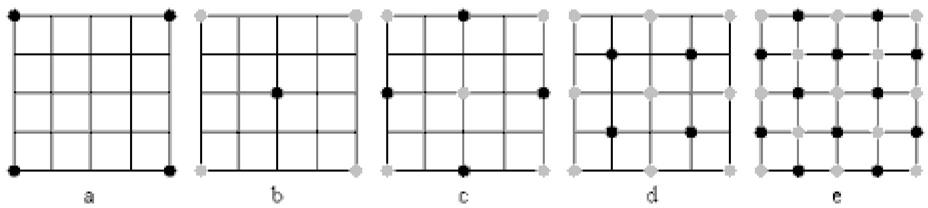
Diamond – Square Algorithm implementation with Cell DEVS

Qingzhong Liang 7354542

The real system to study is the Diamond – Square Algorithm implementation. First let us see what diamond–square algorithm is. “The diamond-square algorithm is a method for generating height maps for computer graphics” -Wikipedia.

1.1. How is it work

As a simple example, let's use a 5x5 array. In figure a four corner "seed" values are highlighted in black:



This is the starting-point for the iterative subdivision routine, which is in two steps:

- The diamond step: Taking a square of four points, generate a random value at the square midpoint, where the two diagonals meet. The midpoint value is calculated by averaging the four corner values, plus a random amount. This gives you diamonds when you have multiple squares arranged in a grid.
- The square step: Taking each diamond of four points, generate a random value at the center of the diamond. Calculate the midpoint value by averaging the corner values, plus a random amount generated in the same range as used for the diamond step. This gives you squares again.

Here is the detail of this algorithm: http://www.gameprogrammer.com/fractal.html

1.2. Implement with Cell DEVS

1. Set rules to simulate the diamond step in a certain time
2. Then simulate the square step in a certain time after step one.
3. Repeat 1 and 2, until calculate all the cells.

1.2.1.About simulate diamond step

Here is the idea:
Every cell which has a value send its value to four directions (NE, SE, NW, and SW). When a cell which has a value 0 saw that its neighborhoods (NE, SE, NW, and SW) have value, it will start to calculate its own value with certain rules.
For example:

2				4
5				3

->

2				4
	2		4	
	5		3	
5				3

->

2				4
	2		4	
		Cal		
	5		3	
5				3

1.2.2.About simulate square step

Here is the idea:
Every cell which has a value send its value to four directions (N, E, S, and W). When a cell which has a value 0 saw that its neighborhoods (N, E, S, and W) have value, it will start to calculate its own value with certain rules.
For example:

2				4
		1		
5				3

->

2	2		4	4
2		1		4
	1	1	1	
5		1		3
5	5		3	3

->

2	2	Cal	4	4
2		1		4
Cal	1	1	1	Cal
5		1		3
5	5	Cal	3	3

2. Formal Specification

The formal specification of a Cell-DEVS model is given by

GCC = <Xlist, Ylist, I, *X*, *Y*, η, N, {t1, t2, t3}, C, B, Z, select>

This model has two level. The bottom level represent the height values in the map, and second level represent the phase of step (even is Diamond step, odd is square step).

$\mathbf{Ylist} = \{\emptyset\};$

$\mathbf{Xlist} = \{\emptyset\};$

$\mathbf{I} = < P^x, P^y >$

X = {0...20000};

Y = {0...20000};

$\eta = 3$
$\mathbf{t1} = \mathbf{t2} = 17, \mathbf{t3} = 2$
$\mathbf{N} = \{(-1,-1,0), (-1,0,0), (-1,1,0),$
$(0,-1,0), (0,0,0), (0,1,0),$
$(1,-1,0), (1,0,0), (1,1,0),$
$(0,0,1)\ \}.$
$\mathbf{C} = \{C_{ijk} | i \in [1, t1], j \in [1, t2], k \in [1,2]\}$, where C_{ijk} is a Cell − DEVS atomic model;
$\mathbf{B} = \{\text{nowrapped}\}$
$Z = \{\dots\}$
$\mathbf{Select} = \{(-1,-1,0), (-1,0,0), (-1,1,0), (0,-1,0), (0,0,0), (0,1,0), (1,-1,0), (1,0,0), (1,1,0), (0,0,1)\}$

