Building Cell-DEVS Model using Triangular and Hexagonal Meshes

Ritvik Rajnikant Pandey
Department of Systems and Computer Engineering
Carleton University
1125 Colonel By Drive
Ottawa, ON, Canada
ritvikrajnikantpande@cmail.carleton.ca

ABSTRACT

The DEVS formalism has been utilized for displaying and recreating strategy for various normal and simulated frameworks. Cell-DEVS is an expansion of DEVS that takes into consideration executing cellular automata models with the favourable position of assessing the cells asynchronously with different timing delays. Cell-DEVS is a worldview of determination to depict cell models in view of a rectangular geometry. In this paper, we present a variation of it, utilizing a hexagonal geometry which provides higher isotropy and triangular geometry which uses limited neighbors to represent diverse topology, and demonstrate their conceivable applications using the CD++ toolkit.

1. INTRODUCTION

Various research endeavours have exhibited answers for demonstrating and recreation of environmental frameworks utilizing cell models. A cell space composes the structure of the model of a given physical framework by partitioning the area of impact into geometrically distributed cells [1]. This approach is helpful, as most environmental systems are heterogeneous, and they should think about various factors, while determining the behaviour of the system in space and time. Most popular approach to deal with such systems is cellular automata which has delivered behaviours of many complex physical systems [2]. This is where Cell-DEVS comes into picture, it was defined as an extension to cellular automata.

Cell-DEVS is a worldview of detail to portray cell models in light of a rectangular geometry. The Cell-DEVS formalism permits characterizing a cell space as a grid of cells holding a figuring assembly accountable for refreshing the cell state using a set of local rules [4]. The cell states are updated by use of the present cell state and the neighborhood details using the different timing delays. There are numerous physical issues, which by their inclination, have attributes of expansion or diffusion. For these issues its portrayal in a rectangular space, does not prove to be efficient. For example, if we consider a neighborhood with 4 neighbors we cannot get the precise results corresponding to the actual phenomenon as it would only consider four directions up, down, right and left for the implementation [3]. Something similar is experienced considering a neighborhood with 8 neighbors where all the 8 squares are geometrically different. Issues like this motivated an extension to the formalism in order to represent hexagonal and triangular topologies. The hexagonal topology provides equal distances from all its neighbors hence portraying isotropic nature while triangular topologies are reserved for cases wherein limited number of neighbors are required to observe a certain phenomenon. Here, we show how the CD++ toolkit [5] helps translate hexagonal/triangular rules into compatible square rules and how we can apply these rules to certain physical systems.

2. BACKGROUND

The Discrete Events Systems specifications formalism [3] permits reuse of models by various leveled development of the models. In DEVS the essential models are called atomic models which are consolidated to form coupled models.
DEVS atomic model is described as:

$M = < X, S, Y, d_{int}, d_{ext}, l, D >$

Here, X is the input events set, S is the state set, and Y is the output events set, d_{int} is internal transitions function, dext external transitions function, l shows the outputs, and D shows the elapsed time [1]. The atomic model is most essential model in DEVS which can be outlined as shown below [3]:

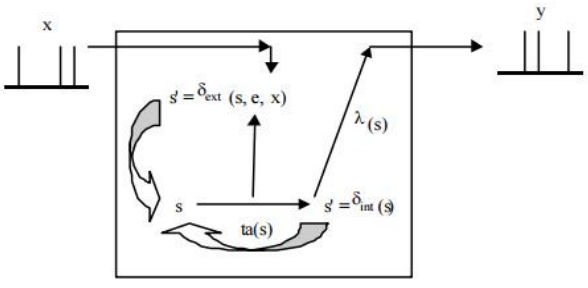

Figure 1. DEVS atomic model [3]

A DEVS atomic model [3] comprises of input port(x) and output ports(y) to associate with various models. Each model

has a state (s) which is related with a time advance (ta) function. Time advance (ta) work allocates the length of a phase. After (ta) is expended an internal transition is activated. An output function is actuated which gives the output from the output ports. After this and internal transition function (d_{int}) is terminated, which does a nearby state change. The inputs that are gotten from different models i.e. the outside events are taken care of by the external transition function(d_{ext}). Several atomic models consolidate to build a coupled model in DEVS as shown in Figure 2.

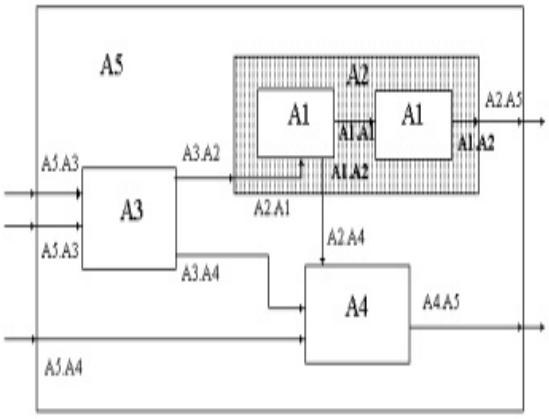

Figure 2. DEVS Coupled Model [3]

Coupled models are characterized as an arrangement of essential parts (atomic or coupled), which are interconnected through the model interfaces. The model's coupling plan characterizes the interconnectivity amongst models and the interface with the outside world.

Cell-DEVS [4] has expanded DEVS, permitting the execution of cell models with timing delays. Every cell is characterized as a DEVS atomic model, and it can be later coordinated to a coupled model speaking to the cell space. Cell-DEVS atomic model can be portrayed as follows in Figure 3.

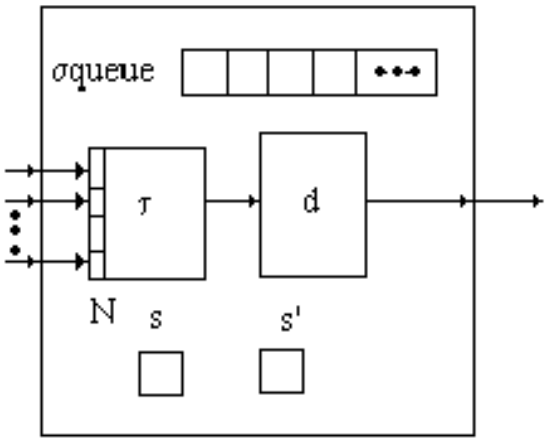

Figure 3. Cell-DEVS Atomic Model [4]

Every Cell of a Cell-DEVS model consists of a state variable and a transition function which is helpful in recognizing the cell state using the current state and its neighborhood. Every cell is described as:

TDC = < X,Y,?,N,delay,d,d_{int},d_{ext}, t , ?,D>

X and Y denotes the external inputs and outputs in the model. ? expresses the cell state, delay represents the kind of delay: transport or inertial and d is the duration of that delay, d_{int} and d_{ext} are the internal and external transitions. Every cell uses t(N) to compute future states. ? and D show the outputs and state's duration.

In Cell-DEVS [4] every cell utilizes N inputs from its neighborhood to compute the future state. The inputs N are received form the model interface, and are responsible for activating the local computing function (t). Two types of delay can be provided to each cell: transport delay and inertial delay. The state changes can be transferred to other models after the delay time is consumed [4]. A Cell-DEVS coupled model is shown in figure 4.

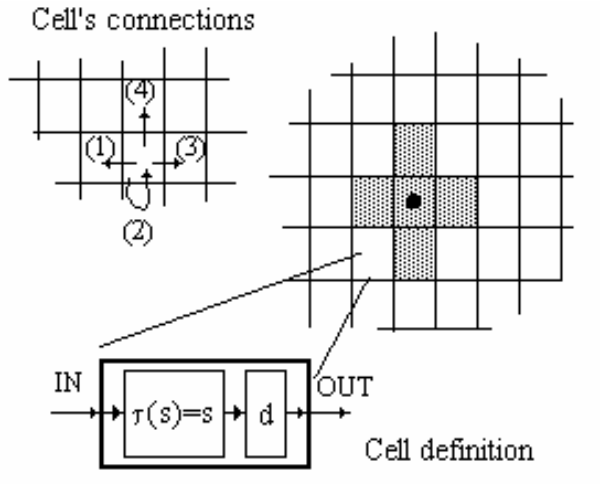

Figure 4. Cell-DEVS Coupled Model [4]

A Cell-DEVS coupled model is defined by:

GCC= < X_{list}, Y_{list}, X,Y,n,{t_1,…, t_n},N,C,B,Z >

Here Xlist and Ylist denotes are input/output coupling lists, used to define model interface. The input/output events are portrayed by X and Y. The value n shows the dimension of the cell space, {t1,…, tn} are the number of cells per dimension. N is neighbourhood set while C defines the cell space along with B which shows the border cells and Z which shows the translation function [4].

CD++ [5] is a modeling and simulation environment developed in C++ following the specifications of DEVS and Cell-DEVS models. It is used to construct DEVS and Cell-DEVS models. When constructing a Cell-DEVS model the model specification considers the size and dimension of a cell space, type of neighborhood and the borders. The local

computing function is computed by defining certain set of rules with a certain format which is shown below.

POSTCONDITION DELAY { PRECONDITION }

The implementation of the above format is as follows: When the precondition is true the state of the cell will be changed to the postcondition after certain amount of delay has elapsed. On the other hand, if the precondition is false then the next rule is evaluated until the last rule provided. If there is a case where no rule is evaluated or more than one rule are evaluated as true, the CD++ modeler generates an error. Operators such as arithmetic, comparison, time, neighborhood values, conditionals, rounding and constants are provided by CD++.

Cell-DEVS by default define the cell space in square topologies, however in certain cases square topologies are no proven to be efficient in representing the advanced cell space models. Apart from square meshes CD++ provides triangular and hexagonal meshes for representing the cell spaces. Triangular meshes permit covering zones with additional fluctuated topology, while allowing each cell to have a set number of nearby neighbors. Hexagonal meshes on the other hand have higher isotropy which proclaims the ability to represent identical conduct in each possible direction [5]. While limiting the neighbors gives us data abstraction wherein the number of messages communicated from the neighbors are lowered, but the representation and visualization of such models become difficult. There is a tool in CD++ which defines the cells behaviour based on triangular and hexagonal topologies and translates them into square compatible rules using shift mapping. LTRANS (Lattice Translator) [5] is the tool responsible for translating triangular and hexagonal rules into square rules. This tool only implements 2D models using the closest neighbors. CD++ provides a specification language [5] to describe cells behaviour based on such rules. The set of rules based on either triangular or hexagonal structure are given as input to the LTRANS tool and the square compatible rules are given as a output by LTRANS, which are then applied to the model to be simulate. The language used to model the cells behaviour in hexagonal/triangular geometry is the same except the neighbors are referenced in a different way. In CD++ a cell in a 2D space is referenced as a tuple (x,y) where x denotes the row and y denotes the columns. LTRANS only supports closest neighbors hence for hexagonal and triangular geometry nearest neighbors were defined using [n] which displays the number assigned to every neighbor. Figure 5 shows the hexagonal neighbor referencing, it is seen that a hexagonal geometry can only have 6 neighbors in total. While a triangular geometry shown in Figure 6 shows that only 3 neighbors are referenced in this topology.

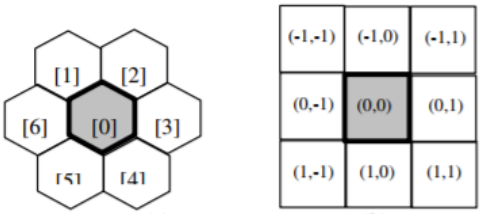

Figure 5. Hexagonal Neighbors [5]

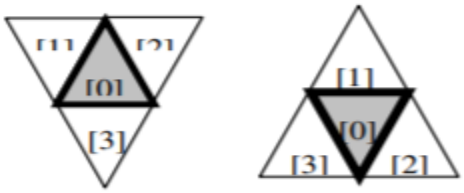

Figure 6. Triangular Neighbors [5]

Hexagonal mapping is shown in figure 7, It uses a function that shifts the alternate rows in opposite directions, at the same time preserving the boundary conditions. Triangular mapping shown in the figure 8 is done in a similar way every other cell has different orientation and each row of the triangle is mapped to a row of square reliant on the parity x+y.

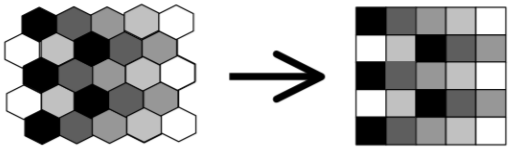

Figure 7. Hexagonal Mapping [5]

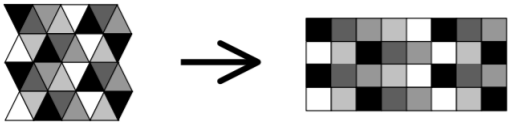

Figure 8. Triangular Mapping [5]

The neighborhood relation are transformed diversely relying on the row index x which can be oven or odd as shown in figure 9 for hexagonal geometry. While for triangular geometry figure 10 displays the even and odd combinations for neighbor relations.

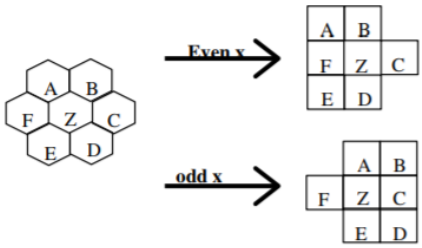

Figure 9. Hexagonal Neighborhood Relation [5]

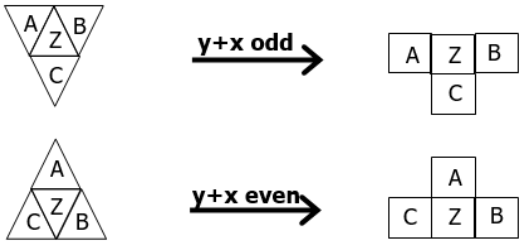

Figure 10. Triangular Neighborhood Relation [5]

Well know mathematician Martin Gardner presented the Life Game in scientific American. In this game living cells stay alive or they die. The rules of this game can be defined as follows:

1. Active cell will stay active if it two or three active neighbors.
2. An inactive cell will transit to active state if it has two or three active neighbors.
3. In any other situation the cell will die.

