Utility commands.

The **colormap** commands define a color map, which can be used to map the concentration of a reagent to a set of given colors. The **heat** type is the traditional hue transition from blue (zero) to red (one). The **striped** and **gradient** types create a color map based on colors placed on 100 slots, either by simple repetition or by linear interpolation, respectively. The color themselves can be specified by a CSS color string, a hex triplex or by individual RGB byte values.

The **snap** commands generate automated screenshots to image files in the PNG format, at the given iteration counts. The **stop** command simply interrupts the simulation at the given iteration. The **texture size** command defines the output resolution for the high-quality interpolated texture. Finally, the **zoom level** command sets an initial zoom to be seen on the GUI.

2.7 Language overview

In this section, we briefly summarize the commands of our text-based language. In Table 1 we list all statements, grouped by their category. We then link each command to the corresponding section where it is defined.

Table 1: Summary of language statements.

Statement	Category	Description
`define chemical`	simulation parameters	Section 2.1
`define division_limit`		
`define domain`		
`define time_step`		
`use chemical`	default values	Section 2.2
`use polarity`		
`use seed`		
`create cell`	cell placement	Section 2.3
`create sqr_grid`		
`create sqr_circle`		
`create hex_grid`		
`create hex_circle`		
`create shape`		
`set cell`	explicit parameters	Section 2.4
`set cells`		
`rule always ...`	rule conditions	Section 2.5
`rule if ...`		
`rule probability ...`		
`... change`	rule actions	Section 2.5
`... react`		
`... divide`		
`... map`		
`... polarize`		
`... and`		
`... move`		
`colormap`	utility commands	Section 2.6
`snap`		
`stop`		
`texture`		
`zoom`		

3 Experiments

In this section, we present some extra experiments and also the rules used for the results shown in our paper. A short summary of timing and other details are presented in Table 2.

Table 2: Detailed information for figures.

Figure	Description	Cell count	Reagents	Iterations	Total time (s)	cell arrangement	Output type
3	Turing scale	3600	2	4000	3.038	square grid	screenshot
4	Modulation	3600	4	10000	7.278	square grid	screenshot
5	Anisotropic rays	10000	4	10000	71.918	hexagonal grid	screenshot
6	Anisotropic rings	10000	4	18000	125.158	hexagonal grid	screenshot
7	Growing domain	2400	2	60000	47.762	free	screenshot
8	Shell pattern	10400	2	26000	86.066	free	screenshot
9	Spiral	7501	2	15000	53.786	free	screenshot
10	Uniform growth	5248	2	15000	37.292	free	screenshot
11	Growth on borders	6111	2	20000	36.313	free	screenshot
12	Branching	6602	2	20000	17.491	free	screenshot
13	Frozen pattern	5248	2	15000	36.159	free	screenshot
14	Reinforcement	5248	3	15000	38.599	free	screenshot
15	Game of life	625	2	160	0.765	square grid	screenshot
16	Wireworld	225	3	72	0.985	square grid	screenshot
17	Laplacian growth	10000	3	3200	2.919	square grid	screenshot
18	Cow	10000	1	2000	3.098	square grid	screenshot
19	Dalmatian	10000	1	2000	3.072	square grid	screenshot
20	African wild dog	10000	1	2000	3.094	square grid	screenshot
21	Giraffe	10000	2	38000	54.722	square grid	screenshot
22	Nudibranch	10000	2	3000	4.801	square grid	screenshot
23	Poison dart frog	10000	2	8000	8.135	square grid	texture
24	Moray eel	10000	3	5000	21.122	square grid	texture
25	Leopard	6228	2	14000	22.298	free	texture

The current simulator is implemented in C++ on a Ubuntu 14.04 Linux system. The test machine is powered by an Intel Core i7-4500U CPU running at 1.80 GHz.

3.1 Turing patterns

In Figure 3, we present a simple experiment where isolated groups of cells have distinct diffusion rates. Such rates for chemicals U and V have the same proportion, though, which gives rise to different scales of the same kind of labyrinthine Turing pattern. This is done to illustrate that reaction-diffusion only depends on local parameters. Thus, we can compare the visual effect of altering parameters in a single simulation.

```
// Turing patterns

define chemical U
define chemical V

// -------- prepattern

use seed 1
use chemical U conc 4 dev 0.5 diff 0.01   dev 0
use chemical V conc 4 dev 0.5 diff 0.0005 dev 0
create sqr_grid 30 30 at -35 35

use seed 1
use chemical U conc 4 dev 0.5 diff 0.02  dev 0
use chemical V conc 4 dev 0.5 diff 0.001 dev 0
create sqr_grid 30 30 at 35 35

use seed 1
use chemical U conc 4 dev 0.5 diff 0.04  dev 0
use chemical V conc 4 dev 0.5 diff 0.002 dev 0
create sqr_grid 30 30 at -35 -35

use seed 1
use chemical U conc 4 dev 0.5 diff 0.08  dev 0
use chemical V conc 4 dev 0.5 diff 0.004 dev 0
create sqr_grid 30 30 at 35 -35

// -------- rules

rule always react U V scale 0.004 turing alpha 16 beta 12

stop at 4000
```

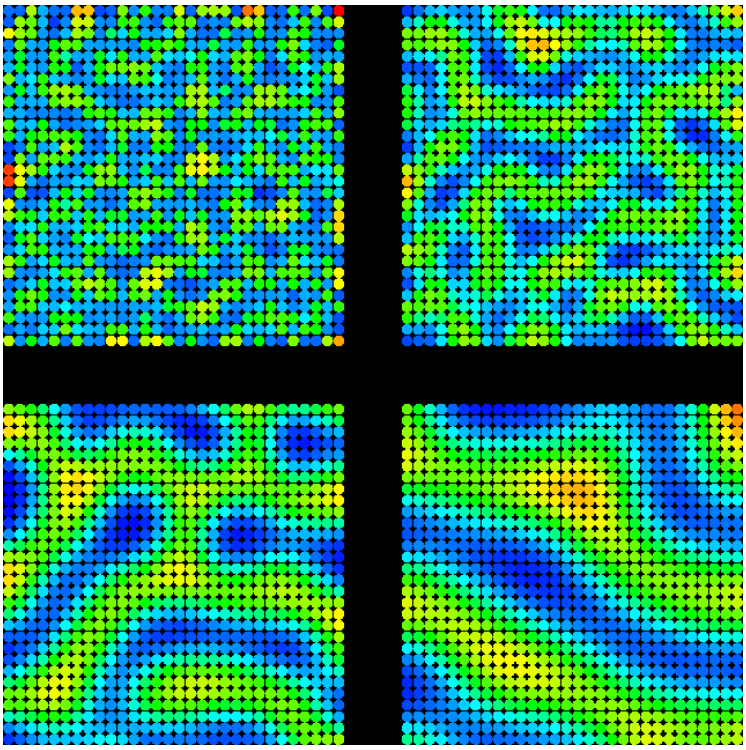

Figure 3: Effect of diffusion rate on the scale of patterns.

3.2 Parameter modulation

Another simple feature of our model is the spatial control of some parameter, based on a given chemical concentration. We call this "modulation", and it can be achieved by simply creating a mapping of real values through the **map** action. This is the method used in [9] for their specific patterns.