2.1. Model in CD++

```
[top]
components : DiamondSquare

[DiamondSquare]
type : cell
dim : (17,17,2)
delay : transport
defaultDelayTime : 100
border : nowrapped
neighbors : DiamondSquare(-1,-1,0)    DiamondSquare(-1,0,0) DiamondSquare(-1,1,0)
neighbors : DiamondSquare(0,-1,0)    DiamondSquare(0,0,0)    DiamondSquare(0,1,0)
neighbors : DiamondSquare(1,-1,0)    DiamondSquare(1,0,0)    DiamondSquare(1,1,0)
neighbors : DiamondSquare(0,0,1)
initialvalue : 0
initialCellsValue : DiamondSquare.val
zone : DiamondSquare-rule { (0,0,0)..(16,16,0) }
zone : second-rule { (0,0,1)..(16,16,1) }


[second-rule]
rule : {trunc((0,0,0))+0.1} 200 {fractional((0,0,0)) = 0}
rule : {trunc((0,0,0))+ 1 } 100 {fractional((0,0,0)) = 0.1}

rule : {0} 1000 {t}

[DiamondSquare-rule]
…
```

2.1.1.Second-level

The second level is use for changing phase step and clean useless value.
Let us look at the rule:

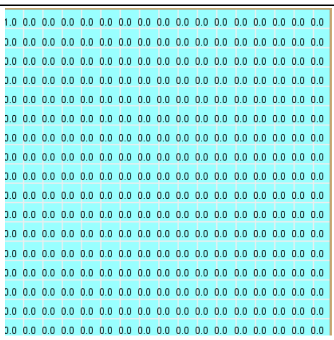
```
28.  rule : {trunc((0,0,0))+0.1} 200 {fractional((0,0,0)) = 0}
29.  rule : {trunc((0,0,0))+ 1 } 100 {fractional((0,0,0)) = 0.1}
```

(1) At the beginning, the value of the second level are 0, and after 200 millisecond, they will become 0.1 which mean that time for cleaning useless value. The clean rule as follow:

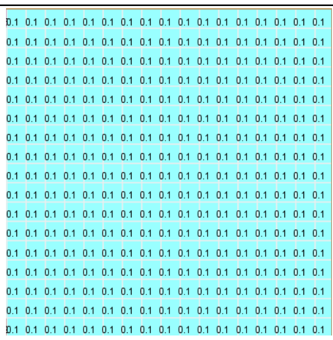
```
% all clear
rule : {0} 0 {fractional((0,0,0)) > 0 and (fractional((0,0,1)) > 0)}
```

(2) Then, when they became 0.1, they will satisfy rule 1, which is the fractional part equal 0. They will become 1 (0 + 1) after 200 millisecond. This means that the square step begins (even is Diamond step, odd is square step).
(3) And repeat to (1).

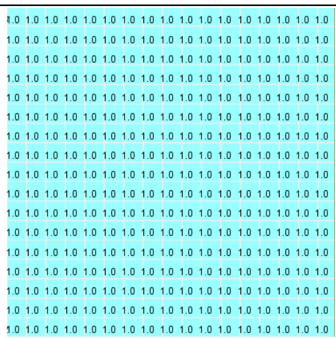
Example:



t = 0



t = 200



t = 300

2.1.2.Bottom level

2.1.2.1. **Value Definition**

(1) The fractional part of the value at bottom level means the transmitting direction.

0.1 - N

0.2 - E

0.3 - S

0.4 - W

0.5 - NE

0.6 - SE

0.7 - SW

0.8 – NW

(2) The integer part is the height value.

(3) The value without fractional part is the real height value. The value with fractional part is the transmitting value.