The rules that define cell behaviour are mentioned below in figure 11.

```
[lifegame-rule}

rule: 1 100 { {(0,0) = 1 and (truecount = 3 or truecount = 4) }
rule: 1 100 { {(0,0) = 0 and truecount = 2 }
rule: 0 100 { t }
```

Figure 11. Life Game Rules

```
[lifegame-rule}

rule: 1 100 { [0] = 1 and (truecount = 3 or truecount = 4) }
rule: 1 100 { [0] = 0 and truecount = 2 }
rule: 0 100 { t }
```

Figure 12. Hexagonal Neighbor Referencing in Life Game Rules

Figure 12 represents the hexagonal neighbor referencing in CD++. To explain the translation of rules the life game rules are written in such a way that the neighbors represent the hexagonal cells (0 to 6). After this the LTRANS tool is used to translate these rules into square compatible rules for CD++ which can be shown below in Figure 13. These rules are added to the model file while keeping the rest of the structure/specifications the same.

```
[lifegame-rule}

rule: 1 100 { ( ( (0,0) = 1 ) and ( ( if((truecount
- (if((-1,1) = 1,1,0)) -
(if((1,1) = 1,1,0))) < 0 , 0 , (truecount - (if((-1,1)
= 1,1,0)) - (if((1,1)
= 1,1,0)))) = 3 ) or ( if((truecount - (if((-1,1) = 1,1,0))
- (if((1,1) =
1,1,0))) < 0 , 0 , (truecount - (if((-1,1) = 1,1,0)) -
(if((1,1) = 1,1,0))))
= 4 ) ) ) and even(cellpos(1)) }
rule: 1 100 { ( ( (0,0) = 1 ) and ( ( if((truecount -
(if((-1,-1) = 1,1,0))
- (if((1,-1) = 1,1,0))) < 0,0,(truecount - (if((-1,-1) =
1,1,0)) - (if((1,-
1) = 1,1,0)))) = 3 ) or ( if((truecount - (if((-1,-1)
= 1,1,0)) - (if((1,-1)= 1,1,0))) < 0,0,(truecount -
(if((-1,-1) = 1,1,0)) - (
if((1,-1) = 1,1,0))))
= 4 ) ) ) and odd(cellpos(1)) }
rule: 1 100 { ( ( (0,0) = 0 ) and ( if((truecount -
(if((-1,1) = 1,1,0)) -
(if((1,1) = 1,1,0))) < 0 , 0 , (truecount - (if((-1,1)
= 1,1,0)) - (if((1,1)
= 1,1,0)))) = 2 ) ) and even(cellpos(1)) }
rule: 1 100 { ( ( (0,0) = 0 ) and ( if((truecount -
(if((-1,-1) = 1,1,0)) -
(if((1,-1) = 1,1,0))) < 0,0,(truecount - (if((-1,-1)
= 1,1,0)) - (if((1,-1)
= 1,1,0)))) = 2 ) ) and odd(cellpos(1)) }
rule: 0 100 { t and even(cellpos(1)) }
rule: 0 100 { t and odd(cellpos(1)) }
```

Figure 13. Translated Rules for Life Game

3. DEFINING DIFFERENT PHYSICAL SYSTEMS IN HEXAGONAL/TRIANGULAR TOPOLOGY

EXCITABLE MEDIA:

An excitable medium is a nonlinear dynamical framework which has the ability to engender a wave of some specification and which cannot allow passage to another wave until the point that a specific measure of time has passed [6]. Heart tissue, magnetic field and forest fire can be considered as an excitable medium. For every excitable medium a cell can take up one of the three states: resting, excited or recovering. For a forest fire the states can be unburnt, burnt or burning. We initialize with the Cell-DEVS coupled model and its parameters which can be seen below. The excitable-rules section corresponds to the local computing function.

```
[top]
components : excitable

[excitable]
type : cell
width : 11
height : 11
delay : transport
defaultDelayTime : 100
border : wrapped
neighbors : excitable(-1,0)
neighbors : excitable(0,-1) excitable(0,0) excitable(0,1)
neighbors : excitable(1,0)
initialvalue : 0
initialrowvalue : 5 00000200000
localtransition : excitable-rule

[excitable-rule]
rule : 0 100 {(0,0)=0 and statecount(2)=0 }
rule : 2 100 {(0,0)=0 and statecount(2)>0 }
rule : 1 100 { (0,0) = 2 }
rule : 0 100 { (0,0) = 1 }
rule : { (0,0) } 100 { t }
```

Figure 14. Excitable Media Coupled Model

The first rule in the excitable rule section portrays that the cell value should be 0 when there are cell neighbors with value 0, in other words if there are no excited cells nearby then the cell will remain in the resting state [6] . The second rule says that if the cell has value 0 and there are neighbors with value 2 (excited) then the cell will become excited (take value 2). The third and the fourth rule shows that the cell will remain in a particular state [6]. The default rule will be every cell will stay in its present state.

We now define the Excitable Media Coupled model using hexagonal neighbor referencing. In order to implement the hexagonal mesh rules in CD++ we have to first translate the hexagonal rules into square compatible rules which can be seen below in Figure 15 and Figure 16.

```
[excitable-rule]
rule: 0 100 {[0]=0 and statecount(2)=0 }
rule: 2 100 {[0]=0 and statecount(2)>0 }
rule: 1 100 { [0] = 2 }
rule: 0 100 { [0] = 1 }
rule: { [0] } 100 { t }
```

Figure 15. Excitable Media Rules using Hexagonal Neighbor Referencing

Figure 16 represents the translated rules into square meshes which can now be added to the coupled model. The translation of rules for the triangular geometry is the same only the number of neighbors are restricted to three which doesn't change anything in the rules mentioned above. Figure 17. Shows the translated rules for the triangular geometry. The difference between the triangular and hexagonal geometry is only the neighbor referencing i.e. hexagonal grid takes into account 6 neighbors while triangular grid considers nearby 3 neighbors. The odd and even cellpos refers to Figure 9 and Figure 10.

```
[excitable-rule]
rule : 0 100 { ( ( (0,0) = 0 ) and ( if((statecount(2)
- (if((-1,1) = 2,1,0))
- (if((1,1) = 2,1,0))) < 0,0,(statecount(2) - (if((-1,1)
= 2,1,0)) - (if((1,1)
= 2,1,0)))) = 0 ) ) and even(cellpos(1)) }
rule : 0 100 { ( ( (0,0) = 0 ) and ( if((statecount(2)
- (if((-1,-1) = 2,1,0))
- (if((1,-1) = 2,1,0))) < 0,0,(statecount(2) - (if((-1,-1)
= 2,1,0)) - (if((1,-1)
= 2,1,0)))) = 0 ) ) and odd(cellpos(1)) }
rule : 2 100 { ( ( (0,0) = 0 ) and ( if((statecount(2) -
(if((-1,1) = 2,1,0))
- (if((1,1) = 2,1,0))) < 0,0,(statecount(2) - (if((-1,1) =
2,1,0)) - (if((1,1)
= 2,1,0)))) > 0 ) ) and even(cellpos(1)) }
rule : 2 100 { ( ( (0,0) = 0 ) and ( if((statecount(2) -
(if((-1,-1) = 2,1,0))
- (if((1,-1) = 2,1,0))) < 0,0,(statecount(2) - (if((-1,-1)
= 2,1,0)) - (if((1,-1)
= 2,1,0)))) > 0 ) ) and odd(cellpos(1)) }
rule : 1 100 { ( (0,0) = 2 ) and even(cellpos(1)) }
rule : 1 100 { ( (0,0) = 2 ) and odd(cellpos(1)) }
rule : 0 100 { ( (0,0) = 1 ) and even(cellpos(1)) }
rule : 0 100 { ( (0,0) = 1 ) and odd(cellpos(1)) }
rule : {(0,0)} 100 { t and even(cellpos(1)) }
```

Figure 16. Translated Rules from Hexagonal to Square

Figure 18 shows the simulation result for the excitable medium on a hexagonal mesh. The excitable media is set at the centre of the mesh and it can be seen that how it evolves over time. Figure 20 shows the simulation results when we change the position of the excitable media. Figure 22 shows what happens when two excitable media are put up on a grid at a few cells apart.

```
[top]
components : excitable

[excitable]
type : cell
width : 11
height : 11
delay : transport
defaultDelayTime : 100
border : wrapped
neighbors : excitable(-1,0)
neighbors : excitable(0,-1) excitable(0,0) excitable(0,1)
neighbors : excitable(1,0)
initialvalue : 0
initialrowvalue : 5 00000200000
localtransition : excitable-rule

[excitable-rule]
rule : 0 100 { ( ( (0,0) = 0 ) and ( if((statecount(2) - (if((-1,1) = 2,1,0))
 - (if((1,1) = 2,1,0))) < 0,0,(statecount(2) - (if((-1,1) = 2,1,0)) - (if((1,1)
 = 2,1,0)))) = 0 ) ) and even(cellpos(1)) }
rule : 0 100 { ( ( (0,0) = 0 ) and ( if((statecount(2) - (if((-1,-1) = 2,1,0))
- (if((1,-1) = 2,1,0))) < 0,0,(statecount(2) - (if((-1,-1) = 2,1,0)) - (if((1,-1)
= 2,1,0)))) = 0 ) ) and odd(cellpos(1)) }
rule : 2 100 { ( ( (0,0) = 0 ) and ( if((statecount(2) - (if((-1,1) = 2,1,0)) -
if((1,1) = 2,1,0))) < 0,0,(statecount(2) - (if((-1,1) = 2,1,0)) - (if((1,1) = 2,1,
0)))) > 0 ) ) and even(cellpos(1)) }
rule : 2 100 { ( ( (0,0) = 0 ) and ( if((statecount(2) - (if((-1,-1) = 2,1,0)) -
(if((1,-1) = 2,1,0))) < 0,0,(statecount(2) - (if((-1,-1) = 2,1,0)) - (if((1,-1) =
2,1,0)))) > 0 ) ) and odd(cellpos(1)) }
rule : 1 100 { ( (0,0) = 2 ) and even(cellpos(1)) }
rule : 1 100 { ( (0,0) = 2 ) and odd(cellpos(1)) }
rule : 0 100 { ( (0,0) = 1 ) and even(cellpos(1)) }
rule : 0 100 { ( (0,0) = 1 ) and odd(cellpos(1)) }
rule : {(0,0)} 100 { t and even(cellpos(1)) }
rule : {(0,0)} 100 { t and odd(cellpos(1)) }
```

Figure 17. Coupled Model of Excitablehex.ma

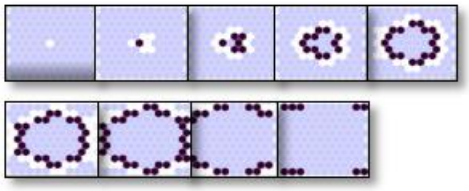

Figure 18. Simulation for Excitablehex.ma

```
[top]
components : excitable

[excitable]
type : cell
width : 11
height : 11
delay : transport
defaultDelayTime : 100
border : nowrapped
neighbors : excitable(-1,1) excitable(-1,0) excitable(-1,-1)
neighbors : excitable(0,-1) excitable(0,0) excitable(0,1)
neighbors : excitable(1,0)  excitable(1,1) excitable(1,-1)
initialvalue : 0
initialrowvalue : 1 00000200000
localtransition : excitable-rule

[excitable-rule]
rule : 0 100 { ( ( (0,0) = 0 ) and ( if((statecount(2) - (if((-1,1) =
2,1,0)) - (if((1,1) = 2,1,0))) < 0,0,(statecount(2) - (if((-1,1) = 2,1,
0)) - (if((1,1) = 2,1,0)))) = 0 ) ) and even(cellpos(1)) }
rule : 0 100 { ( ( (0,0) = 0 ) and ( if((statecount(2) - (if((-1,-1) =
2,1,0)) - (if((1,-1) = 2,1,0))) < 0,0,(statecount(2) - (if((-1,-1) = 2,
1,0)) - (if((1,-1) = 2,1,0)))) = 0 ) ) and odd(cellpos(1)) }
rule : 2 100 { ( ( (0,0) = 0 ) and ( if((statecount(2) - (if((-1,1) = 2,
1,0)) - (if((1,1) = 2,1,0))) < 0,0,(statecount(2) - (if((-1,1) = 2,1,0))
- (if((1,1) = 2,1,0)))) > 0 ) ) and even(cellpos(1)) }
rule : 2 100 { ( ( (0,0) = 0 ) and ( if((statecount(2) - (if((-1,-1) =
2,1,0)) - (if((1,-1) = 2,1,0))) < 0,0,(statecount(2) - (if((-1,-1) = 2,1
,0)) - (if((1,-1) = 2,1,0)))) > 0 ) ) and odd(cellpos(1)) }
rule : 1 100 { ( (0,0) = 2 ) and even(cellpos(1)) }
rule : 1 100 { ( (0,0) = 2 ) and odd(cellpos(1)) }
rule : 0 100 { ( (0,0) = 1 ) and even(cellpos(1)) }
rule : 0 100 { ( (0,0) = 1 ) and odd(cellpos(1)) }
rule : {(0,0)} 100 { t and even(cellpos(1)) }
rule : {(0,0)} 100 { t and odd(cellpos(1)) }
```