In Figure 4, we show both the pattern defined by the V reagent and the gradient on the A reagent. The linear gradient was created by the row of producer cells on the bottom, which is then used to control the β parameter for the Turing equations. We also wait until the gradient stabilizes in iteration 1000, then triggering the reactions. This results in a continuous transition from stripes to spot patterns.

```
// modulation

define chemical V
define chemical U
define chemical A
define chemical P

// -------- prepattern

use chemical V conc 4 dev 0.5 diff 0.001
use chemical U conc 4 dev 0.5 diff 0.0225
use chemical A conc 0         diff 0.05
use chemical P conc 0         diff 0
create sqr_grid 60 60

set cells 0 to 59 chemical P conc 1

// -------- rules

rule if P conc >= 1 change A conc  0.05
rule if P conc < 1  change A conc -0.001
rule always map A conc 1 7 to BETA 10 12
rule from 1000 always react U V scale 0.004 turing alpha 16 beta BETA

stop at 10000
```

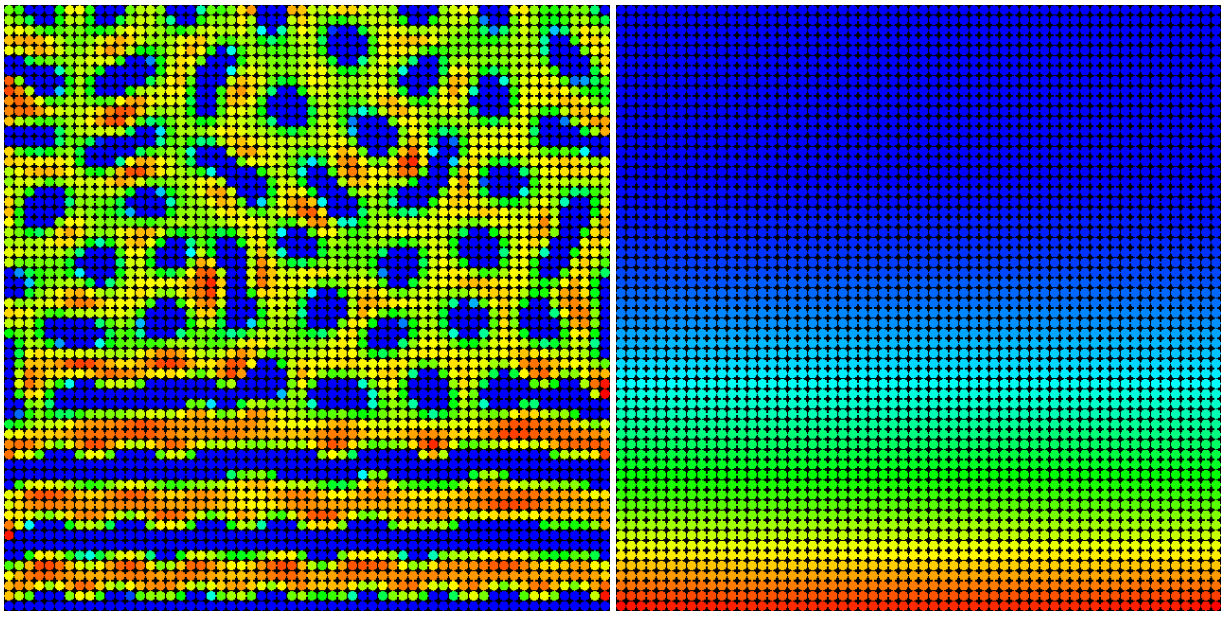

Figure 4: Parameter modulation for Turing pattern (left) based on a linear gradient (right).

3.3 Gradient and anisotropy

Here we show the input files for the examples discussed in the paper.

```
// rays

define chemical V anisotropic
define chemical U
define chemical P
define chemical G

// -------- prepattern

use chemical V conc 4 dev 1 diff 0.00625
use chemical U conc 4 dev 1 diff 0.024
use chemical P conc 0       diff 0
use chemical G conc 0       diff 0.05

create hex_grid 100 100

set cell 5050 chemical P conc 1

// -------- rules

rule if P conc == 1 change G conc  4
rule if P conc == 0 change G conc -0.001

rule always polarize G conc

rule always react U V scale 0.004 turing alpha 16 beta 12

colormap select gradient
colormap slot 0  use color "white"
colormap slot 99 use color "420009"

stop at 10000
```

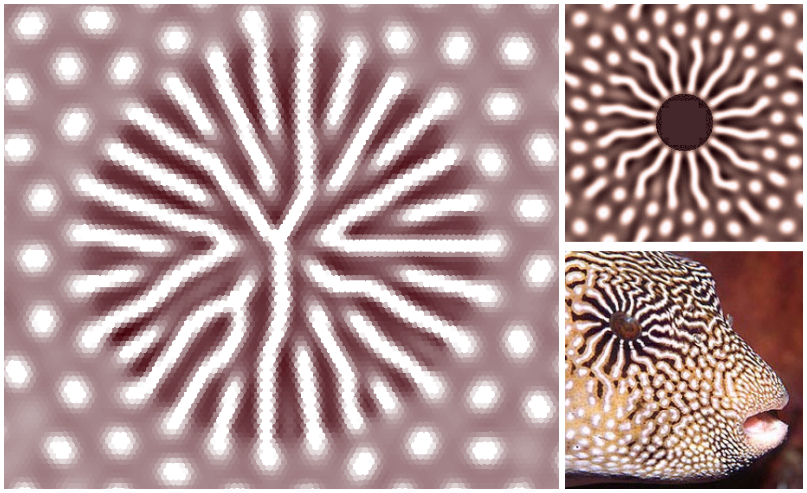

Figure 5: Effect of anisotropic diffusion for V reagent only (left). Similar results from [9], made by parameter modulation, are shown in the right column.

```
// rings

define chemical V
define chemical U anisotropic
define chemical P
define chemical G

// -------- prepattern

use chemical U conc 4 dev 1 diff 0.03
use chemical V conc 4 dev 1 diff 0.00625
use chemical P conc 0       diff 0
use chemical G conc 0       diff 0.05

create hex_grid 100 100

set cell 5050 chemical P conc 1

// -------- rules

rule if P conc >= 1 change G conc  4
rule if P conc <  1 change G conc -0.001

rule always polarize G conc

rule from 3000 always react U V scale 0.006 turing alpha 16 beta 12

colormap select gradient
colormap slot 40 use color "0ee1da"
colormap slot 80 use color "80763e"

stop at 18000
```

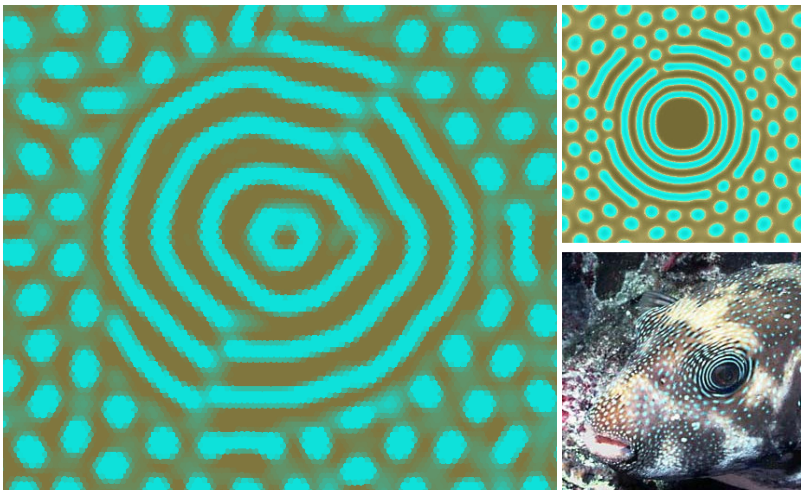

Figure 6: Effect of anisotropic diffusion for U reagent only (left). Similar results from [9], made by parameter modulation, are shown in the right column.

3.4 Simple growth patterns

In this section, we present some simple patterns that emerge from the combination of reaction-diffusion and cell division.

The first example is shown in Figure 7, where the effect of expanding domain (like in [7]) is simulated by a continually growing front that reaches the limits of a constrained simulation area. We can observe the appearance of new stripes as the tissue gets larger, maintaining the usual spatial wavelength of periodic stripes common to the Turing patterns. The growth movement is created by restricting that only newer cells may divide.

```
// limited domain

define chemical U
define chemical V

define domain 180 180

// -------- prepattern

use chemical V conc 4 dev 0.5 diff 0.001
use chemical U conc 4 dev 0.5 diff 0.008
use polarity -90

create sqr_grid 20 1 dev 0.1

// -------- rules

rule always react U V scale 0.001 turing alpha 16 beta 12
rule if AGE in 500 500 divide direction 0 dev 0

stop at 60000

zoom level 1.0
```