2.1.2.2. **Transmitting method**

Take transmitting a value to North for example:

(1) If a cell with value 0 saw that its south neighbor with just a integer value, then it will become the integer value plus the direction (north=0.11)

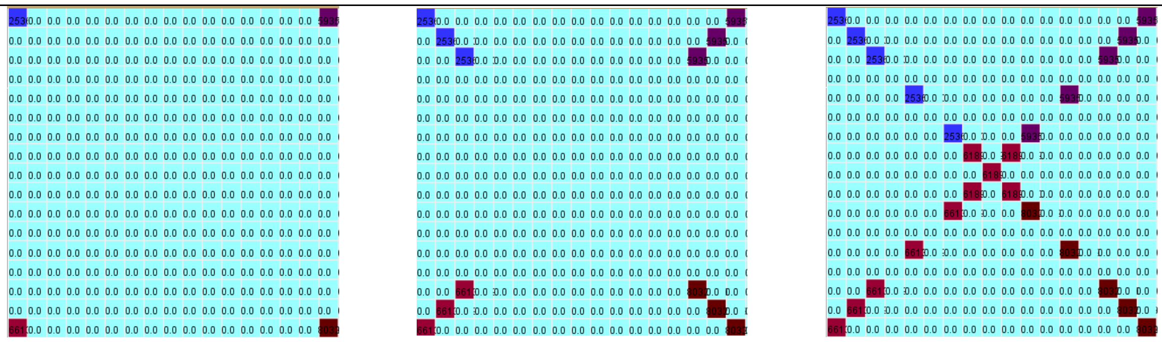
(2) If a cell with value 0 saw that its south neighbor with a real number(integer part and fractional part), then it will become the integer part plus the direction (north=0.11)

(3) If a cell with a value which fractional part is 0.11, then its fractional part will become 0.1 after certain time. This is use for showing the value moving, not a necessary action.

2.1.2.3. **Diamond step**

In this step, we should send the value through its diagonal to the center of those points.

Here is the example:



t = 0 t = 30 t = 150

Figure 2

Rules are here:

```
% send
rule : {trunc((-1,-1,0))+0.81} 10 { (0,0,0)=0 and (-1,-1,0) != 0 and (fractional((-1,-1,0)) = 0.8 or fractional((-1,-1,0)) = 0 ) and even((0,0,1))}
rule : {trunc((1, 1,0))+0.61} 10 { (0,0,0)=0 and (1,    1,0) != 0 and (fractional((1,    1,0)) = 0.6 or fractional(( 1, 1,0)) = 0 ) and even((0,0,1))}
rule : {trunc((-1, 1,0))+0.71} 10 { (0,0,0)=0 and (-1, 1,0) != 0 and (fractional((-1, 1,0)) = 0.7 or fractional((-1, 1,0)) = 0 ) and even((0,0,1))}
rule : {trunc((1, -1,0))+0.51} 10 { (0,0,0)=0 and (1, -1,0) != 0 and (fractional((1, -1,0)) = 0.5 or fractional(( 1,-1,0)) = 0 ) and even((0,0,1))}
…
% keep direction
rule : {trunc((0,0,0))+0.8} 10 {fractional((0,0,0))=0.81 }
rule : {trunc((0,0,0))+0.7} 10 {fractional((0,0,0))=0.71 }
rule : {trunc((0,0,0))+0.6} 10 {fractional((0,0,0))=0.61 }
rule : {trunc((0,0,0))+0.5} 10 {fractional((0,0,0))=0.51 }
```

In the diamond step, when a cell has value 0, which four diagonal neighbor have value, it will start to calculate its own value. (Figure 2, t = 150)

Calculation rule:

```
% cal
rule : {trunc(((-1,-1,0)+(1,1,0)+(-1,1,0)+(1,-1,0))/4 + uniform(-1,1) * 10000 * power(2,-1 *(0,0,1)))} 10 { (0,0,0)=0 and (-1,-1,0) > 0 and
(1,1,0) > 0 and (-1,1,0) > 0 and (1,-1,0) > 0 and fractional((-1,-1,0)) = 0.81}
```

rule : {trunc(((-1,-1,0)+(1,1,0)+(-1, 1,0))/3 + uniform(-1,1) * 10000 * power(2,-1 *(0,0,1)))} 10 { (0,0,0)=0 and (-1,-1,0) > 0 and (1,1,0) > 0 and (-1,1,0) > 0 and (1,-1,0) = ?}

rule : {trunc(((-1,-1,0)+(1,1,0)+(1,-1,0))/3 + uniform(-1,1) * 10000 * power(2,-1 *(0,0,1)))} 10 { (0,0,0)=0 and (-1,-1,0) > 0 and (1,1,0) > 0 and (-1,1,0) = ? and (1,-1,0) > 0}