Figure 19. Couple Model for Excitablehex1.ma

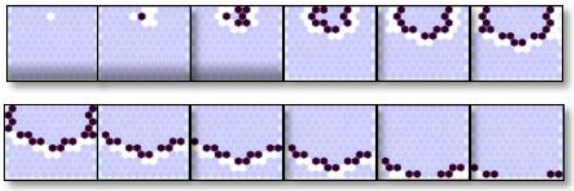

Figure 20. Simulation Result for Excitablehex1.ma

```
[top]
components : excitable

[excitable]
type : cell
width : 11
height : 11
delay : transport
defaultDelayTime : 100
border : nowrapped
neighbors : excitable(-1,1) excitable(-1,0) excitable(-1,-1)
neighbors : excitable(0,-1) excitable(0,0) excitable(0,1)
neighbors : excitable(1,0)  excitable(1,1) excitable(1,-1)
initialvalue : 0
initialrowvalue : 1 00000200000
initialrowvalue : 10 00000200000
localtransition : excitable-rule

[excitable-rule]
rule : 0 100 { ( ( (0,0) = 0 ) and ( if((statecount(2) - (if
((-1,1) = 2,1,0)) - (if((1,1) = 2,1,0))) < 0,0,(statecount(2)
- (if((-1,1) = 2,1,0)) - (if((1,1) = 2,1,0)))) = 0 ) ) and even
(cellpos(1)) }
rule : 0 100 { ( ( (0,0) = 0 ) and ( if((statecount(2) - (if((-1,
-1) = 2,1,0)) - (if((1,-1) = 2,1,0))) < 0,0,(statecount(2) - (if(
(-1,-1) = 2,1,0)) - (if((1,-1) = 2,1,0)))) = 0 ) ) and odd(cellpos
(1)) }
rule : 2 100 { ( ( (0,0) = 0 ) and ( if((statecount(2) - (if((-1,1)
 = 2,1,0)) - (if((1,1) = 2,1,0))) < 0,0,(statecount(2) - (if((-1,1)
  = 2,1,0)) - (if((1,1) = 2,1,0)))) > 0 ) ) and even(cellpos(1)) }
rule : 2 100 { ( ( (0,0) = 0 ) and ( if((statecount(2) - (if((-1,-1)
 = 2,1,0)) - (if((1,-1) = 2,1,0))) < 0,0,(statecount(2) - (if((-1,-1)
  = 2,1,0)) - (if((1,-1) = 2,1,0)))) > 0 ) ) and odd(cellpos(1)) }
rule : 1 100 { ( (0,0) = 2 ) and even(cellpos(1)) }
rule : 1 100 { ( (0,0) = 2 ) and odd(cellpos(1)) }
rule : 0 100 { ( (0,0) = 1 ) and even(cellpos(1)) }
rule : 0 100 { ( (0,0) = 1 ) and odd(cellpos(1)) }
rule : {(0,0)} 100 { t and even(cellpos(1)) }
rule : {(0,0)} 100 { t and odd(cellpos(1)) }
```

Figure 21. Coupled Model for Excitablehex2.ma

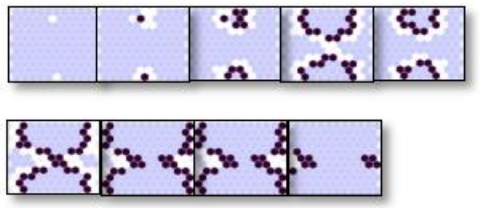

Figure 22. Simulation Result for Excitablehex2.ma

Figure 15 represents the triangular rules as well, and hence we will use the same rules for triangular geometry. Using the LTRANS tool we will translate the triangular rules into square compatible rules for CD++. The translated rules can be seen in Figure 23.

```
[excitable-rule]
rule : 0 100 { ( ( (0,0) = 0 ) and ( if((statecount(2) - (if((-1,-1) = 2,1,0))
 - (if((-1,0) = 2,1,0)) - (if((-1,1) = 2,1,0)) - (if((1,-1) = 2,1,0))
 - (if((1,1) = 2,1,0))) < 0,0,(statecount(2) - (if((-1,-1) = 2,1,0))
 - (if((-1,0) = 2,1,0)) - (if((-1,1) = 2,1,0)) - (if((1,-1) = 2,1,0))
 - (if((1,1) = 2,1,0)))) = 0 ) ) and even(cellpos(0) + cellpos(1)) }
rule : 0 100 { ( ( (0,0) = 0 ) and ( if((statecount(2) - (if((-1,-1)
= 2,1,0)) - (if((-1,1) = 2,1,0)) - (if((1,-1) = 2,1,0)) - (if((1,0)
= 2,1,0)) - (if((1,1) = 2,1,0))) < 0,0,(statecount(2) - (if((-1,-1)
= 2,1,0)) - (if((-1,1) = 2,1,0)) - (if((1,-1) = 2,1,0)) - (if((1,0)
= 2,1,0)) - (if((1,1) = 2,1,0)))) = 0 ) ) and odd(cellpos(0) + cellpos(1)) }
rule : 2 100 { ( ( (0,0) = 0 ) and ( if((statecount(2) - (if((-1,-1) = 2,1,0))
- (if((-1,0) = 2,1,0)) - (if((-1,1) = 2,1,0)) - (if((1,-1) = 2,1,0)) - (if((1,1
= 2,1,0))) < 0,0,(statecount(2) - (if((-1,-1) = 2,1,0)) - (if((-1,0) = 2,1,0))
- (if((-1,1) = 2,1,0)) - (if((1,-1) = 2,1,0)) - (if((1,1) = 2,1,0)))) > 0 ) )
and even(cellpos(0) + cellpos(1)) }
rule : 2 100 { ( ( (0,0) = 0 ) and ( if((statecount(2) - (if((-1,-1) = 2,1,0))
- (if((-1,1) = 2,1,0)) - (if((1,-1) = 2,1,0)) - (if((1,0) = 2,1,0)) - (if((1,1)
= 2,1,0))) < 0,0,(statecount(2) - (if((-1,-1) = 2,1,0)) - (if((-1,1) = 2,1,0))
- (if((1,-1) = 2,1,0)) - (if((1,0) = 2,1,0)) - (if((1,1) = 2,1,0)))) > 0 ) )
and odd(cellpos(0) + cellpos(1)) }
rule : 1 100 { ( (0,0) = 2 ) and even(cellpos(0) + cellpos(1)) }
rule : 1 100 { ( (0,0) = 2 ) and odd(cellpos(0) + cellpos(1)) }
rule : 0 100 { ( (0,0) = 1 ) and even(cellpos(0) + cellpos(1)) }
rule : 0 100 { ( (0,0) = 1 ) and odd(cellpos(0) + cellpos(1)) }
rule : {(0,0)} 100 { t and even(cellpos(0) + cellpos(1)) }
rule : {(0,0)} 100 { t and odd(cellpos(0) + cellpos(1)) }
```

Figure 23. Translated Rules from Triangular to Square

Figure 25 shows the simulation result of excitable media on a triangular geometry considering the nearest 3 neighbors. Figure 27 and Figure 29 displays the evolution of the excitable media changing the position of the excitable cell and taking two excitable cells at the same time.

```
[top]
components : excitable

[excitable]
type : cell
width : 11
height : 11
delay : transport
defaultDelayTime : 100
border : wrapped
neighbors : excitable(-1,0) excitable(-1,-1) excitable(-1,1)
neighbors : excitable(0,-1) excitable(0,0) excitable(0,1)
neighbors : excitable(1,0)  excitable(1,-1) excitable(1,1)
initialvalue : 0
initialrowvalue : 5 00000200000
localtransition : excitable-rule

[excitable-rule]
rule : 0 100 { ( ( (0,0) = 0 ) and ( if((statecount(2) - (if((-1,-1) =
2,1,0))
- (if((-1,0) = 2,1,0)) - (if((-1,1) = 2,1,0)) - (if((1,-1) = 2,1,0)) -
(if((1,1)
= 2,1,0))) < 0,0,(statecount(2) - (if((-1,-1) = 2,1,0)) - (if((-1,0) =
2,1,0))
- (if((-1,1) = 2,1,0)) - (if((1,-1) = 2,1,0)) - (if((1,1) = 2,1,0)))) = 0 )
)
and even(cellpos(0) + cellpos(1)) }
rule : 0 100 { ( ( (0,0) = 0 ) and ( if((statecount(2) - (if((-1,-1) =
2,1,0))
- (if((-1,1) = 2,1,0)) - (if((1,-1) = 2,1,0)) - (if((1,0) = 2,1,0)) -
(if((1,1)
= 2,1,0))) < 0,0,(statecount(2) - (if((-1,-1) = 2,1,0)) - (if((-1,1) =
2,1,0))
- (if((1,-1) = 2,1,0)) - (if((1,0) = 2,1,0)) - (if((1,1) = 2,1,0)))) = 0 )
)
and odd(cellpos(0) + cellpos(1)) }
rule : 2 100 { ( ( (0,0) = 0 ) and ( if((statecount(2) - (if((-1,-1) =
2,1,0))
- (if((-1,0) = 2,1,0)) - (if((-1,1) = 2,1,0)) - (if((1,-1) = 2,1,0)) -
(if((1,1)
= 2,1,0))) < 0,0,(statecount(2) - (if((-1,-1) = 2,1,0)) - (if((-1,0) =
2,1,0))
- (if((-1,1) = 2,1,0)) - (if((1,-1) = 2,1,0)) - (if((1,1) = 2,1,0)))) > 0 )
)
and even(cellpos(0) + cellpos(1)) }
rule : 2 100 { ( ( (0,0) = 0 ) and ( if((statecount(2) - (if((-1,-1) =
2,1,0))
- (if((-1,1) = 2,1,0)) - (if((1,-1) = 2,1,0)) - (if((1,0) = 2,1,0)) -
(if((1,1)
= 2,1,0))) < 0,0,(statecount(2) - (if((-1,-1) = 2,1,0)) - (if((-1,1) =
2,1,0))
- (if((1,-1) = 2,1,0)) - (if((1,0) = 2,1,0)) - (if((1,1) = 2,1,0)))) > 0 )
)
and odd(cellpos(0) + cellpos(1)) }
rule : 1 100 { ( (0,0) = 2 ) and even(cellpos(0) + cellpos(1)) }
rule : 1 100 { ( (0,0) = 2 ) and odd(cellpos(0) + cellpos(1)) }
rule : 0 100 { ( (0,0) = 1 ) and even(cellpos(0) + cellpos(1)) }
rule : 0 100 { ( (0,0) = 1 ) and odd(cellpos(0) + cellpos(1)) }
rule : {(0,0)} 100 { t and even(cellpos(0) + cellpos(1)) }
rule : {(0,0)} 100 { t and odd(cellpos(0) + cellpos(1)) }
```

Figure 24. Couple Model for Excitabletri.ma

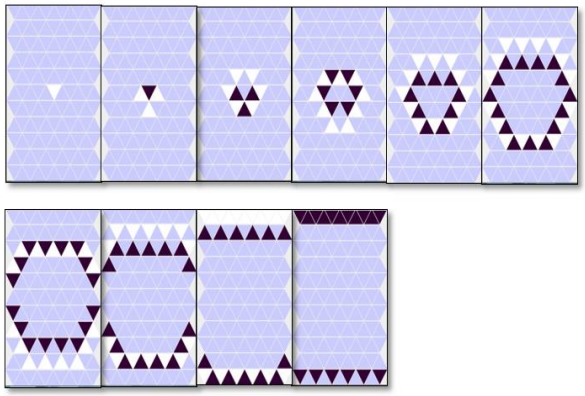

Figure 25. Simulation Result for Excitabletri.ma

```
[top]
components : excitable

[excitable]
type : cell
width : 11
height : 11
delay : transport
defaultDelayTime : 100
border : nowrapped
neighbors : excitable(-1,0) excitable(-1,-1) excitable(-1,1)
neighbors : excitable(0,-1) excitable(0,0) excitable(0,1)
neighbors : excitable(1,0)  excitable(1,-1) excitable(1,1)
initialvalue : 0
initialrowvalue : 1 00000200000
localtransition : excitable-rule

[excitable-rule]
rule : 0 100 { ( ( (0,0) = 0 ) and ( if((statecount(2) - (if((-1,-1) =
2,1,0))
 - (if((-1,0) = 2,1,0)) - (if((-1,1) = 2,1,0)) - (if((1,-1) = 2,1,0)) -
(if((1,1)
  = 2,1,0))) < 0,0,(statecount(2) - (if((-1,-1) = 2,1,0)) - (if((-1,0) =
2,1,0))
   - (if((-1,1) = 2,1,0)) - (if((1,-1) = 2,1,0)) - (if((1,1) = 2,1,0)))) =
0 ) )
    and even(cellpos(0) + cellpos(1)) }
rule : 0 100 { ( ( (0,0) = 0 ) and ( if((statecount(2) - (if((-1,-1) =
2,1,0)) -
 (if((-1,1) = 2,1,0)) - (if((1,-1) = 2,1,0)) - (if((1,0) = 2,1,0)) -
(if((1,1) =
  2,1,0))) < 0,0,(statecount(2) - (if((-1,-1) = 2,1,0)) - (if((-1,1) =
2,1,0)) -
   (if((1,-1) = 2,1,0)) - (if((1,0) = 2,1,0)) - (if((1,1) = 2,1,0)))) = 0 )
) and
    odd(cellpos(0) + cellpos(1)) }
rule : 2 100 { ( ( (0,0) = 0 ) and ( if((statecount(2) - (if((-1,-1) =
2,1,0)) -
 (if((-1,0) = 2,1,0)) - (if((-1,1) = 2,1,0)) - (if((1,-1) = 2,1,0)) -
(if((1,1)
 = 2,1,0))) < 0,0,(statecount(2) - (if((-1,-1) = 2,1,0)) - (if((-1,0) =
2,1,0))
 - (if((-1,1) = 2,1,0)) - (if((1,-1) = 2,1,0)) - (if((1,1) = 2,1,0)))) > 0
) )
 and even(cellpos(0) + cellpos(1)) }
rule : 2 100 { ( ( (0,0) = 0 ) and ( if((statecount(2) - (if((-1,-1) =
2,1,0))
- (if((-1,1) = 2,1,0)) - (if((1,-1) = 2,1,0)) - (if((1,0) = 2,1,0)) -
(if((1,1)
= 2,1,0))) < 0,0,(statecount(2) - (if((-1,-1) = 2,1,0)) - (if((-1,1) =
2,1,0))
- (if((1,-1) = 2,1,0)) - (if((1,0) = 2,1,0)) - (if((1,1) = 2,1,0)))) > 0 )
) and
 odd(cellpos(0) + cellpos(1)) }
rule : 1 100 { ( (0,0) = 2 ) and even(cellpos(0) + cellpos(1)) }
rule : 1 100 { ( (0,0) = 2 ) and odd(cellpos(0) + cellpos(1)) }
rule : 0 100 { ( (0,0) = 1 ) and even(cellpos(0) + cellpos(1)) }
rule : 0 100 { ( (0,0) = 1 ) and odd(cellpos(0) + cellpos(1)) }
rule : {(0,0)} 100 { t and even(cellpos(0) + cellpos(1)) }
rule : {(0,0)} 100 { t and odd(cellpos(0) + cellpos(1)) }
```