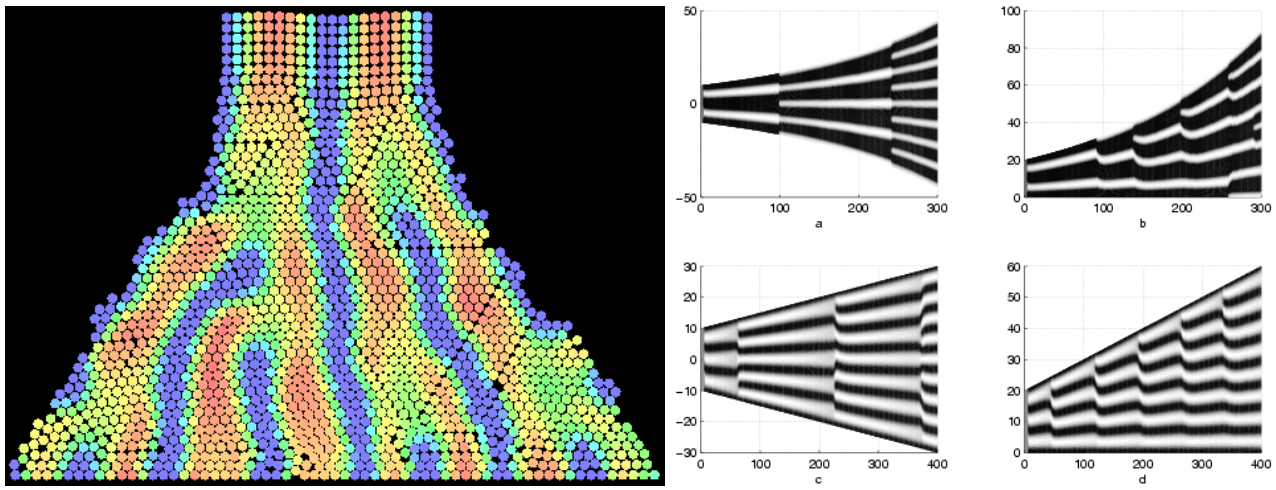

Figure 7: Effect of expanding domain generated by constrained growth, on the left, and results from [7], on the right.

A similar growing front can be created to freeze an active reaction-diffusion system in time. In the example shown in Figure 8, we keep the row of cells at bottom both dividing and with an active reaction. After division, the pattern is frozen on the parent cells, by making the diffusion rate drop to zero, whereas reaction continues on the newborn cells, which again divide in the next iteration. The result is a linear stripe pattern similar to a growing mollusk shell [2].

```
// front growth

define chemical V
define chemical U

define time_step 0.1

// -------- prepattern

use polarity -90

use chemical V  conc 4 dev 0.5  diff 0.01
use chemical U  conc 4 dev 0.5  diff 0.08

create sqr_grid 80 1

// -------- rules

rule if AGE in 200 200 divide direction 0 dev 0
rule if AGE <  200 react U V scale 0.01 turing alpha 16 beta 12
rule if AGE in 200 200 change V diff -0.01
rule if AGE in 200 200 change U diff -0.08

colormap select gradient
colormap slot 0 use color "dd9861"
colormap slot 60 use color "3f0400"
colormap slot 80 use color "782e07"

stop at 26000
```

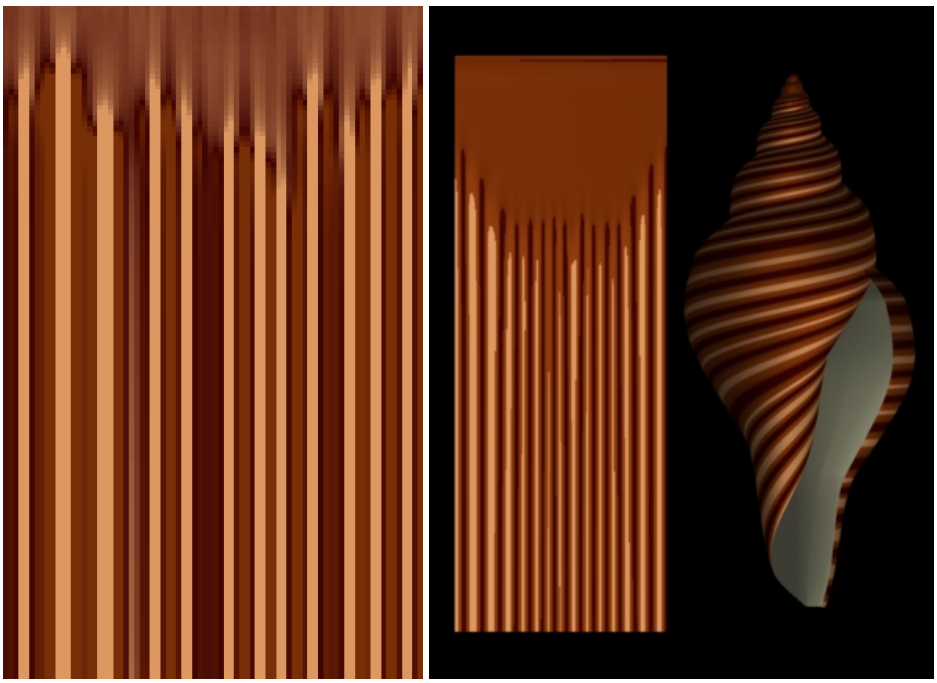

Figure 8: Growing front that freezes the older generated pattern, on the left, and results from [2], on the right.

Another interesting structural pattern emerges when we keep dividing only the last cell and enforcing that the newborn cells are at exactly 137.5 from their parents (Figure 9). The result is that the divisions occur mostly at the same spot, in the center of the pattern, evenly spreading older cells in an almost perfectly regular arrangement, similar to the natural phenomena of Spiral Phyllotaxis [3]. Continuing growth and reaction then define concentric patterns around the borders.

```
// center growth

define chemical V
define chemical U

// -------- prepattern

use chemical V conc 4 dev 0.5 diff 0.001
use chemical U conc 4 dev 0.5 diff 0.008

create cell 0 0

// -------- rules

rule always react U V scale 0.001 turing alpha 16 beta 12
rule if AGE in 1 1 divide direction 137.5 dev 0

stop at 12000

zoom level 1.309
```

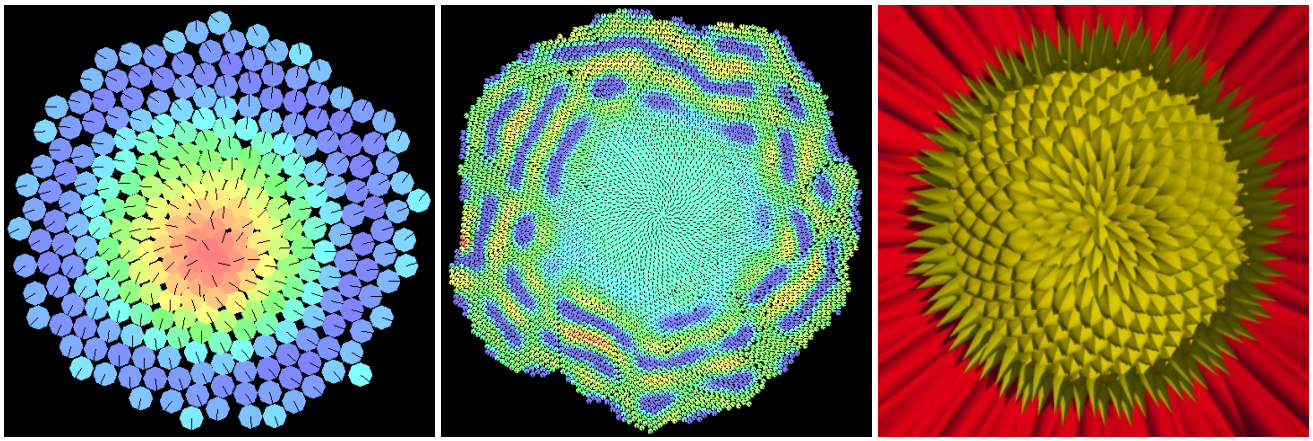

Figure 9: Spiral phyllotaxis initial structural pattern (left) and later concentration pattern (center). A similar structure from [3] is shown on the right.

3.5 Growth on borders

Here we briefly discuss experiments where growth is constrained to occur in pattern borders. Thus, we can use the mechanism to determine how many neighbors a given cell has to start cell division. In this way, the tissue expansion occurs only at its edges. By affecting the polarity of the newborn cell, it can result in either directional or random growth.