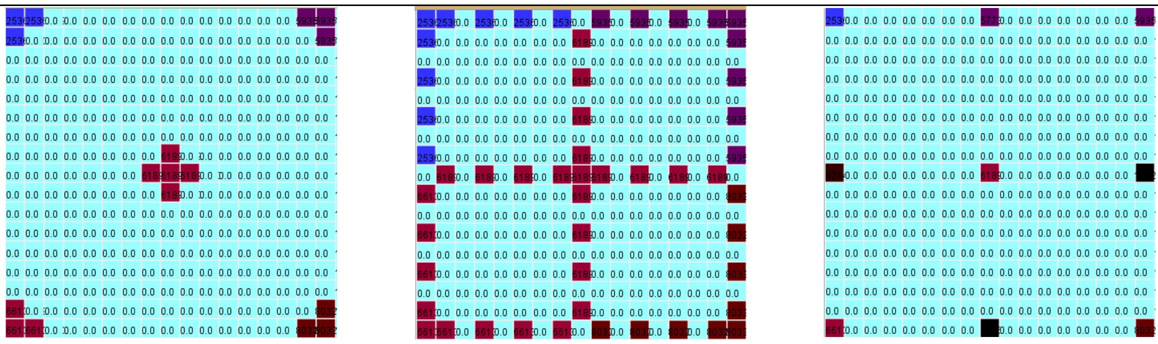
rule : {trunc(((-1,-1,0)+(-1,1,0)+(1,-1,0))/3 + uniform(-1,1) * 10000 * power(2,-1 *(0,0,1)))} 10 { (0,0,0)=0 and (-1,-1,0) > 0 and (1,1,0) = ? and (-1,1,0) > 0 and (1,-1,0) > 0}

rule : {trunc(((1, 1,0)+(-1,1,0)+(1,-1,0))/3 + uniform(-1,1) * 10000 * power(2,-1 *(0,0,1)))} 10 { (0,0,0)=0 and (-1,-1,0) = ? and (1,1,0) > 0 and (-1,1,0) > 0 and (1,-1,0) > 0}

2.1.2.4. **Square step**

In this step, we should send the value to its up, down, left and right direction, and get to the center of those points.

Here is the example:



t = 310 t = 430 t = 440

Figure 3

Rules are here:

% send

rule : {trunc((0, -1,0))+0.41} 10 { (0,0,0)=0 and (0, -1,0) != 0 and (fractional((0, -1,0)) = 0.4 or fractional((0, -1,0)) = 0) and odd((0,0,1))}

rule : {trunc((0, 1,0))+0.21} 10 { (0,0,0)=0 and (0, 1,0) != 0 and (fractional((0, 1,0)) = 0.2 or fractional((0, 1,0)) = 0) and odd((0,0,1))}

rule : {trunc((1, 0,0))+0.11} 10 { (0,0,0)=0 and (1, 0,0) != 0 and (fractional((1, 0,0)) = 0.1 or fractional((1, 0,0)) = 0) and odd((0,0,1))}

rule : {trunc((-1, 0,0))+0.31} 10 { (0,0,0)=0 and (-1, 0,0) != 0 and (fractional((-1, 0,0)) = 0.3 or fractional((-1, 0,0)) = 0) and odd((0,0,1))}

…

% keep direction

rule : {trunc((0,0,0))+0.4} 10 {fractional((0,0,0))=0.41 }

rule : {trunc((0,0,0))+0.3} 10 {fractional((0,0,0))=0.31 }

rule : {trunc((0,0,0))+0.2} 10 {fractional((0,0,0))=0.21 }

rule : {trunc((0,0,0))+0.1} 10 {fractional((0,0,0))=0.11 }

In the square step, when a cell has value 0, which four (three if it is at border) direction neighbor have value, it will start to calculate its own value. (Figure 2, t = 440)