Figure 26. Couple Model for Excitabletri1.ma

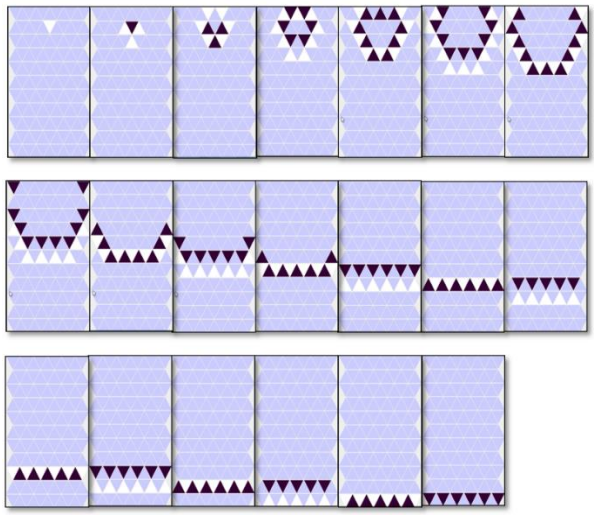

Figure 27. Simulation Result for Excitabletri1.ma

```
[top]
components : excitable

[excitable]
type : cell
width : 11
height : 11
delay : transport
defaultDelayTime : 100
border : nowrapped
neighbors : excitable(-1,0) excitable(-1,-1) excitable(-1,1)
neighbors : excitable(0,-1) excitable(0,0) excitable(0,1)
neighbors : excitable(1,0)  excitable(1,-1) excitable(1,1)
initialvalue : 0
initialrowvalue : 1 00000200000
initialrowvalue : 10 00000200000

localtransition : excitable-rule

[excitable-rule]
Rule : 0 100 { ( ( (0,0) = 0 ) and ( if((statecount(2) - (if((-1,-1)
= 2,1,0)) - (if((-1,0) = 2,1,0)) - (if((-1,1) = 2,1,0)) - (if((1,-1)
= 2,1,0)) - (if((1,1) = 2,1,0))) < 0,0,(statecount(2) - (if((-1,-1)
= 2,1,0)) - (if((-1,0) = 2,1,0)) - (if((-1,1) = 2,1,0)) - (if((1,-1)
= 2,1,0)) - (if((1,1) = 2,1,0)))) = 0 ) ) and even(cellpos(0) +
cellpos(1)) }
rule : 0 100 { ( ( (0,0) = 0 ) and ( if((statecount(2) - (if((-1,-1)
= 2,1,0)) - (if((-1,1) = 2,1,0)) - (if((1,-1) = 2,1,0)) - (if((1,0) =
2,1,0)) - (if((1,1) = 2,1,0))) < 0,0,(statecount(2) - (if((-1,-1) =
2,1,0)) - (if((-1,1) = 2,1,0)) - (if((1,-1) = 2,1,0)) - (if((1,0) =
2,1,0)) - (if((1,1) = 2,1,0)))) = 0 ) ) and odd(cellpos(0) + cellpos(1))}
rule : 2 100 { ( ( (0,0) = 0 ) and ( if((statecount(2) - (if((-1,-1) =
2,1,0)) - (if((-1,0) = 2,1,0)) - (if((-1,1) = 2,1,0)) - (if((1,-1) = 2,
1,0)) - (if((1,1) = 2,1,0))) < 0,0,(statecount(2) - (if((-1,-1) = 2,1,0
)) - (if((-1,0) = 2,1,0)) - (if((-1,1) = 2,1,0)) - (if((1,-1) = 2,1,0))
- (if((1,1) = 2,1,0)))) > 0 ) ) and even(cellpos(0) + cellpos(1)) }
rule : 2 100 { ( ( (0,0) = 0 ) and ( if((statecount(2) - (if((-1,-1) = 2
,1,0)) - (if((-1,1) = 2,1,0)) - (if((1,-1) = 2,1,0)) - (if((1,0) = 2,1,0)
) - (if((1,1) = 2,1,0))) < 0,0,(statecount(2) - (if((-1,-1) = 2,1,0)) -
(if((-1,1) = 2,1,0)) - (if((1,-1) = 2,1,0)) - (if((1,0) = 2,1,0)) -
(if((1,1) = 2,1,0)))) > 0 ) ) and odd(cellpos(0) + cellpos(1)) }
rule : 1 100 { ( (0,0) = 2 ) and even(cellpos(0) + cellpos(1)) }
rule : 1 100 { ( (0,0) = 2 ) and odd(cellpos(0) + cellpos(1)) }
rule : 0 100 { ( (0,0) = 1 ) and even(cellpos(0) + cellpos(1)) }
rule : 0 100 { ( (0,0) = 1 ) and odd(cellpos(0) + cellpos(1)) }
rule : {(0,0)} 100 { t and even(cellpos(0) + cellpos(1)) }
rule : {(0,0)} 100 { t and odd(cellpos(0) + cellpos(1)) }
```

Figure 28. Couple Model for Excitabletri2.ma

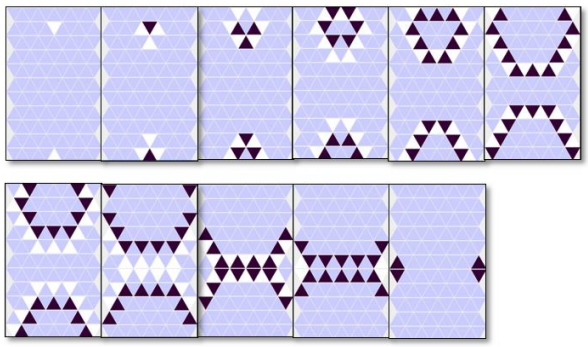

Figure 29. Simulation Result for Excitabletri2.ma

SURFACE TENSION:

Surface tension can be defined as an elastic tendency of a fluid which influences it to obtain the slightest surface area possible. The model representing surface tension is considered as a majority voting system [6] i.e. in each step the new state of the cell will depend on most of its neighbors. Two types of states are defined to represent the cells presence and absence of particles corresponding to values 1 and 0 [6]. According to the majority voting system if at least 5 neighbors are occupied then the cell state remains the same else it changes.

We now define the coupled model for surface tension as shown in the figure 23.

```
[top]
components : surfacetension

[surfacetension]
type : cell
dim : (60,60)
delay : transport
defaultDelayTime  : 100
border : wrapped
neighbors : surfacetension(-1,-1)
surfacetension(-1,0) surfacetension(-1,1)
neighbors : surfacetension(0,-1)
surfacetension(0,0)  surfacetension(0,1)
neighbors : surfacetension(1,-1)
surfacetension(1,0)  surfacetension(1,1)
initialvalue : 0
initialCellsValue : surfacetension.val
localtransition : calculatetension-rule

[calculatetension-rule]

rule : 0 100 { statecount(0) >= 5 }
rule : 1 100 { t }
```

Figure 30. Surface Tension Coupled Model

The surface tension rules are now defined in hexagonal geometry using the hexagonal neighbor referencing which can be seen in Figure 24. It uses the majority voting system to calculate the tension but as it considers hexagonal geometry

only 6 neighbors are considered hence, minimum 4 neighbors are taken which can be occupied to make the majority. Using the LTRANS tool the rules will be translated into square compatible rules for CD++ which can be shown in the Figure 25.

```
[calculatetension-rule]

rule: 0 100 { statecount(0) >= 4 }
rule: 1 100 { t }
```

Figure 31. Hexagonal Rules

```
[calculatetension-rule]
rule : 0 100 { ( if((statecount(0) - (if((-1,1) = 0,1,0))
- (if((1,1) = 0,1,0))) < 0,0,(statecount(0) -
(if((-1,1) = 0,1,0)) - (if((1,1) = 0,1,0)))) >= 4 )
and even(cellpos(1)) }
rule : 0 100 { ( if((statecount(0) - (if((-1,-1) = 0,1,0))
- (if((1,-1) = 0,1,0))) < 0,0,(statecount(0) - (if((-1,-1)
= 0,1,0)) - (if((1,-1) = 0,1,0)))) >= 4 ) and odd(cellpos(1)) }
rule : 1 100 { t and even(cellpos(1)) }
rule : 1 100 { t and odd(cellpos(1)) }
```

Figure 32. Translated Rules from Hexagonal to Square

Now the translated rules are implemented into the coupled model. The presence of particles is randomly distributed and the coupled model surfacetensionhex.ma is build using hexagonal topology. Figure 33 displays the coupled model while Figure 34 shows the simulation of the model.

```
[top]
components : surfacetension

[surfacetension]
type : cell
dim : (60,60)
delay : transport
defaultDelayTime  : 100
border : wrapped
neighbors : surfacetension(-1,-1) surfacetension(-1,0)
surfacetension(-1,1)
neighbors : surfacetension(0,-1)  surfacetension(0,0)
surfacetension(0,1)
neighbors : surfacetension(1,-1)  surfacetension(1,0)
surfacetension(1,1)
initialvalue : 0
initialCellsValue : surfacetensionhex.val
localtransition : calculatetension-rule

[calculatetension-rule]
rule : 0 100 { ( if((statecount(0) - (if((-1,1) = 0,1,0)) -
(if((1,1) = 0,1,0))) < 0,0,(statecount(0) - (if((-1,1) = 0,
1,0)) - (if((1,1) = 0,1,0)))) >= 4 ) and even(cellpos(1)) }
rule : 0 100 { ( if((statecount(0) - (if((-1,-1) = 0,1,0))
- (if((1,-1) = 0,1,0))) < 0,0,(statecount(0) - (if((-1,-1)
= 0,1,0)) - (if((1,-1) = 0,1,0)))) >= 4 ) and odd(cellpos(1)) }
rule : 1 100 { t and even(cellpos(1)) }
rule : 1 100 { t and odd(cellpos(1)) }
```

Figure 33. Coupled Model for surfacetensionhex.ma

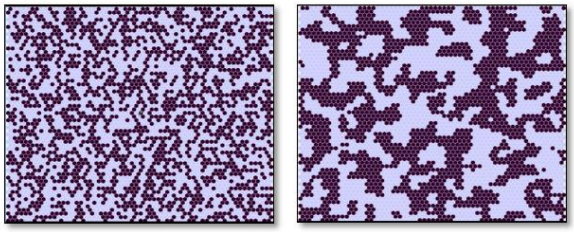

Figure 34. Simulation Result for surfacetensionhex.ma

A second case is considered with a smaller dimension of cell space and observe the results closely. Figure 35 corresponds to the coupled model for this case and Figure 36 shows the simulation result.

```
[top]
components : surfacetension

[surfacetension]
type : cell
dim : (20,20)
delay : transport
defaultDelayTime  : 100
border : wrapped
neighbors : surfacetension(-1,-1) surfacetension(-1,0)
surfacetension(-1,1)
neighbors : surfacetension(0,-1)  surfacetension(0,0)
surfacetension(0,1)
neighbors : surfacetension(1,-1)  surfacetension(1,0)
surfacetension(1,1)
initialvalue : 0
initialCellsValue : surfacetensionhex1.val
localtransition : calculatetension-rule

[calculatetension-rule]
rule : 0 100 { ( if((statecount(0) - (if((-1,1) = 0,1,0))
- (if((1,1) = 0,1,0))) < 0,0,(statecount(0) - (if((-1,1)
= 0,1,0)) - (if((1,1) = 0,1,0)))) >= 4 ) and even(cellpos(1)) }
rule : 0 100 { ( if((statecount(0) - (if((-1,-1) = 0,1,0))
- (if((1,-1) = 0,1,0))) < 0,0,(statecount(0) - (if((-1,-1)
 = 0,1,0)) - (if((1,-1) = 0,1,0)))) >= 4 ) and odd(cellpos(1)) }
rule : 1 100 { t and even(cellpos(1)) }
rule : 1 100 { t and odd(cellpos(1)) }
```

Figure 35. Coupled Model for surfacetensionhex1.ma

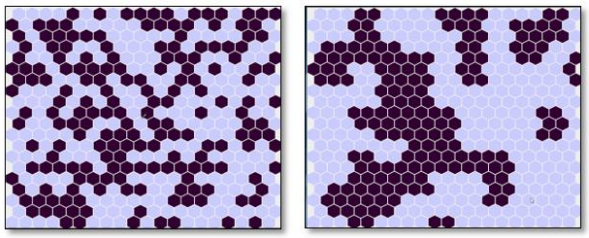

Figure 36. Simulation Result for surfacetensionhex1.ma

The triangular rules are now translated for surface tension. Triangular geometry considers nearest 3 neighbors hence, threshold of 2 neighbors are considered for calculating majority.

```
[calculatetension-rule]

rule: 0 100 { statecount(0) >= 2 }

rule: 1 100 { t }
```