Figure 10 shows two variations of the same experiment, where cells divide only if they have three or fewer neighbors. We start from a single cell. Anisotropic diffusion is kept for one of the two reagents. A reaction is then performed at each iteration to induce oriented patterns. On the left, the division spawns cells with arbitrary polarity, whereas on the right, each new cell diverges exactly 10 from its parent. This is similar to the general process of Diffusion-Limited Aggregation (DLA), used for the modeling of lichen growth [1], for example. The input file for the both variations is shown.

```
// border growth

define chemical V anisotropic
define chemical U

// -------- prepattern

use chemical U conc 4 dev 1 diff 0.05
use chemical V conc 4 dev 1 diff 0.00625

create cell 0 0

// -------- rules

rule always react U V scale 0.004 turing alpha 16 beta 12

rule if NEIGHBORS <= 3 and

rule probability 0.004 divide direction 0 dev 180 // left image

//rule probability 0.004 divide direction 10 dev 0 // right image

stop at 16000

zoom level 1.127
```

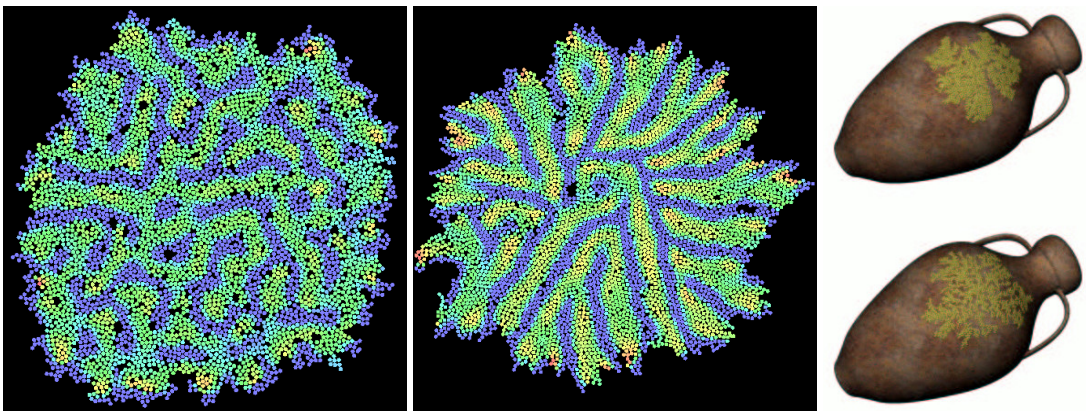

Figure 10: Pattern growth at borders: random division (left) and directed division (center). On the right, we show two results from [1].

In another experiment, inspired by the rules of a stochastic L-system, we have constrained division to occur when a cell has two or fewer neighbors. As this restricts division to very loose cells, it tends to generate filaments in an approximate linear growth. The experiment depicted in Figure 11 adds another criterion for subdivision: a less probable directional division of 20 from its parent. This results in a bifurcation, which then creates a branched structure. As division is restricted to cells on the "tip" of each branch, as soon as they collide with other structures the neighborhood gets denser and division stops locally.

```
// branching growth

define chemical V anisotropic
define chemical U

// -------- prepattern

use seed 2
//use seed 4

use chemical U conc 4 dev 1 diff 0.05
use chemical V conc 4 dev 1 diff 0.00625
use polarity 90

create cell 0 0

// -------- rules

rule always react U V scale 0.008 turing alpha 16 beta 12

rule if NEIGHBORS <= 2 and
rule probability 0.008 divide direction   0 dev 0

rule if NEIGHBORS <= 2 and
rule probability 0.0008 divide direction  20 dev 0

rule if NEIGHBORS <= 2 and
rule probability 0.0008 divide direction -20 dev 0

stop at 15000

zoom level 0.447
```

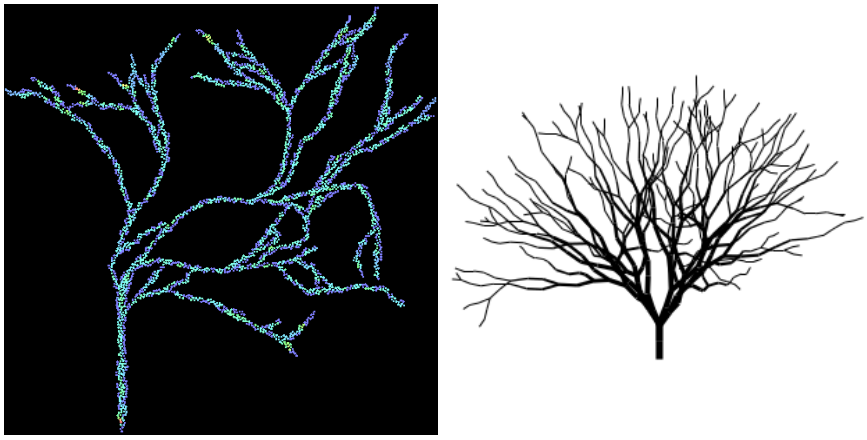

Figure 11: Randomly branched pattern (left) and one result from [8] generated by a stochastic L-system (right).

3.6 Pattern enlargement

Here we show the input files for the examples discussed in the paper.

```
// uniform growth

define division_limit 6

define chemical V
define chemical U

// -------- prepattern

use seed 1

use chemical U conc 4 dev 2 diff 0.15
use chemical V conc 4 dev 2 diff 0.01

create hex_circle 45

// -------- rules

rule always react U V scale 0.01 turing alpha 16 beta 12

rule from 2000 until 15000 probability 0.0001 divide direction 0 dev 180

stop at 15000

zoom level 1.303
```

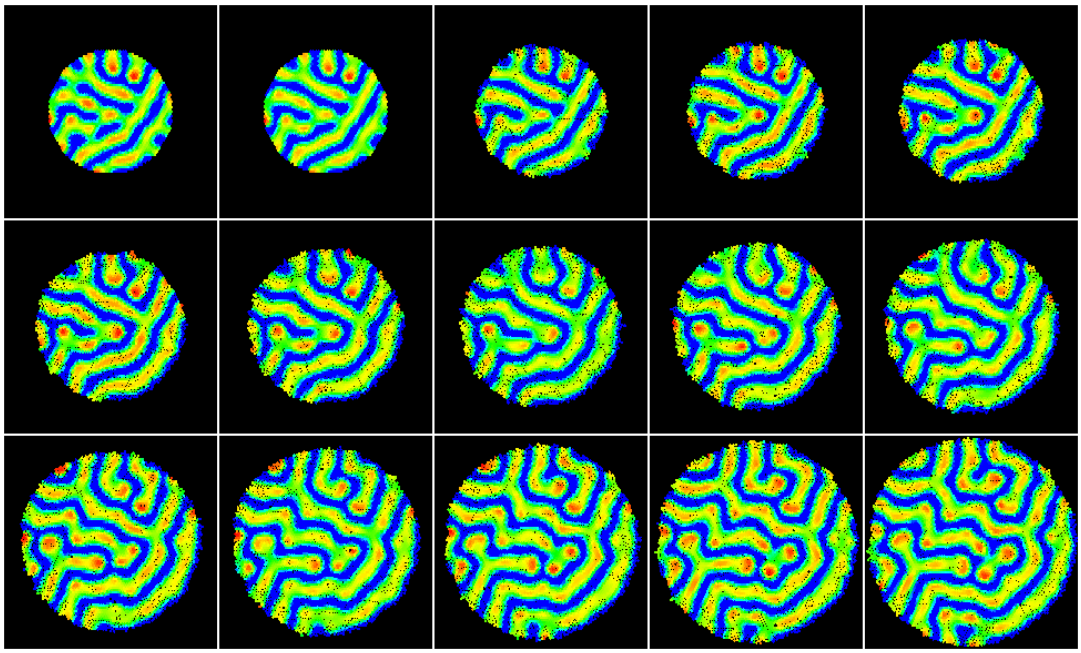

Figure 12: Pattern growth: standard RD behavior.

```
// frozen chemicals

define division_limit 6

define chemical V
define chemical U

// -------- prepattern

use seed 1

use chemical U conc 4 dev 2 diff 0.15
use chemical V conc 4 dev 2 diff 0.01

create hex_circle 45

// -------- rules

rule until 1998 always react U V scale 0.01 turing alpha 16 beta 12

rule from 1999 until 1999 always change V diff -0.01 // stop diffusion

rule from 2000 until 15000 probability 0.0001 divide direction 0 dev 180

stop at 15000

zoom level 1.303
```