Calculation rules:

rule : {trunc(((0,-1,0)+(0,1,0)+(-1,0,0)+ (1,0,0))/4 + uniform(-1,1) * 10000 * power(2,-1 *(0,0,1)))} 10 { (0,0,0)=0 and (0,-1,0) > 0 and (0,1,0) > 0 and (-1,0,0) > 0 and (1,0,0) > 0 and fractional((-1,0,0)) = 0.31} rule : {trunc(((0,-1,0)+(0,1,0)+(-1,0,0))/3 + uniform(-1,1) * 10000 * power(2,-1 *(0,0,1)))} 10 { (0,0,0)=0 and (0,-1, 0) > 0 and (0,1,0) > 0 and (-1,0,0) > 0 and (1,0,0) = ? }

rule : {trunc(((0,-1,0)+(0,1,0)+(1,0,0))/3 + uniform(-1,1) * 10000 * power(2,-1 *(0,0,1)))} 10 { (0,0,0)=0 and (0,-1, 0) > 0 and (0,1,0) > 0 and (-1,0,0) = ? and (1,0,0) > 0 }

rule : {trunc(((0,-1,0)+(-1,0,0)+(1,0,0))/3 + uniform(-1,1) * 10000 * power(2,-1 *(0,0,1)))} 10 { (0,0,0)=0 and (0,-1, 0) > 0 and (0,1,0) = ? and (-1,0,0) > 0 and (1,0,0) > 0 }

rule : {trunc(((0, 1,0)+(-1,0,0)+(1,0,0))/3 + uniform(-1,1) * 10000 * power(2,-1 *(0,0,1)))} 10 { (0,0,0)=0 and (0,-1, 0) = ? and (0,1,0) > 0 and (-1,0,0) > 0 and (1,0,0) > 0 }

2.1.2.5. **Termination calculation**

This situation means that there are no place to transmit value, also means that it can be calculation without transmit value.

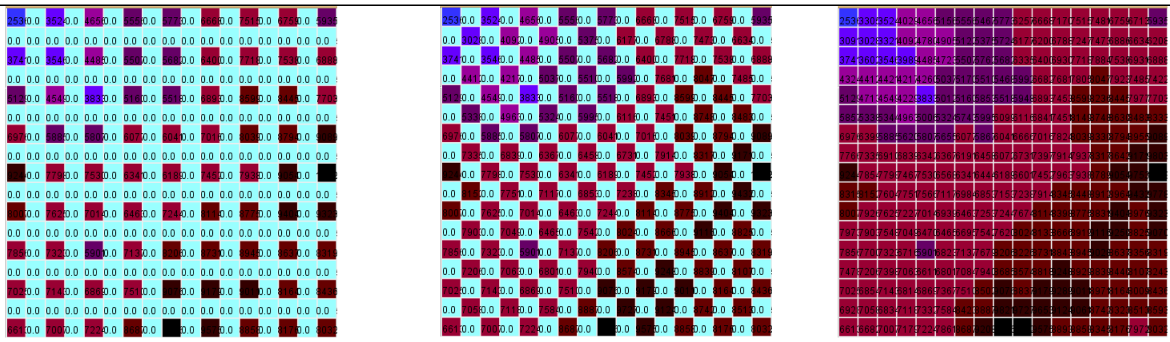
Rules:

%last cal

rule : {trunc(((-1,-1,0)+(1,1,0)+(-1,1,0)+(1,-1,0))/4 + uniform(-1,1) * 10000 * power(2,-1 *(0,0,1)))} 10 { (0,0,0)=0 and (-1,-1,0) > 0 and (1,1,0) > 0 and (-1,1,0) > 0 and (1,-1,0) > 0 and fractional((-1,-1,0)) = 0 and fractional((1, 1,0)) = 0 and fractional((-1, 1,0)) = 0 and fractional((1,-1,0)) = 0}

rule : {trunc(((0,-1,0)+(0,1,0)+(-1,0,0)+ (1,0,0))/4 + uniform(-1,1) * 10000 * power(2,-1 *(0,0,1)))} 10 { (0,0,0)=0 and (0,-1,0) > 0 and (0,1,0) > 0 and (-1,0,0) > 0 and (1,0,0) > 0 and fractional((0,-1,0)) = 0 and fractional((0, 1,0)) = 0 and fractional((-1, 0,0)) = 0 and fractional((1, 0,0)) = 0 }

Example:



t=1520 t=1540 t=1560

Figure 4

At t=1520, in the diamond step, it will be calculated immediately, because there are no place to transmit one value.
At t=1540, in the square step, it will also be calculated immediately.
At t=1560, that is the final result.