Figure 37. Triangular Rules for Surface Tension

```
[calculatetension-rule]
rule : 0 100 { ( if((statecount(0) - (if((-1,-1) = 0,1,0))
- (if((-1,0) = 0,1,0)) - (if((-1,1) = 0,1,0)) - (if((1,-1)
= 0,1,0)) - (if((1,1) = 0,1,0))) < 0,0,(statecount(0) -
(if((-1,-1) = 0,1,0)) - (if((-1,0) = 0,1,0)) - (if((-1,1)
= 0,1,0)) - (if((1,-1) = 0,1,0)) - (if((1,1) = 0,1,0))))
>= 2 ) and even(cellpos(0) + cellpos(1)) }
rule : 0 100 { ( if((statecount(0) - (if((-1,-1) = 0,1,0)) -
(if((-1,1) = 0,1,0)) - (if((1,-1) = 0,1,0)) - (if((1,0) = 0,1,0))
- (if((1,1) = 0,1,0))) < 0,0,(statecount(0) - (if((-1,-1) = 0,1,0))
- (if((-1,1) = 0,1,0)) - (if((1,-1) = 0,1,0)) - (if((1,0) = 0,1,0))
- (if((1,1) = 0,1,0)))) >= 2 ) and odd(cellpos(0) + cellpos(1)) }
rule : 1 100 { t and even(cellpos(0) + cellpos(1)) }
rule : 1 100 { t and odd(cellpos(0) + cellpos(1)) }
```

Figure 38. Translated Rules from Triangular to Square

The translated rules are now included into the coupled model. The presence of particles is randomly distributed and the coupled model surfacetensionhex.ma is build using triangular topology. Figure 39 displays the coupled model while Figure 40 shows the simulation of the model.

```
[top]
components : surfacetension

[surfacetension]
type : cell
dim : (60,60)
delay : transport
defaultDelayTime  : 100
border : wrapped
neighbors : surfacetension(-1,-1) surfacetension(-1,0)
surfacetension(-1,1)
neighbors : surfacetension(0,-1)  surfacetension(0,0)
surfacetension(0,1)
neighbors : surfacetension(1,-1)  surfacetension(1,0)
surfacetension(1,1)
initialvalue : 0
initialCellsValue : surfacetensiontri.val
localtransition : calculatetension-rule
[calculatetension-rule]
rule : 0 100 { ( if((statecount(0) - (if((-1,-1) = 0,1,0))
- (if((-1,0) = 0,1,0)) - (if((-1,1) = 0,1,0)) - (if((1,-1)
= 0,1,0)) - (if((1,1) = 0,1,0))) < 0,0,(statecount(0) - (if
((-1,-1) = 0,1,0)) - (if((-1,0) = 0,1,0)) - (if((-1,1) = 0,1
,0)) - (if((1,-1) = 0,1,0)) - (if((1,1) = 0,1,0)))) >= 2 )
and even(cellpos(0) + cellpos(1)) }
rule : 0 100 { ( if((statecount(0) - (if((-1,-1) = 0,1,0))
- (if((-1,1) = 0,1,0)) - (if((1,-1) = 0,1,0)) - (if((1,0) =
0,1,0)) - (if((1,1) = 0,1,0))) < 0,0,(statecount(0) - (if(
(-1,-1) = 0,1,0)) - (if((-1,1) = 0,1,0)) - (if((1,-1) = 0,1,0
)) - (if((1,0) = 0,1,0)) - (if((1,1) = 0,1,0)))) >= 2 ) and
odd(cellpos(0) + cellpos(1)) }
rule : 1 100 { t and even(cellpos(0) + cellpos(1)) }
rule : 1 100 { t and odd(cellpos(0) + cellpos(1)) }
```

Figure 39. Coupled Model for surfactensiontri.ma

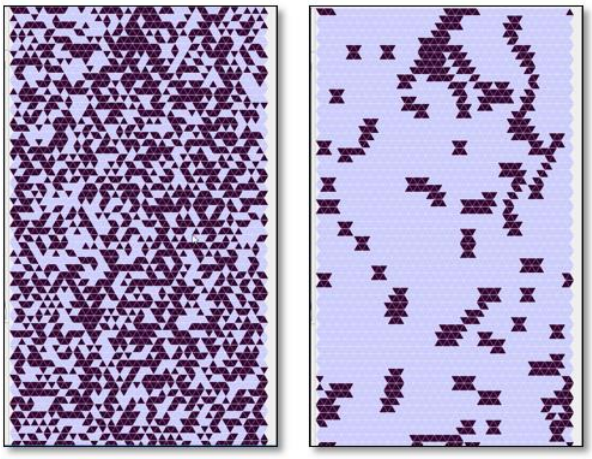

Figure 40. Simulation Result for surfactensiontri.ma

A second case with a smaller dimension of cell space is considered and observation is made closely. Figure 41 corresponds to the coupled model for this case and Figure 42 shows the simulation result.

```
[top]
components : surfacetension

[surfacetension]
type : cell
dim : (20,20)
delay : transport
defaultDelayTime  : 100
border : wrapped
neighbors : surfacetension(-1,-1) surfacetension(-1,0)
surfacetension(-1,1)
neighbors : surfacetension(0,-1)  surfacetension(0,0)
surfacetension(0,1)
neighbors : surfacetension(1,-1)  surfacetension(1,0)
surfacetension(1,1)
initialvalue : 0
initialCellsValue : surfacetensiontri1.val
localtransition : calculatetension-rule
[calculatetension-rule]
rule : 0 100 { ( if((statecount(0) - (if((-1,-1) = 0,1,0)) - (if((-1,0)
= 0,1,0)) - (if((-1,1) = 0,1,0)) - (if((1,-1) = 0,1,0)) - (if((1,1) =
0,1,0))) < 0,0,(statecount(0) - (if((-1,-1) = 0,1,0)) - (if((-1,0) = 0,
1,0)) - (if((-1,1) = 0,1,0)) - (if((1,-1) = 0,1,0)) - (if((1,1) = 0,1,0
)))) >= 2 ) and even(cellpos(0) + cellpos(1)) }
rule : 0 100 { ( if((statecount(0) - (if((-1,-1) = 0,1,0)) - (if((-1,1)
= 0,1,0)) - (if((1,-1) = 0,1,0)) - (if((1,0) = 0,1,0)) - (if((1,1) = 0,
1,0))) < 0,0,(statecount(0) - (if((-1,-1) = 0,1,0)) - (if((-1,1) = 0,1,
0)) - (if((1,-1) = 0,1,0)) - (if((1,0) = 0,1,0)) - (if((1,1) = 0,1,0))))
 >= 2 ) and odd(cellpos(0) + cellpos(1)) }
rule : 1 100 { t and even(cellpos(0) + cellpos(1)) }
rule : 1 100 { t and odd(cellpos(0) + cellpos(1)) }
```

Figure 41. Coupled Model for surfacetensiontri1.ma

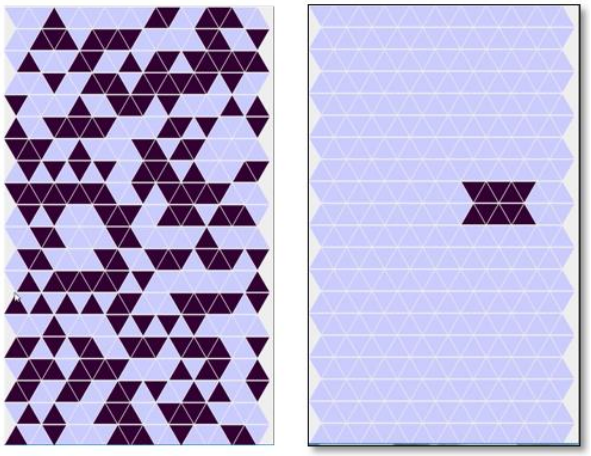

Figure 42. Simulation Result for surfacetensiontri1.ma

HEAT DIFFUSION:

Heat diffusion refers to the transfer of heat on a plane. We will discuss a 2D heat diffusion model which will show the heat diffusion on a surface [7]. The 2D heat diffusion coupled model is now defined as shown in Figure 43.

```
[top]
components : diffusion generatorHeat@Generator generatorCold@Generator
link : out@generatorHeat inputHeat@diffusion
link : out@generatorCold inputCold@diffusion

[diffusion]
type : cell
dim : (10,10)
delay : transport
defaultDelayTime  : 100
border : wrapped
neighbors : diffusion(-1,-1) diffusion(-1,0) diffusion(-1,1)
neighbors : diffusion(0,-1)  diffusion(0,0)  diffusion(0,1)
neighbors : diffusion(1,-1)  diffusion(1,0)  diffusion(1,1)
initialvalue : 25
in : inputHeat inputCold
link : inputHeat in@diffusion(6,6)
link : inputHeat in@diffusion(2,2)
link : inputCold  in@diffusion(6,5)
link : inputCold  in@diffusion(2,8)
localtransition : diffusion-rule
portInTransition : in@diffusion(6,6)  setHeat
portInTransition : in@diffusion(2,2)  setHeat
portInTransition : in@diffusion(6,5)  setCold
portInTransition : in@diffusion(2,8)  setCold

[diffusion-rule]
rule : { ( (0,0) + (0,1) + (0,-1) + (1,1) + (1,0) + (1,-1) + (-1,-1) + (-1,0)
+ (-1,1)) / 9 } 1000 { t }

[setHeat]
rule : { uniform(25,50) } 1000 { t }

[setCold]
rule : { uniform(-20,15) } 1000 { t }



[generatorHeat]
distribution : exponential
mean : 50
initial : 1
increment : 0

[generatorCold]
distribution : exponential
mean : 50
initial : 1
increment : 0
```

Figure 43. Heat Diffusion Coupled Model [7]

The cell space considered is a 10x10 cell grid. Each cell represents a temperature value [7]. There are two input entities heat and cold and we set the heat and cold temperature according to a uniform distribution with the range 25 to 50 for heat temperature and -20 to 15 for cold temperature. The diffusion rule works as follows: Each cell takes average of all the neighbors including itself updates the cell's temperature value [7]. The hexagonal rules are now defined for heat diffusion in the below mentioned figure 44.

```
[diffusion-rule]
rule: { ( [0] + [1] + [2] + [3] + [4] + [5] + [6] ) / 7 } 1000 { t }


[setHeat]
rule: { uniform(25,50) } 1000 { t }

[setCold]
rule: { uniform(-20,15) } 1000 { t }
```

Figure 44. Heat Diffusion Hexagonal Rules

As the hexagonal geometry only supports 6 neighbors we have modified the actual rules considering hexagonal neighbor referencing and 7 total neighbors. The translated rules into the square geometry can be shown below in Figure 45.

```
[diffusion-rule]
rule : {{ ( ( ( ( ( ( (0,0) + (-1,-1) ) + (-1,0) ) + (0,1) )
 + (1,0) ) + (1,-1) ) + (0,-1) ) / 7 )} 1000 { t and even
 (cellpos(1)) }
rule : {{ ( ( ( ( ( ( (0,0) + (-1,0) ) + (-1,1) ) + (0,1) )
+ (1,1) ) + (1,0) ) + (0,-1) ) / 7 )} 1000 { t and odd
(cellpos(1)) }
[setHeat]
rule : {uniform( 25 , 50 ) } 1000 { t and even(cellpos(1)) }
rule : {uniform( 25 , 50 ) } 1000 { t and odd(cellpos(1)) }
[setCold]
rule : {uniform( -20 , 15 ) } 1000 { t and even(cellpos(1)) }
rule : {uniform( -20 , 15 ) } 1000 { t and odd(cellpos(1)) }
```

Figure 45. Translated rules from hexagonal to square

A coupled model is now defined wherein we heat is generated at coordinates (6,6) and (2,2) and cold at (6,5) and (2,8) so that we can observe how the temperature changes in the cell space when the heat is diffused. Figure 46 shows the coupled model for this specification whereas Figure 47 shows the simulation result on a hexagonal grid.

```
[top]
components : diffusion generatorHeat@Generator generatorCold@Generator
link : out@generatorHeat inputHeat@diffusion
link : out@generatorCold inputCold@diffusion

[diffusion]
type : cell
dim : (10,10)
delay : transport
defaultDelayTime  : 100
border : wrapped
neighbors : diffusion(-1,-1) diffusion(-1,0)
diffusion(-1,1)
neighbors : diffusion(0,-1)  diffusion(0,0)
diffusion(0,1)
neighbors : diffusion(1,-1)  diffusion(1,0)
diffusion(1,1)
initialvalue : 25
in : inputHeat inputCold
link : inputHeat in@diffusion(6,6)
link : inputHeat in@diffusion(2,2)
link : inputCold  in@diffusion(6,5)
link : inputCold  in@diffusion(2,8)
localtransition : diffusion-rule
portInTransition : in@diffusion(6,6)  setHeat
portInTransition : in@diffusion(2,2)  setHeat
portInTransition : in@diffusion(6,5)  setCold
portInTransition : in@diffusion(2,8)  setCold

[diffusion-rule]
rule : {{ ( ( ( ( ( ( (0,0) + (-1,-1) ) + (-1,0) ) + (0,1) )
 + (1,0) ) + (1,-1) ) + (0,-1) ) / 7 )} 1000 { t and even
 (cellpos(1)) }
rule : {{ ( ( ( ( ( ( (0,0) + (-1,0) ) + (-1,1) ) + (0,1) )
+ (1,1) ) + (1,0) ) + (0,-1) ) / 7 )} 1000 { t and odd
(cellpos(1)) }
[setHeat]
rule : {uniform( 25 , 50 ) } 1000 { t and even(cellpos(1)) }
rule : {uniform( 25 , 50 ) } 1000 { t and odd(cellpos(1)) }
[setCold]
rule : {uniform( -20 , 15 ) } 1000 { t and even(cellpos(1)) }
rule : {uniform( -20 , 15 ) } 1000 { t and odd(cellpos(1)) }

[generatorHeat]
distribution : exponential
mean : 50
initial : 1
increment : 0

[generatorCold]
distribution : exponential
mean : 50
initial : 1
increment : 0
```