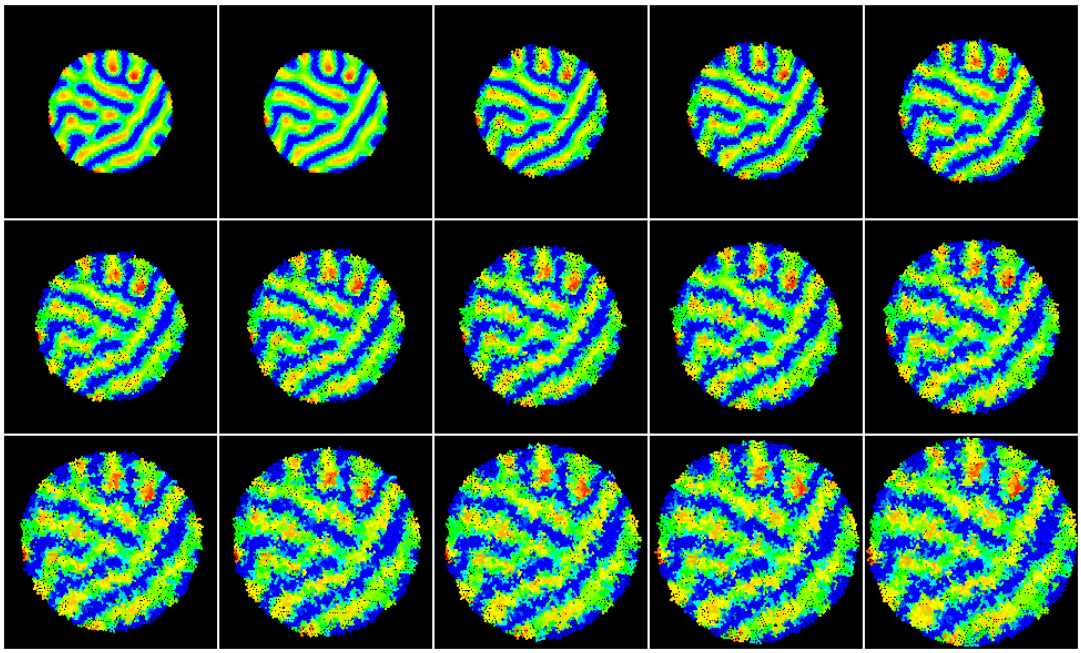

Figure 13: Pattern growth: pigmentation is frozen with no reaction and zero diffusion.

```
// reinforcement

define division_limit 6

define chemical P
define chemical V
define chemical U

// -------- prepattern

use seed 1

use chemical P conc 0       diff 0.0004
use chemical V conc 4 dev 2 diff 0.01
use chemical U conc 4 dev 2 diff 0.15

create hex_circle 45

// -------- rules

rule always react U V scale 0.01 turing alpha 16 beta 12

// copy pattern from V to P, mapping it to the [0,2] interval
rule from 1999 until 1999 always map V conc 1.0 7.0 to PAT 0.0 2.0
rule from 1999 until 1999 always change P conc PAT

// reinforcement of P
rule from 1999 if P conc in 0.1 0.9 change P conc -0.0004
rule from 1999 if P conc in 1.1 1.9 change P conc +0.0004

rule from 2000 until 15000 probability 0.0001 divide direction 0 dev 180

stop at 15000

zoom level 1.303
```

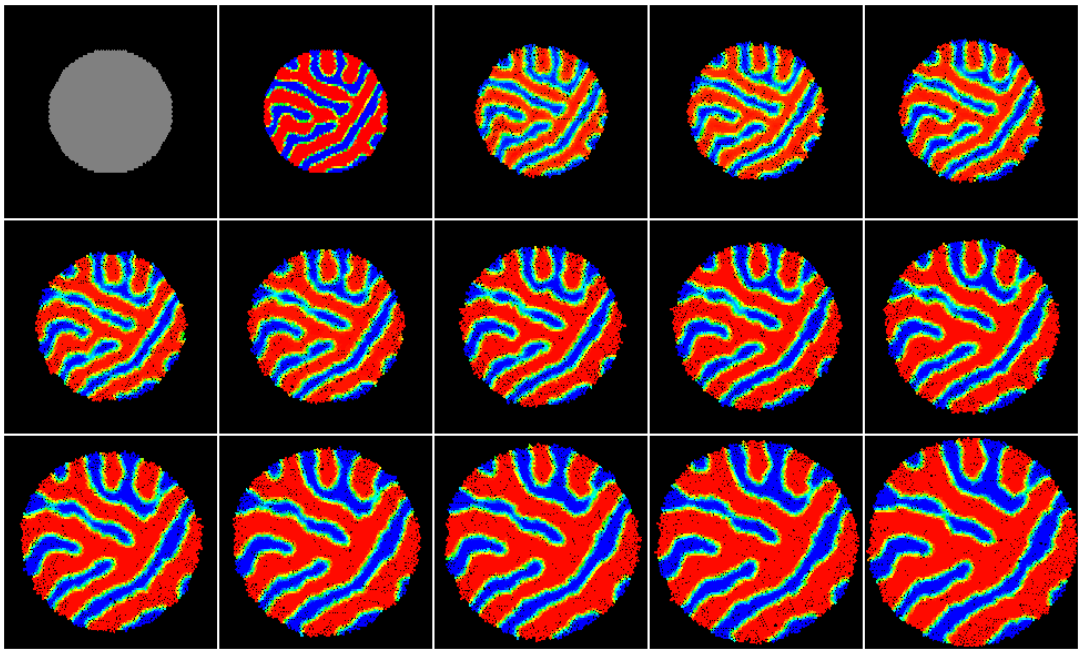

Figure 14: Pattern growth: pigmentation is copied and reinforced in a third chemical.

3.7 Cellular automata

Here we have reproduced the classic Game of Life [4] cellular automaton. The prepattern for this experiment defines the well-known *glider* from Game of Life, made of alive cells. It is a cyclic pattern, that slides one cell down and one cell to the right each four iterations. After this cycle, it returns to its original form. Figure 15 shows the state at iteration zero (left), and then the state at iteration 80 (right).

```
// game of life

define chemical L limit 1
define chemical C

// -------- prepattern

use chemical L conc 0 diff 0.01
use chemical C conc 0 diff 0

create sqr_grid 25 25
set cell 526 chemical L conc 1
set cell 527 chemical L conc 1
set cell 528 chemical L conc 1
set cell 553 chemical L conc 1
set cell 577 chemical L conc 1

// -------- rules

rule if C conc == 0 change C conc +1
rule if C conc == 1 change C conc -1

// kill all positions by default
rule if C conc == 1 change L conc -1

// a position with three neighbors becomes populated
rule if C conc == 1 and
rule if L conc in 0.025 0.034 change L conc +2

// a position with two or three neighbors survives to next generation
rule if C conc == 1 and
rule if L conc in 0.935 0.954 change L conc +2

stop at 160
```

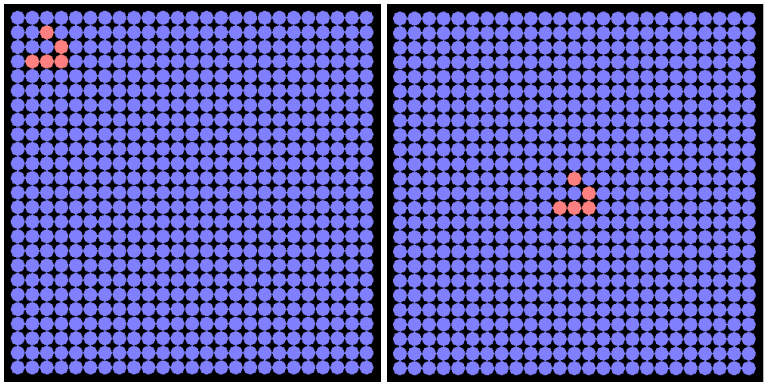

Figure 15: Game of Life: starting state (left) and after 80 iterations (right).

We first establish a regular square grid where positions can be either “dead” or “alive” in the sense of the automaton. We map these two states to the reagent *L*, being 0 and 1, respectively. As there is no explicit mechanism to sense neighbors, we set the diffusion rate of *L* to be slow, and then apply conditions based on the concentration after just one iteration. For example, a dead position with just one alive neighbor will get a slight increase in its own concentration for *L*, whereas an alive position with four empty neighbors will have a proportional decrease in *L*. Depending on the current concentration of L we can either consume all of it, to reach a dead state, or produce more chemical, to reach an alive state. We have set the saturation level for *L* to be 1, by using the **`limit`** keyword.