2.1.3. **Problems About this implantation method**

First, I cannot know when the diamond step or square step is finished. Therefore I just set one step running in a certain time. This may cause lots of resources are wasted. Also, it will create a big log file. The log file will be 100MB approximately if the size become 33 * 33.

3. Result

Color definition:

From	To	Color
0	1,000	
1,000	2,000	
2,000	3,000	
3,000	4,000	
4,000	5,000	
5,000	6,000	
6,000	7,000	
7,000	8,000	
8,000	9,000	
9,000	10,000	
10,000	11,000	

Test 1:

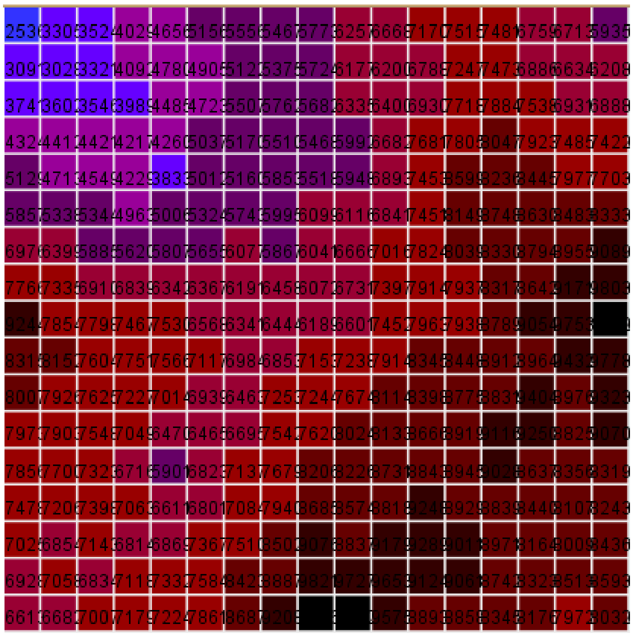
With initial value:

(0,0 ,0) = 2536.0

(0,16 ,0) = 5935.0

(16,0 ,0) = 6613.0

(16,16,0) = 8032.0



Test 2:

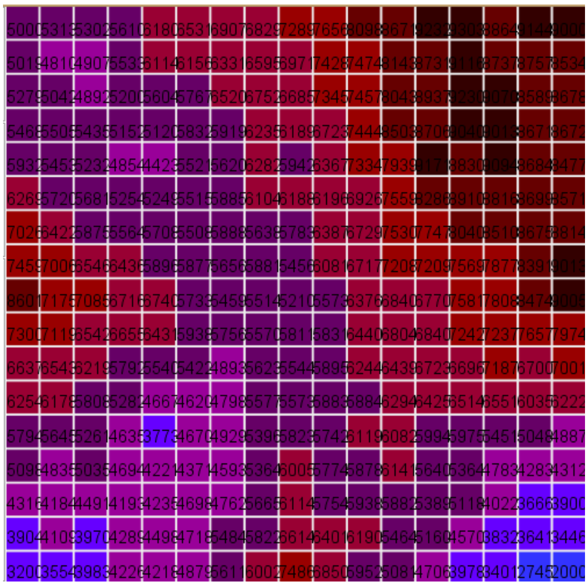
With initial value:

(0,0 ,0) = 5000.00

(0,16 ,0) = 3200.00

(16,0 ,0) = 9000.00

(16,16,0) = 2000.00



Test 3:

With initial value:

(0,0   ,0) = 1000.00

(0,16 ,0) = 3200.00

(16,0 ,0) = 9000.00

(16,16,0) = 5400.00