Figure 46. Couple Model for heatdiffuionhex.ma

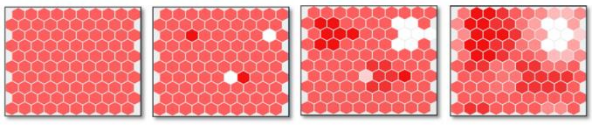

Figure 47. Simulation Result for heatdiffusionhex.ma

Considering another case where we generate heat on the top and the bottom of the cell space and observe how the heat diffusion takes place on a hexagonal grid we build the coupled model as shown in figure 48. Figure 49 shows the simulation results for the same. Note: The simulation results consist of many frames out of which only four are displayed. Better results can be seen in the simulation video.

```
[top]
components : diffusion generatorHeat@Generator generatorCold@Generator
link : out@generatorHeat inputHeat@diffusion
link : out@generatorCold inputCold@diffusion

[diffusion]
type : cell
dim : (10,10)
delay : transport
defaultDelayTime  : 100
border : wrapped
neighbors : diffusion(-1,-1) diffusion(-1,0)
 diffusion(-1,1)
neighbors : diffusion(0,-1)  diffusion(0,0)
diffusion(0,1)
neighbors : diffusion(1,-1)  diffusion(1,0)
diffusion(1,1)
initialvalue : 24
in : inputHeat inputCold
link : inputHeat in@diffusion(0,0)
link : inputHeat in@diffusion(0,1)
link : inputHeat in@diffusion(0,2)
link : inputHeat in@diffusion(0,3)
link : inputHeat in@diffusion(0,4)
link : inputHeat in@diffusion(0,5)
link : inputHeat in@diffusion(0,6)
link : inputHeat in@diffusion(0,7)
link : inputHeat in@diffusion(0,8)
link : inputHeat in@diffusion(0,9)
link : inputHeat in@diffusion(9,0)
link : inputHeat in@diffusion(9,1)
link : inputHeat in@diffusion(9,2)
link : inputHeat in@diffusion(9,3)
link : inputHeat in@diffusion(9,4)
link : inputHeat in@diffusion(9,5)
link : inputHeat in@diffusion(9,6)
link : inputHeat in@diffusion(9,7)
link : inputHeat in@diffusion(9,8)
link : inputHeat in@diffusion(9,9)
localtransition : diffusion-rule
portInTransition : in@diffusion(0,0)  setHeat
portInTransition : in@diffusion(0,1)  setHeat
portInTransition : in@diffusion(0,2)  setHeat
portInTransition : in@diffusion(0,3)  setHeat
portInTransition : in@diffusion(0,4)  setHeat
portInTransition : in@diffusion(0,5)  setHeat
portInTransition : in@diffusion(0,6)  setHeat
portInTransition : in@diffusion(0,7)  setHeat
portInTransition : in@diffusion(0,8)  setHeat
portInTransition : in@diffusion(0,9)  setHeat
portInTransition : in@diffusion(9,0)  setHeat
portInTransition : in@diffusion(9,1)  setHeat
portInTransition : in@diffusion(9,2)  setHeat
portInTransition : in@diffusion(9,3)  setHeat
portInTransition : in@diffusion(9,4)  setHeat
portInTransition : in@diffusion(9,5)  setHeat
portInTransition : in@diffusion(9,6)  setHeat
portInTransition : in@diffusion(9,7)  setHeat
portInTransition : in@diffusion(9,8)  setHeat
portInTransition : in@diffusion(9,9)  setHeat

[diffusion-rule]
rule : {( ( ( ( ( ( (0,0) + (-1,-1) ) + (-1,0) ) +
(0,1) ) + (1,0) ) + (1,-1) ) + (0,-1) ) / 7 )} 1000
{ t and even(cellpos(1)) }
rule : {( ( ( ( ( ( (0,0) + (-1,0) ) + (-1,1) ) +
(0,1) ) + (1,1) ) + (1,0) ) + (0,-1) ) / 7 )} 1000
{ t and odd(cellpos(1)) }
[setHeat]
rule : {uniform( 24 , 40 ) } 1000 { t and even(cellpos(1)) }
rule : {uniform( 24 , 40 ) } 1000 { t and odd(cellpos(1)) }
[setCold]
rule : {uniform( -10 , 15 ) } 1000 { t and even(cellpos(1)) }
rule : {uniform( -10 , 15 ) } 1000 { t and odd(cellpos(1)) }

[generatorHeat]
distribution : exponential
mean : 50
initial : 1
increment : 0

[generatorCold]
distribution : exponential
mean : 50
initial : 1
increment : 0
```

Figure 48. Coupled Model for heatdiffusionhex1.ma

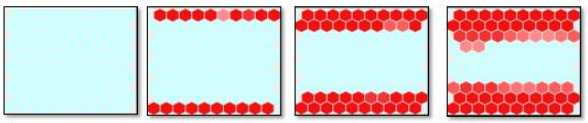

Figure 49. Simulation Results for heatdiffusionhex1.ma

Triangular rules are now defined for heat diffusion where we can only consider nearest 3 neighbors. Figure 50 shows the triangular rules for heat diffusion and Figure 51 shows the translated rules from triangular to square compatible rules.

```
[diffusion-rule]
rule: { ( [0] + [1] + [2] + [3]) / 4 } 1000 { t }

[setHeat]
rule: { uniform(25,50) } 1000 { t }

[setCold]
rule: { uniform(-20,15) } 1000 { t }
```

Figure 50. Heat Diffusion triangular rules

```
[diffusion-rule]
rule : {( ( ( ( (0,0) + (-1,0) ) + (0,1) ) + (0,-1) )
 / 4 )} 1000 { t and even(cellpos(0) + cellpos(1)) }
rule : {( ( ( ( (0,0) + (0,-1) ) + (0,1) ) + (1,0) )
/ 4 )} 1000 { t and odd(cellpos(0) + cellpos(1)) }
[setHeat]
rule : {uniform( 25 , 50 ) } 1000 { t and even(cellpos
(0) + cellpos(1)) }
rule : {uniform( 25 , 50 ) } 1000 { t and odd(cellpos
(0) + cellpos(1)) }
[setCold]
rule : {uniform( -20 , 15 ) } 1000 { t and even(cellpos
(0) + cellpos(1)) }
rule : {uniform( -20 , 15 ) } 1000 { t and odd(cellpos
(0) + cellpos(1)) }
```

Figure 51. Translated rules from triangular to square

The model is now implemented using these rules on two similar cases as done for the for hexagonal grid i.e. firstly the function set heat and set cold uniformly generates cold and heat at four cells which will vary the temperature in the cell space and secondly, the heat diffusion is observed when the heat is constantly generated at the top and bottom of the cell space. Figure 52 shows the coupled model specification for the first case and Figure 53 shows the simulation results for the same. Whereas the second case can be modeled as shown in Figure 54 and the simulation results can be observed in Figure 55. Note: The simulation results consist of many frames out of which only four are displayed. Better results can be seen in the simulation video.

```
[top]
components : diffusion generatorHeat@Generator generatorCold@Generator
link : out@generatorHeat inputHeat@diffusion
link : out@generatorCold inputCold@diffusion

[diffusion]
type : cell
dim : (10,10)
delay : transport
defaultDelayTime  : 100
border : wrapped
neighbors : diffusion(-1,-1) diffusion(-1,0) diffusion(-1,1)
neighbors : diffusion(0,-1)  diffusion(0,0)  diffusion(0,1)
neighbors : diffusion(1,-1)  diffusion(1,0)  diffusion(1,1)
initialvalue : 25
in : inputHeat inputCold
link : inputHeat in@diffusion(6,6)
link : inputHeat in@diffusion(2,2)
link : inputCold  in@diffusion(6,5)
link : inputCold  in@diffusion(2,8)
localtransition : diffusion-rule
portInTransition : in@diffusion(6,6)  setHeat
portInTransition : in@diffusion(2,2)  setHeat
portInTransition : in@diffusion(6,5)  setCold
portInTransition : in@diffusion(2,8)  setCold

[diffusion-rule]
rule : {( ( ( ( (0,0) + (-1,0) ) + (0,1) ) + (0,-1) )
 / 4 )} 1000 { t and even(cellpos(0) + cellpos(1)) }
rule : {( ( ( ( (0,0) + (0,-1) ) + (0,1) ) + (1,0) )
/ 4 )} 1000 { t and odd(cellpos(0) + cellpos(1)) }
[setHeat]
rule : {uniform( 25 , 50 ) } 1000 { t and even(cellpos
(0) + cellpos(1)) }
rule : {uniform( 25 , 50 ) } 1000 { t and odd(cellpos
(0) + cellpos(1)) }
[setCold]
rule : {uniform( -20 , 15 ) } 1000 { t and even(cellpos
(0) + cellpos(1)) }
rule : {uniform( -20 , 15 ) } 1000 { t and odd(cellpos
(0) + cellpos(1)) }

[generatorHeat]
distribution : exponential
mean : 50
initial : 1
increment : 0

[generatorCold]
distribution : exponential
mean : 50
initial : 1
increment : 0
```

Figure 52. Coupled model for heatdiffusiontri.ma

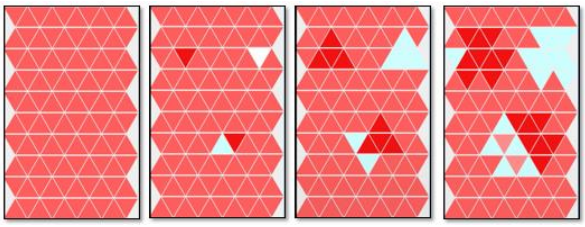

Figure 53. Simulation Result for heatdiffusiontri.ma

```
[top]
components : diffusion generatorHeat@Generator
generatorCold@Generator
link : out@generatorHeat inputHeat@diffusion
link : out@generatorCold inputCold@diffusion

[diffusion]
type : cell
dim : (10,10)
delay : transport
defaultDelayTime  : 100
border : wrapped
neighbors : diffusion(-1,-1) diffusion(-1,0)
diffusion(-1,1)
neighbors : diffusion(0,-1)  diffusion(0,0)
diffusion(0,1)
neighbors : diffusion(1,-1)  diffusion(1,0)
diffusion(1,1)
initialvalue : 24
in : inputHeat inputCold
link : inputHeat in@diffusion(0,0)
link : inputHeat in@diffusion(0,1)
link : inputHeat in@diffusion(0,2)
link : inputHeat in@diffusion(0,3)
link : inputHeat in@diffusion(0,4)
link : inputHeat in@diffusion(0,5)
link : inputHeat in@diffusion(0,6)
link : inputHeat in@diffusion(0,7)
link : inputHeat in@diffusion(0,8)
link : inputHeat in@diffusion(0,9)
link : inputHeat in@diffusion(9,0)
link : inputHeat in@diffusion(9,1)
link : inputHeat in@diffusion(9,2)
link : inputHeat in@diffusion(9,3)
link : inputHeat in@diffusion(9,4)
link : inputHeat in@diffusion(9,5)
link : inputHeat in@diffusion(9,6)
link : inputHeat in@diffusion(9,7)
link : inputHeat in@diffusion(9,8)
link : inputHeat in@diffusion(9,9)
localtransition : diffusion-rule
portInTransition : in@diffusion(0,0)  setHeat
portInTransition : in@diffusion(0,1)  setHeat
portInTransition : in@diffusion(0,2)  setHeat
portInTransition : in@diffusion(0,3)  setHeat
portInTransition : in@diffusion(0,4)  setHeat
portInTransition : in@diffusion(0,5)  setHeat
portInTransition : in@diffusion(0,6)  setHeat
portInTransition : in@diffusion(0,7)  setHeat
portInTransition : in@diffusion(0,8)  setHeat
portInTransition : in@diffusion(0,9)  setHeat
portInTransition : in@diffusion(9,0)  setHeat
portInTransition : in@diffusion(9,1)  setHeat
portInTransition : in@diffusion(9,2)  setHeat
portInTransition : in@diffusion(9,3)  setHeat
portInTransition : in@diffusion(9,4)  setHeat
portInTransition : in@diffusion(9,5)  setHeat
portInTransition : in@diffusion(9,6)  setHeat
portInTransition : in@diffusion(9,7)  setHeat
portInTransition : in@diffusion(9,8)  setHeat
portInTransition : in@diffusion(9,9)  setHeat

[diffusion-rule]
rule : {( ( ( ( (0,0) + (-1,0) ) + (0,1) ) + (0,-1)
) / 4 )} 1000 { t and even(cellpos(0) + cellpos(1)) }
rule : {( ( ( ( (0,0) + (0,-1) ) + (0,1) ) + (1,0) )
/ 4 )} 1000 { t and odd(cellpos(0) + cellpos(1)) }
[setHeat]
rule : {uniform( 24 , 40 ) } 1000 { t and even(cellpos
(1)) }
rule : {uniform( 24 , 40 ) } 1000 { t and odd(cellpos
(1)) }
[setCold]
rule : {uniform( -10 , 15 ) } 1000 { t and even(cellpos
(1)) }
rule : {uniform( -10 , 15 ) } 1000 { t and odd(cellpos
(1)) }

[generatorHeat]
distribution : exponential
mean : 50
initial : 1
increment : 0

[generatorCold]
distribution : exponential
mean : 50
initial : 1
increment : 0
```