There is, however, another issue to be handled: by definition, actions started by rules and diffusion occur simultaneously. Therefore, it is adequate to set reagent levels only at an even iteration, skip an iteration (so only the normal diffusion occurs) and then test chemical levels again. For this process, we have created an artificial “clock” reagent called *C*, which is alternatively set to 0 or 1, marking even and odd iterations. This chemical has zero diffusion rate.

The Wireworld cellular automaton takes four distinct states, and we have reproduced it with a more general approach, by using three reagents and cycle of three (controlled by reagent *C*). Figure 16 shows the first few iterations of a clock-generating mechanism defined in the prepattern. The color scheme is the same used by the Golly package[1].

```
// wireworld

define chemical S
define chemical N
define chemical C

// -------- prepattern

use chemical S conc 0 diff 0
use chemical N conc 0 diff 0.01
use chemical C conc 0 diff 0

create sqr_grid 15 15

set cells 77 to 79 chemical S conc 3
set cell 61 chemical S conc 3
set cell 65 chemical S conc 3
set cell 46 chemical S conc 3
set cells 50 to 58 chemical S conc 3
set cell 31 chemical S conc 3
set cell 35 chemical S conc 3
set cell 17 chemical S conc 2
set cell 18 chemical S conc 1
set cell 19 chemical S conc 3

// -------- rules

// periodic clock
rule if C conc == 0 change C conc +1
rule if C conc == 1 change C conc +1
rule if C conc == 2 change C conc -2

// clock 0: mark head [1] in the neighborhood
rule if C conc == 0 and
rule if S conc == 1 change N conc +1

// clock 1: diffusion happens in the neighborhood

// clock 2: force neighborhood to zero
rule if C conc == 2 change N conc -1

// clock 2: empty [0] -> empty [0]

// clock 2: head [1] -> tail [2]
rule if C conc == 2 and
rule if S conc == 1 change S conc +1

// clock 2: tail [2] -> wire [3]
rule if C conc == 2 and
rule if S conc == 2 change S conc +1

// clock 2: wire [3] -> head [1] if exactly one or two neighbor cells are heads, otherwise keep as wire
rule if C conc == 2 and
rule if S conc == 3 and
rule if N conc in 0.01 0.02 change S conc -2

colormap select striped
colormap slot 00 use color "303030" // empty
colormap slot 25 use color "0080ff" // head
colormap slot 50 use color "ffffff" // tail
colormap slot 75 use color "ff8000" // wire

stop at 72
```

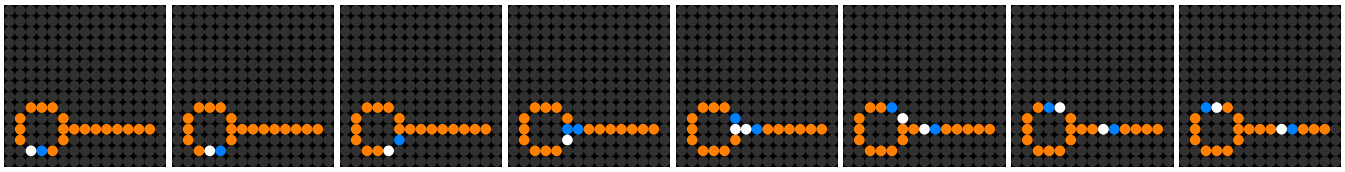

Figure 16: Wireworld automata implementing a clock-generating mechanism.

[1] http://golly.sourceforge.net/

A more complex experiment can be made by coupling a two-state automaton and an extra diffusing chemical. In the left of Figure 17 we show an approximation for a Laplacian Growth process (more specifically, the DBM technique described in [5]). The basic idea is to grow a pattern from an initial group of active cells (shown in red), updating at each iteration a probability field. Only non-active blue cells that are adjacent neighbors of red cells can be activated, and this process is governed by the probability given by the concentration of a reagent that is continually consumed by active cells, shown in the right of Figure 17. Therefore, the higher probabilities are far from the active cells, where new fronts are expected to grow.

```
// laplacian growth

define chemical P limit 1
define chemical G
define chemical C

// -------- prepattern

use chemical P conc 0   diff 0.01
use chemical G conc 0.1 diff 0.1
use chemical C conc 0   diff 0

create sqr_grid 100 100
set cell 4949 chemical P conc 1
set cell 4950 chemical P conc 1
set cell 4951 chemical P conc 1

// -------- rules

// clock
rule if C conc == 0 change C conc +1
rule if C conc == 1 change C conc -1

// force all to zero
rule if C conc == 1 change P conc -1

// some non-active neighbors will be activated
rule if C conc == 1 and
rule probability G conc and
rule if P conc in 0.03 0.04 change P conc +2

// keep previously activated
rule if C conc == 1 and
rule if P conc > 0.9 change P conc +2

// change probability gradient
rule if P conc == 0 change G conc +0.0001
rule if P conc == 1 change G conc -0.01

stop at 3200
```

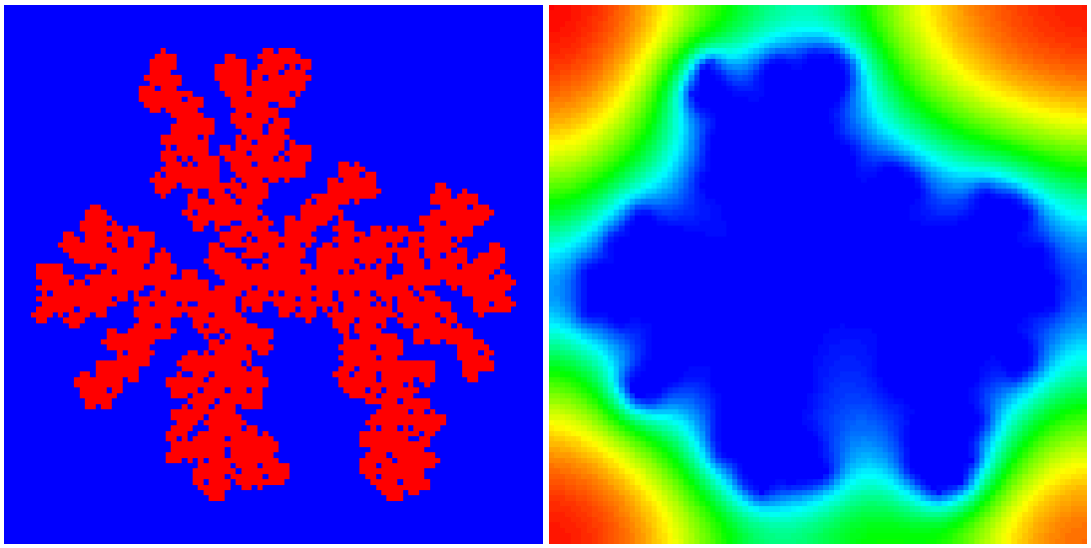

Figure 17: Approximate reproduction of Laplacian Growth using the DBM technique described in [5]. On right the active (red) and inactive (blue) cells. The probability field is shown on the right (blue is the lowest, whereas red is the highest).