Figure 54. Coupled model for heatdiffusiontri1.ma

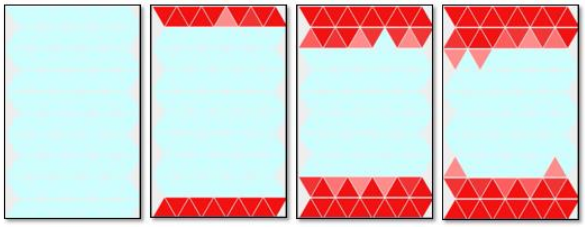

Figure 55. Simulation Result for heatdiffusiontri1.ma

GAS DIFFUSION:

The model of gas diffusion has been referenced from the sample model's webpage of sce.carleton.ca. The gas diffusion model works as follows: As the particles move freely in the lattice until they are collided by another particle and change their direction.

```
[top]
components : gasdiffusion

[gasdiffusion]
type : cell
width : 15
height : 15
delay : transport
defaultDelayTime  : 100
border : nowrapped
neighbors : gasdiffusion(-1,-1) gasdiffusion(-1,0)
gasdiffusion(-1,1)
neighbors : gasdiffusion(0,-1)  gasdiffusion(0,0)
gasdiffusion(0,1)
neighbors : gasdiffusion(1,-1)  gasdiffusion(1,0)
gasdiffusion(1,1)
initialvalue : 0
initialrow :  4 0  0  0  0  0  2  0  0  0  0
5  0  0  0  0
initialrow :  5 0  0  0  0  3  5  1  0  0  0
0  6  0  0  0
initialrow :  6 12  0  0  0  0  1  0  0  0  0
0  0  0  0  0
initialrow :  11 0  0  0  0  0  0  0  0  0  5
4  0  0  0  0
initialrow :  12 0  0  0  0  0  8  0  0  0 10
4  3  0  0  0
localtransition : gasdiffusion-rule
[gasdiffusion-rule]
rule : 10 100 { IF( (-1,0) > 11 or ((-1,0) > 3 and
(-1,0) < 8), 4, 0 ) = 4 and
                IF( Odd( (1,0) ), 1, 0 ) = 1 and
                IF( (0,1) >= 8, 8, 0 ) = 0 and
                IF( (0,-1) = 2 or (0,-1) = 3 or (0,-1)
                 = 6 or (0,-1) = 7 or (0,-1) = 10 or (
                 0,-1) = 11 or (0,-1) = 14 or (0,-1) =
                 15, 2, 0 ) = 0 }
rule :  5 100 { IF( (0,1) >= 8, 8, 0 ) = 8 and
                IF( (0,-1) = 2 or (0,-1) = 3 or (0,-1)
                = 6 or (0,-1) = 7 or (0,-1) = 10 or (0,-1)
                 = 11 or (0,-1) = 14 or (0,-1) = 15, 2, 0
                 ) = 2 and
                IF( (-1,0) > 11 or ((-1,0) > 3 and (-1,0)
                < 8), 4, 0 ) = 0 and
                IF( Odd( (1,0) ), 1, 0 ) = 0 }
rule : { IF( (-1,0) > 11 or ((-1,0) > 3 and (-1,0) < 8), 4
, 0 ) +
        IF( Odd( (1,0) ), 1, 0 ) +
        IF( (0,-1) = 2 or (0,-1) = 3 or (0,-1) = 6 or (0,-1)
         = 7 or (0,-1) = 10 or (0,-1) = 11 or (0,-1) = 14 or
          (0,-1) = 15, 2, 0 ) +
        IF( (0,1) >= 8, 8, 0 ) }  100  { t }
[comentarios]
# north = IF( (-1,0) > 11 or ((-1,0) > 3 and (-1,0) < 8), 4, 0 )
# south = IF( Odd( (1,0) ), 1, 0 )
# east  = IF( (0,1) >= 8, 8, 0 )
# west  = IF( (0,-1) = 2 or (0,-1) = 3 or (0,-1) = 6 or (0,-1) = 7
or (0,-1) = 10 or (0,-1) = 11 or (0,-1) = 14 or (0,-1) = 15, 2, 0 )
```

Figure 56. Gas Diffusion Coupled Model [8]

The above figure 56 shows the coupled model of gas diffusion. A 15x15 cell grid is considered for observing the flow of gas in the lattice. Here, 0 indicates no particles whereas 1-15 indicates the presence of a particle travelling in all four directions (north = 1, south = 4, east = 2 and west = 8) [8]

```
[gasdiffusion-rule]
rule: 10 100 { if( [1] > 11 or ([1] > 3 and [1] < 8), 4, 0 ) = 4
 and if( Odd( [2] ), 1, 0 ) = 1 and if( [6] >= 8, 8, 0 ) = 0 and
 if( [4] = 2 or [4] = 3 or [4] = 6 or [4] = 7 or [4] = 10 or [4]
 = 11 or [4] = 14 or [4] = 15, 2, 0 ) = 0 }
rule: 5 100 { if( [6] >= 8, 8, 0 ) = 8 and if( [4] = 2 or [4] = 3
or [4] = 6 or [4] = 7 or [4] = 10 or [4] = 11 or [4] = 14 or [4]
= 15, 2, 0 ) = 2 and if( [1] > 11 or ([1] > 3 and [1] < 8), 4, 0 )
= 0 and if( Odd( [2] ), 1, 0 ) = 0 }
rule: { if( [1] > 11 or ([1] > 3 and [1] < 8), 4, 0 ) + if( Odd(
[2] ), 1, 0 ) + if( [4] = 2 or [4] = 3 or [4] = 6 or [4] = 7 or [4]
= 10 or [4] = 11 or [4] = 14 or [4] = 15, 2, 0 ) + if( [6] >= 8, 8, 0
) }  100  { t }
```

Figure 57. Hexagonal Rules for Gas Diffusion

```
[gasdiffusion-rule]
rule : 10 100 { ( ( ( ( ( if( ( ( (-1,-1) > 11 ) or ( ( (-1,-1) > 3 )
 and ( (-1,-1) < 8 ) ) ) , 4 , 0 )  = 4 ) and ( if( odd((-1,0))  ,
 1 , 0 )  = 1 ) ) and ( if( ( (0,-1) >= 8 ) , 8 , 0 )  = 0 ) ) and
 ( if( ( ( ( ( ( ( ( (1,0) = 2 ) or ( (1,0) = 3 ) ) or ( (1,0) =
 6 ) ) or ( (1,0) = 7 ) ) or ( (1,0) = 10 ) ) or ( (1,0) = 11 ) )
 or ( (1,0) = 14 ) ) or ( (1,0) = 15 ) ) , 2 , 0 )  = 0 ) ) and even
 (cellpos(1)) }
rule : 10 100 { ( ( ( ( ( if( ( ( (-1,0) > 11 ) or ( ( (-1,0) > 3 )
and ( (-1,0) < 8 ) ) ) , 4 , 0 )  = 4 ) and ( if( odd((-1,1))  , 1 ,
 0 )  = 1 ) ) and ( if( ( (0,-1) >= 8 ) , 8 , 0 )  = 0 ) ) and ( if(
  ( ( ( ( ( ( ( (1,1) = 2 ) or ( (1,1) = 3 ) ) or ( (1,1) = 6 ) )
  or ( (1,1) = 7 ) ) or ( (1,1) = 10 ) ) or ( (1,1) = 11 ) ) or ( (1
  ,1) = 14 ) ) or ( (1,1) = 15 ) ) , 2 , 0 )  = 0 ) ) and odd(cellpos
  (1)) }
rule : 5 100 { ( ( ( ( if( ( (0,-1) >= 8 ) , 8 , 0 )  = 8 ) and ( if(
 ( ( ( ( ( ( ( (1,0) = 2 ) or ( (1,0) = 3 ) ) or ( (1,0) = 6 ) ) or
  ( (1,0) = 7 ) ) or ( (1,0) = 10 ) ) or ( (1,0) = 11 ) ) or ( (1,0)
  = 14 ) ) or ( (1,0) = 15 ) ) , 2 , 0 )  = 2 ) ) and ( if( ( ( ( (-1,-1
  ) > 11 ) or ( ( (-1,-1) > 3 ) and ( (-1,-1) < 8 ) ) ) , 4 , 0 )  = 0
   ) ) and ( if( odd((-1,0))  , 1 , 0 )  = 0 ) ) and even(cellpos(1)) }
rule : 5 100 { ( ( ( ( if( ( (0,-1) >= 8 ) , 8 , 0 )  = 8 ) and ( if(
( ( ( ( ( ( ( (1,1) = 2 ) or ( (1,1) = 3 ) ) or ( (1,1) = 6 ) ) or
( (1,1) = 7 ) ) or ( (1,1) = 10 ) ) or ( (1,1) = 11 ) ) or ( (1,1) =
14 ) ) or ( (1,1) = 15 ) ) , 2 , 0 )  = 2 ) ) and ( if( ( ( ( (-1,0) >
11 ) or ( ( (-1,0) > 3 ) and ( (-1,0) < 8 ) ) ) , 4 , 0 )  = 0 ) ) and
 ( if( odd((-1,1))  , 1 , 0 )  = 0 ) ) and odd(cellpos(1)) }
rule : {( ( ( if( ( ( (-1,-1) > 11 ) or ( ( (-1,-1) > 3 ) and ( (-1,-1)
 < 8 ) ) ) , 4 , 0 )  + if( odd((-1,0))  , 1 , 0 )  ) + if( ( ( ( ( (
  ( (1,0) = 2 ) or ( (1,0) = 3 ) ) or ( (1,0) = 6 ) ) or ( (1,0) = 7 )
  ) or ( (1,0) = 10 ) ) or ( (1,0) = 11 ) ) or ( (1,0) = 14 ) ) or ( (1,0)
   = 15 ) ) , 2 , 0 )  ) + if( ( ( (0,-1) >= 8 ) , 8 , 0 )  )} 100 { t and
   even(cellpos(1)) }
rule : {( ( ( if( ( ( (-1,0) > 11 ) or ( ( (-1,0) > 3 ) and ( (-1,0) < 8 )
) ) , 4 , 0 )  + if( odd((-1,1))  , 1 , 0 )  ) + if( ( ( ( ( ( ( ( (1,1)
= 2 ) or ( (1,1) = 3 ) ) or ( (1,1) = 6 ) ) or ( (1,1) = 7 ) ) or ( (1,1)
= 10 ) ) or ( (1,1) = 11 ) ) or ( (1,1) = 14 ) ) or ( (1,1) = 15 ) ) , 2 ,
 0 )  ) + if( ( (0,-1) >= 8 ) , 8 , 0 )  )} 100 { t and odd(cellpos(1)) }
```