3.8 Reinforcement patterns

The complete input files for the cow, Dalmatian, and African wild dog are provided below.

```
// cow-like spots

define chemical P limit 1

// -------- prepattern

use seed 2

use chemical P conc 0.5 dev 0.5 diff 0.01

create sqr_grid 100 100

// -------- rules

rule if P conc > 0.7 change P conc +0.01
rule if P conc < 0.3 change P conc -0.01

colormap select gradient
colormap slot 00 use color "black"
colormap slot 99 use color "white"

stop at 2000
```

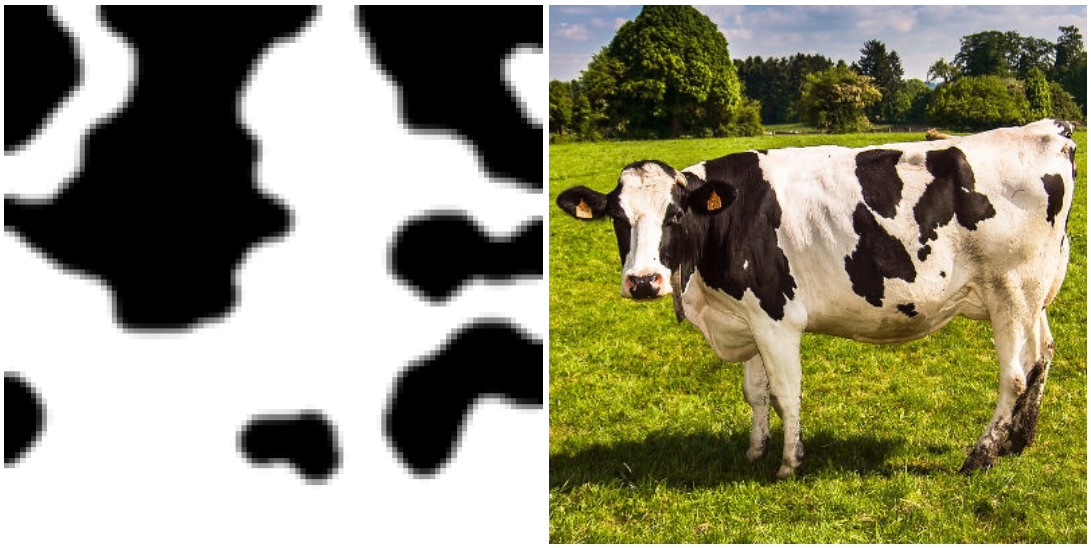

Figure 18: Reinforcement pattern: cow (*Bos taurus*). Photo by Arnaud Ligeois (Pixabay, CC0).

```
// dalmatian-like spots

define chemical P limit 1

// -------- prepattern

use seed 4

use chemical P conc 0.5 dev 0.5 diff 0.01

create sqr_grid 100 100

// -------- rules

rule if P conc > 0.8 change P conc +0.03
rule if P conc < 0.6 change P conc -0.01

colormap select gradient
colormap slot 00 use color "white"
colormap slot 99 use color "black"

stop at 2000
```

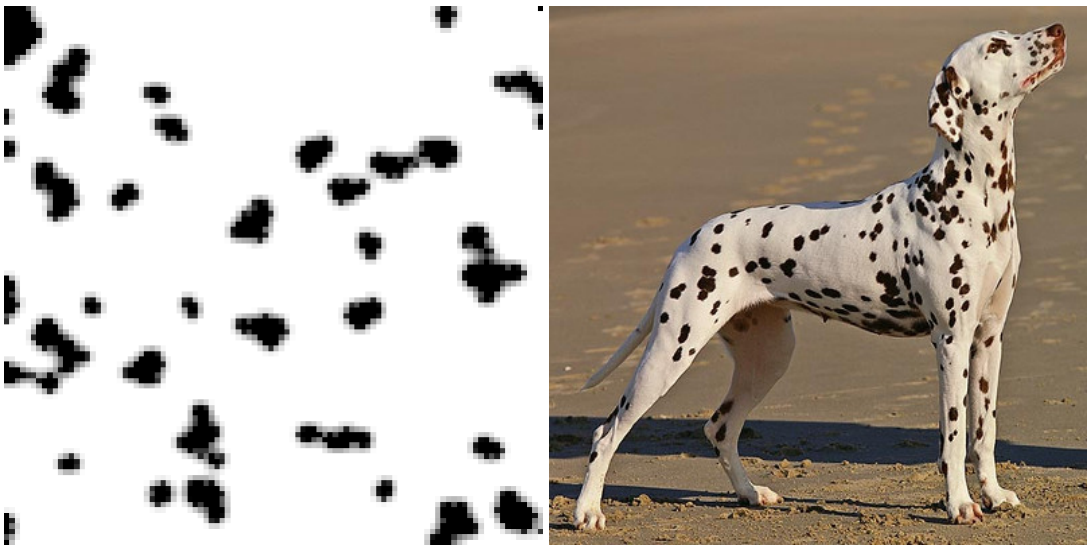

Figure 19: Reinforcement pattern: Dalmatian (*Canis lupus familiaris*). Photo by skeeze (Pixabay, CC0).

```
// african wild dog-like spots

define chemical P limit 3

// -------- prepattern

use seed 5

use chemical P conc 0.5 dev 0.5 diff 0.01

create sqr_grid 100 100

// -------- rules

rule from 100 if P conc > 0.5 change P conc +0.01
rule from 100 if P conc < 0.5 change P conc -0.04

colormap select gradient
colormap slot 00 use color "f3b88e"
colormap slot 40 use color "black"
colormap slot 60 use color "black"
colormap slot 99 use color "white"

stop at 2000
```

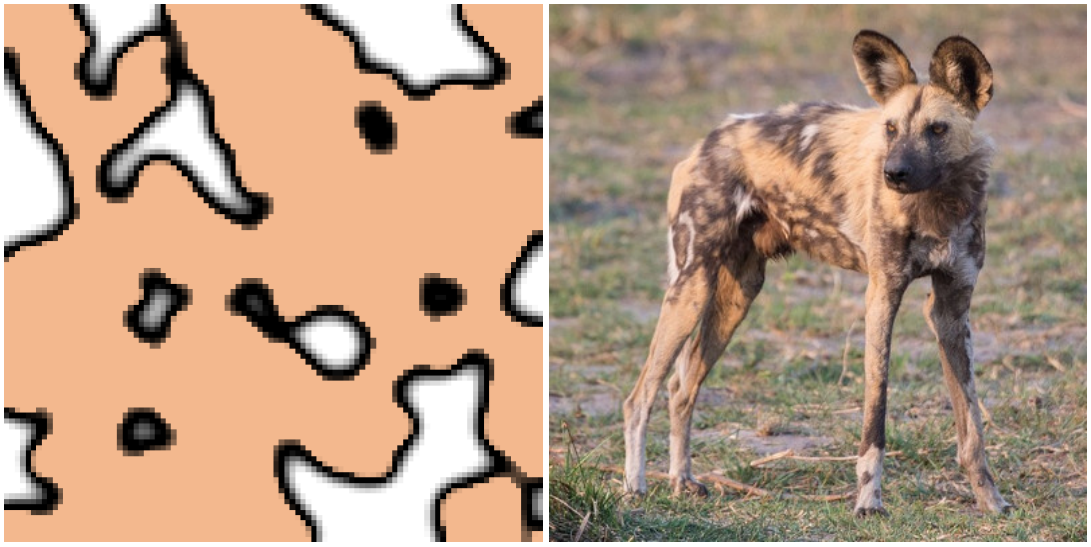

Figure 20: Reinforcement pattern: African wild dog (*Lycaon pictus*). Photo by tonyo_au (Pixabay, CC0).

3.9 Saturated reaction-diffusion

In this section, we present experiments that employed reaction-diffusion and chemical saturation.

```
// giraffe

define chemical U limit 6.3
define chemical V limit 6.3

// -------- prepattern

use seed 1

use chemical U conc 4 dev 3 diff 0.04
use chemical V conc 4 dev 3 diff 0.01

create sqr_grid 100 100 wrap

// -------- rules

rule always react U V scale 0.005 turing alpha 16 beta 12

colormap select gradient
colormap slot 60 use color "white"
colormap slot 75 use color "5b3504"

stop at 38000
```

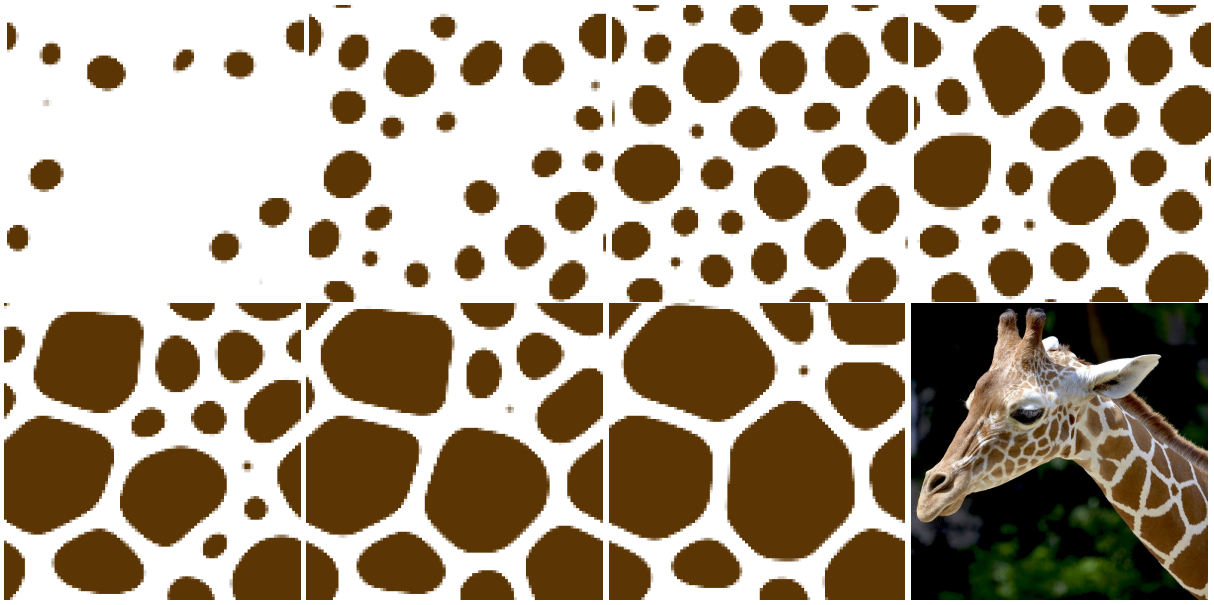

Figure 21: Reaction-diffusion where both U and V reagents achieve saturation, with emergent giraffe (*Giraffa reticulata*) pattern. Photo by Andrea Bohl (Pixabay, CC0).

We have run a few thousand simulations with different combinations of parameters for saturation limits (either for U, V or for both at the same time), diffusion rate for U, initial random seed and scale s. A particularly unusual set of parameters was later identified to match one species of nudibranch (*Hypselodoris iacula*), shown in Figure 22. One of the distinct features of this pattern are the small dots, which seldom occur but are very stable and do not fade away.

```
// nudibranch

define chemical U limit 5.2
define chemical V

// -------- prepattern

use chemical U conc 4 dev 2 diff 0.08
use chemical V conc 4 dev 2 diff 0.01
create sqr_grid 100 100 wrap

// -------- rules

rule always react U V scale 0.02 turing alpha 16 beta 12

colormap select gradient
colormap slot 20 use color "white"
colormap slot 50 use color "eaaa7a"

stop at 3000
```

Figure 22: Experiment reproducing the pigmentation of a nudibranch (*Hypselodoris iacula*) and a living individual. Photo by Steve Childs (Wikimedia Commons, CC BY 2.0).

We have also experimented with patterns with local polarization. Here follows a simple example where oriented stripes are done in a section of the pattern, whereas cells at the bottom do not have polarity. The distinction between parts is done by adjusting diffusion rates. Note that the lower pattern generates spots of different sizes. The inspiration came from the poison dart frog (*Ranitomeya amazonica*).

```
// poison dart frog

define chemical U limit 6.3
define chemical V anisotropic

define time_step 0.5

// -------- prepatterm

use polarity 90

use chemical U conc 4 dev 2 diff 0.30
use chemical V conc 4 dev 2 diff 0.02

create sqr_grid 100 100

set cells 0 to 4999 chemical U conc 4 dev 2 diff 0.011 polarity none
set cells 0 to 4999 chemical V conc 4 dev 2 diff 0.003 polarity none

// -------- rules

rule always react U V scale 0.008 turing alpha 16 beta 12

// -------- coloring

colormap select gradient
colormap slot 20 use color "ffbf00"
colormap slot 25 use color "cyan"
colormap slot 45 use color "202020"

texture size 512 512

stop at 5000
```

Figure 23: Poison dart frog (*Ranitomeya amazonica*) interpolated texture. The colors for all cells are mapped from the same concentration of the *U* reagent. Photo by Sascha Gebhardt (Flickr, CC BY-NC 2.0).

```
// moray eel

define chemical U limit 6.5
define chemical V limit 6.5
define chemical P

define time_step 0.3

// -------- prepattern

use seed 1

use chemical U conc 4 dev 2 diff 0.05
use chemical V conc 4 dev 2 diff 0.01
use chemical P conc 0       diff 0.01

create sqr_grid 200 100

set cells     0 to  4999 chemical U diff 0.40
set cells  5000 to  5399 chemical U diff 0.20

set cells 19000 to 19199 chemical U diff 0.20
set cells 19200 to 19999 chemical U diff 0.40

set cell 10240 chemical P conc 8
set cell 10850 chemical P conc 8

// -------- rules

rule always react U V scale 0.03 turing alpha 16 beta 12

rule from 2500 until 2000 always change P diff -0.01 // stops gradient
rule if P conc > 0.01 change V conc +1.0 // cells affected by gradient

colormap select gradient
colormap slot 25 use color "30292d"
colormap slot 30 use color "e8e336"
colormap slot 40 use color "white"

stop at 5000

texture size 800 400
```

Figure 24: The moray eel *Muraena melanotis*: simulated pattern and real photo. Photo by Richard Bowes (Flickr, used under permission).

```
// leopard rosettes

define chemical U limit 4.05
define chemical V

define division_limit 6
define domain packed
define time_step 0.1

// -------- prepattern

use chemical U conc 4 dev 0 diff 0.40
use chemical V conc 4 dev 0 diff 0.01

create hex_grid 50 58

set cell 205 chemical V conc 10
set cell 215 chemical V conc 10
set cell 225 chemical V conc 10
set cell 235 chemical V conc 10
set cell 245 chemical V conc 10

set cell 610 chemical V conc 10
set cell 620 chemical V conc 10
set cell 630 chemical V conc 10
set cell 640 chemical V conc 10

// other seed cells omitted for brevity:
// 1005, 1015, 1025, 1035, 1045, 1410, 1420, 1430, 1440
// 1805, 1815, 1825, 1835, 1845, 2210, 2220, 2230, 2240
// 2605, 2615, 2625, 2635, 2645

// -------- rules

rule always react U V scale 0.025 turing alpha 16 beta 12

rule from 4000 probability 0.0001 divide direction 0 dev 180

colormap select gradient
colormap slot 20 use color "251c12"
colormap slot 30 use color "98682d"
colormap slot 70 use color "fbc38d"
colormap slot 80 use color "f5e8a8"

stop at 14000

texture size 512 512
zoom level 1.503
```

Figure 25: Leopard (*Panthera pardus*) rosettes from combined uniform growth and saturated RD. The image is a interpolated texture based on the final positions and concentrations of cells. Photo by Kdsphotos (Pixabay, CC0).

References

[1] B. Desbenoit, E. Galin, and S. Akkouche. Simulating and modeling lichen growth. In *Computer Graphics Forum*, volume 23, pages 341–350. Wiley Online Library, 2004.

[2] D. R. Fowler, H. Meinhardt, and P. Prusinkiewicz. Modeling seashells. *Comp. Graphics*, 26(2):379–387, 1992.

[3] D. R. Fowler, P. Prusinkiewicz, and J. Battjes. A collision-based model of spiral phyllotaxis. In *ACM SIGGRAPH Computer Graphics*, volume 26, pages 361–368. ACM, 1992.

[4] M. Gardner. The fantastic combinations of john conway's new solitaire game life. *Scientific American*, 223(10):120–123, Oct. 1970.

[5] T. Kim, J. Sewall, A. Sud, and M. C. Lin. Fast simulation of laplacian growth. *IEEE computer graphics and applications*, 27(2), 2007.

[6] S. Miyazawa, M. Okamoto, and S. Kondo. Blending of animal colour patterns by hybridization. *Nature communications*, 1:66, 2010.

[7] K. Painter. Models for pigment pattern formation in the skin of fishes. In *Mathematical models for biological pattern formation*, pages 59–81. Springer, 2001.

[8] P. Prusinkiewicz, M. James, and R. Mech. Synthetic topiary. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 351–358. ACM, 1994.

[9] A. R. Sanderson, R. M. Kirby, C. R. Johnson, and L. Yang. Advanced reaction-diffusion models for texture synthesis. *Journal of Graphics, GPU, and Game Tools*, 11(3):47–71, 2006.

[10] A. M. Turing. The chemical basis of morphogenesis. *Phil. Trans. Roy. Soc. London*, B(237):37–72, 1952.