Figure 58. Translate rules from Hexagonal to Square

The figure 57 shows the hexagonal rules for gas diffusion in a 2D cellular automata. Upon translation the square compatible rules are generated for the hexagonal topology in Figure 58. These rules are used to observe the behavior of gas diffusion on a hexagonal grid. A coupled model is built with the same specifications but only with the new translated rules which can be seen in the figure 59. On performing simulation on this model, we get the results as shown in the figure 60.

```
[top]
Compo
nents : gasdiffusion

[gasdiffusion]
type : cell
width : 15
height : 15
delay : transport
defaultDelayTime  : 100
border : nowrapped
neighbors : gasdiffusion(-1,-1) gasdiffusion(-1,0) gasdiffusion(-1,1)
neighbors : gasdiffusion(0,-1)  gasdiffusion(0,0)  gasdiffusion(0,1)
neighbors : gasdiffusion(1,-1)  gasdiffusion(1,0) gasdiffusion(1,1)
initialvalue : 0
initialrow :  4      0  0  0  0  0  2  0  0  0  0  0  0  0  0  0
initialrow :  5      0  0  0  0  3  5  1  0  0  0  0  0  0  0  0
initialrow :  6      0  0  0  0  0  1  0  0  0  0  0  0  0  0  0
initialrow :  11     0  0  0  0  0  0  0  0  0  5  4  0  0  0  0
initialrow :  12     0  0  0  0  0  0  0  0  0 10  4  3  0  0  0
localtransition : gasdiffusion-rule

[gasdiffusion-rule]
rule : 10 100 { ( ( ( ( if( ( ( (-1,-1) > 11 ) or ( ( (-1,-1) > 3 )
 and ( (-1,-1) < 8 ) ) ) , 4 , 0 )  = 4 ) and ( if( odd((-1,0))  ,
 1 , 0 )  = 1 ) ) and ( if( ( (0,-1) >= 8 ) , 8 , 0 )  = 0 ) ) and
 ( if( ( ( ( ( ( ( ( (1,0) = 2 ) or ( (1,0) = 3 ) ) or ( (1,0) =
 6 ) ) or ( (1,0) = 7 ) ) or ( (1,0) = 10 ) ) or ( (1,0) = 11 ) )
 or ( (1,0) = 14 ) ) or ( (1,0) = 15 ) ) , 2 , 0 )  = 0 ) ) and even
 (cellpos(1)) }
rule : 10 100 { ( ( ( ( if( ( ( (-1,0) > 11 ) or ( ( (-1,0) > 3 )
and ( (-1,0) < 8 ) ) ) , 4 , 0 )  = 4 ) and ( if( odd((-1,1))  , 1 ,
 0 )  = 1 ) ) and ( if( ( (0,-1) >= 8 ) , 8 , 0 )  = 0 ) ) and ( if(
  ( ( ( ( ( ( ( (1,1) = 2 ) or ( (1,1) = 3 ) ) or ( (1,1) = 6 ) )
  or ( (1,1) = 7 ) ) or ( (1,1) = 10 ) ) or ( (1,1) = 11 ) ) or ( (1
 ,1) = 14 ) ) or ( (1,1) = 15 ) ) , 2 , 0 )  = 0 ) ) and odd(cellpos
  (1)) }
rule : 5 100 { ( ( ( ( if( ( (0,-1) >= 8 ) , 8 , 0 )  = 8 ) and ( if(
 ( ( ( ( ( ( ( ( (1,0) = 2 ) or ( (1,0) = 3 ) ) or ( (1,0) = 6 ) ) or
  ( (1,0) = 7 ) ) or ( (1,0) = 10 ) ) or ( (1,0) = 11 ) ) or ( (1,0)
  = 14 ) ) or ( (1,0) = 15 ) ) , 2 , 0 )  = 2 ) ) and ( if( ( ( (-1,-1
  ) > 11 ) or ( ( (-1,-1) > 3 ) and ( (-1,-1) < 8 ) ) ) , 4 , 0 )  = 0
   ) ) and ( if( odd((-1,0))  , 1 , 0 )  = 0 ) ) and even(cellpos(1)) }
rule : 5 100 { ( ( ( ( if( ( (0,-1) >= 8 ) , 8 , 0 )  = 8 ) and ( if(
 ( ( ( ( ( ( ( ( (1,1) = 2 ) or ( (1,1) = 3 ) ) or ( (1,1) = 6 ) ) or
 ( (1,1) = 7 ) ) or ( (1,1) = 10 ) ) or ( (1,1) = 11 ) ) or ( (1,1) =
14 ) ) or ( (1,1) = 15 ) ) , 2 , 0 )  = 2 ) ) and ( if( ( ( (-1,0) >
11 ) or ( ( (-1,0) > 3 ) and ( (-1,0) < 8 ) ) ) , 4 , 0 )  = 0 ) ) and
 ( if( odd((-1,0))  , 1 , 0 )  = 0 ) ) and odd(cellpos(1)) }
rule : {{ ( ( ( if( ( ( (-1,-1) > 11 ) or ( ( (-1,-1) > 3 ) and ( (-1,-1)
 < 8 ) ) ) , 4 , 0 )  + if( odd((-1,0))  , 1 , 0 )  ) + if( ( ( ( ( ( (
  ( ( (1,0) = 2 ) or ( (1,0) = 3 ) ) or ( (1,0) = 6 ) ) or ( (1,0) = 7 )
  ) or ( (1,0) = 10 ) ) or ( (1,0) = 11 ) ) or ( (1,0) = 14 ) ) or ( (1,0)
   = 15 ) ) , 2 , 0 )  ) + if( ( (0,-1) >= 8 ) , 8 , 0 )  )} 100 { t and
   even(cellpos(1)) }
rule : {{ ( ( ( if( ( ( (-1,0) > 11 ) or ( ( (-1,0) > 3 ) and ( (-1,0) < 8 )
) ) , 4 , 0 )  + if( odd((-1,1))  , 1 , 0 )  ) + if( ( ( ( ( ( ( ( (1,1)
= 2 ) or ( (1,1) = 3 ) ) or ( (1,1) = 6 ) ) or ( (1,1) = 7 ) ) or ( (1,1)
= 10 ) ) or ( (1,1) = 11 ) ) or ( (1,1) = 14 ) ) or ( (1,1) = 15 ) ) , 2 ,
 0 )  ) + if( ( (0,-1) >= 8 ) , 8 , 0 )  )} 100 { t and odd(cellpos(1)) }
```

Figure 59. Coupled Model for gasdiffusionhex.ma

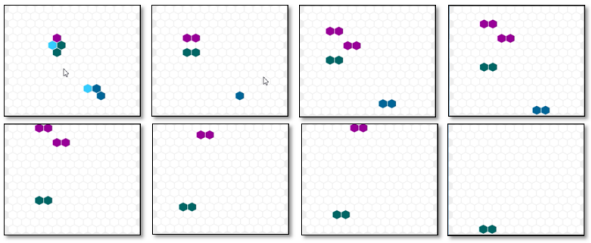

Figure 60. Simulation Result for gasdiffusionhex.ma

```
[gasdiffusion-rule]
rule: 10 100 { if( [1] > 11 or ([1] > 3 and [1] < 8), 4, 0 ) = 4 and
if( Odd( [2] ), 1, 0 ) = 1 and if( [1] >= 8, 8, 0 ) = 0 and if( [3]
= 2 or [3] = 3 or [3] = 6 or [3] = 7 or [3] = 10 or [3] = 11 or [3]
= 14 or [3] = 15, 2, 0 ) = 0 }
rule: 5 100 { if( [1] >= 8, 8, 0 ) = 8 and if( [3] = 2 or [3] = 3 or
[3] = 6 or [3] = 7 or [3] = 10 or [3] = 11 or [3] = 14 or [3] = 15, 2,
0 ) = 2 and if( [1] > 11 or ([1] > 3 and [1] < 8), 4, 0 ) = 0 and if(
Odd( [2] ), 1, 0 ) = 0 }
rule: { if( [1] > 11 or ([1] > 3 and [1] < 8), 4, 0 ) + if( Odd( [2] )
, 1, 0 ) + if( [3] = 2 or [3] = 3 or [3] = 6 or [3] = 7 or [3] = 10 or
[3] = 11 or [3] = 14 or [3] = 15, 2, 0 ) + if( [1] >= 8, 8, 0 ) }  100  { t }
```

Figure 61. Triangular Rules for gas diffusion

```
[gasdiffusion-rule]
rule : 10 100 { ( ( ( ( if( ( ( (-1,0) > 11 ) or ( ( (-1,0) > 3 )
and ( (-1,0) < 8 ) ) ) , 4 , 0 )  = 4 ) and ( if( odd((0,1))  , 1
, 0 )  = 1 ) ) and ( if( ( (-1,0) >= 8 ) , 8 , 0 )  = 0 ) ) and (
if( ( ( ( ( ( ( ( (0,-1) = 2 ) or ( (0,-1) = 3 ) ) or ( (0,-1) =
6 ) ) or ( (0,-1) = 7 ) ) or ( (0,-1) = 10 ) ) or ( (0,-1) = 11 ) )
or ( (0,-1) = 14 ) ) or ( (0,-1) = 15 ) ) , 2 , 0 )  = 0 ) ) and
even(cellpos(0) + cellpos(1)) }
rule : 10 100 { ( ( ( ( if( ( ( (0,-1) > 11 ) or ( ( (0,-1) > 3 )
and ( (0,-1) < 8 ) ) ) , 4 , 0 )  = 4 ) and ( if( odd((0,1))  , 1 ,
0 )  = 1 ) ) and ( if( ( (0,-1) >= 8 ) , 8 , 0 )  = 0 ) ) and ( if
( ( ( ( ( ( ( ( (1,0) = 2 ) or ( (1,0) = 3 ) ) or ( (1,0) = 6 ) )
or ( (1,0) = 7 ) ) or ( (1,0) = 10 ) ) or ( (1,0) = 11 ) ) or ( (1,
0) = 14 ) ) or ( (1,0) = 15 ) ) , 2 , 0 )  = 0 ) ) and odd(cellpos(
0) + cellpos(1)) }
rule : 5 100 { ( ( ( ( if( ( (-1,0) >= 8 ) , 8 , 0 )  = 8 ) and ( if(
( ( ( ( ( ( ( ( (0,-1) = 2 ) or ( (0,-1) = 3 ) ) or ( (0,-1) = 6 ) )
or ( (0,-1) = 7 ) ) or ( (0,-1) = 10 ) ) or ( (0,-1) = 11 ) ) or ( (0
,-1) = 14 ) ) or ( (0,-1) = 15 ) ) , 2 , 0 )  = 2 ) ) and ( if( ( (
-1,0) > 11 ) or ( ( (-1,0) > 3 ) and ( (-1,0) < 8 ) ) ) , 4 , 0 )  =
0 ) ) and ( if( odd((0,1))  , 1 , 0 )  = 0 ) ) and even(cellpos(0) +
 cellpos(1)) }
rule : 5 100 { ( ( ( ( if( ( (0,-1) >= 8 ) , 8 , 0 )  = 8 ) and ( if(
( ( ( ( ( ( ( ( (1,0) = 2 ) or ( (1,0) = 3 ) ) or ( (1,0) = 6 ) ) or
( (1,0) = 7 ) ) or ( (1,0) = 10 ) ) or ( (1,0) = 11 ) ) or ( (1,0) = 14
) ) or ( (1,0) = 15 ) ) , 2 , 0 )  = 2 ) ) and ( if( ( ( (0,-1) > 11 )
or ( ( (0,-1) > 3 ) and ( (0,-1) < 8 ) ) ) , 4 , 0 )  = 0 ) ) and ( if
( odd((0,1))  , 1 , 0 )  = 0 ) ) and odd(cellpos(0) + cellpos(1)) }
rule : {{ ( ( ( if( ( ( (0,-1) > 11 ) or ( ( (-1,0) > 3 ) and ( (-1,0) < 8
) ) ) , 4 , 0 )  + if( odd((0,1))  , 1 , 0 )  ) + if( ( ( ( ( ( ( ( (
0,-1) = 2 ) or ( (0,-1) = 3 ) ) or ( (0,-1) = 6 ) ) or ( (0,-1) = 7 ) )
or ( (0,-1) = 10 ) ) or ( (0,-1) = 11 ) ) or ( (0,-1) = 14 ) ) or ( (0,-1
) = 15 ) ) , 2 , 0 )  ) + if( ( (-1,0) >= 8 ) , 8 , 0 )  )} 100 { t and
even(cellpos(0) + cellpos(1)) }
rule : {{ ( ( ( if( ( ( (0,-1) > 11 ) or ( ( (0,-1) > 3 ) and ( (0,-1) < 8 )
) ) , 4 , 0 )  + if( odd((0,1))  , 1 , 0 )  ) + if( ( ( ( ( ( ( ( (1,0)
= 2 ) or ( (1,0) = 3 ) ) or ( (1,0) = 6 ) ) or ( (1,0) = 7 ) ) or ( (1,0
) = 10 ) ) or ( (1,0) = 11 ) ) or ( (1,0) = 14 ) ) or ( (1,0) = 15 ) ) ,
2 , 0 )  ) + if( ( (0,-1) >= 8 ) , 8 , 0 )  )} 100 { t and odd(cellpos(0)
+ cellpos(1)) }
```

Figure 62. Translated rules from Triangular to Square

```
[top]

Components : gasdiffusion

[gasdiffusion]
type : cell
width : 15
height : 15
delay : transport
defaultDelayTime  : 100
border : nowrapped
neighbors :             gasdiffusion(-1,0)
neighbors : gasdiffusion(0,-1)  gasdiffusion(0,0)  gasdiffusion(0,1)
neighbors :             gasdiffusion(1,0)
initialvalue : 0
initialrow :  4      0  0  0  0  0  2  0  0  0  0  0  0  0  0  0
initialrow :  5      0  0  0  0  3  5  1  0  0  0  0  0  0  0  0
initialrow :  6      0  0  0  0  0  1  0  0  0  0  0  0  0  0  0
initialrow :  11     0  0  0  0  0  0  0  0  0  5  4  0  0  0  0
initialrow :  12     0  0  0  0  0  0  0  0  0 10  4  3  0  0  0
localtransition : gasdiffusion-rule

[gasdiffusion-rule]
rule : 10 100 { ( ( ( ( if( ( ( (-1,0) > 11 ) or ( ( (-1,0) > 3 )
and ( (-1,0) < 8 ) ) ) , 4 , 0 )  = 4 ) and ( if( odd((0,1))  , 1
, 0 )  = 1 ) ) and ( if( ( (-1,0) >= 8 ) , 8 , 0 )  = 0 ) ) and (
if( ( ( ( ( ( ( ( (0,-1) = 2 ) or ( (0,-1) = 3 ) ) or ( (0,-1) =
6 ) ) or ( (0,-1) = 7 ) ) or ( (0,-1) = 10 ) ) or ( (0,-1) = 11 ) )
or ( (0,-1) = 14 ) ) or ( (0,-1) = 15 ) ) , 2 , 0 )  = 0 ) ) and
even(cellpos(0) + cellpos(1)) }
rule : 10 100 { ( ( ( ( if( ( ( (0,-1) > 11 ) or ( ( (0,-1) > 3 )
and ( (0,-1) < 8 ) ) ) , 4 , 0 )  = 4 ) and ( if( odd((0,1))  , 1 ,
0 )  = 1 ) ) and ( if( ( (0,-1) >= 8 ) , 8 , 0 )  = 0 ) ) and ( if
( ( ( ( ( ( ( ( (1,0) = 2 ) or ( (1,0) = 3 ) ) or ( (1,0) = 6 ) )
or ( (1,0) = 7 ) ) or ( (1,0) = 10 ) ) or ( (1,0) = 11 ) ) or ( (1,
0) = 14 ) ) or ( (1,0) = 15 ) ) , 2 , 0 )  = 0 ) ) and odd(cellpos(
0) + cellpos(1)) }
rule : 5 100 { ( ( ( ( if( ( (-1,0) >= 8 ) , 8 , 0 )  = 8 ) and ( if(
( ( ( ( ( ( ( ( (0,-1) = 2 ) or ( (0,-1) = 3 ) ) or ( (0,-1) = 6 ) )
or ( (0,-1) = 7 ) ) or ( (0,-1) = 10 ) ) or ( (0,-1) = 11 ) ) or ( (0
,-1) = 14 ) ) or ( (0,-1) = 15 ) ) , 2 , 0 )  = 2 ) ) and ( if( ( (
-1,0) > 11 ) or ( ( (-1,0) > 3 ) and ( (-1,0) < 8 ) ) ) , 4 , 0 )  =
0 ) ) and ( if( odd((0,1))  , 1 , 0 )  = 0 ) ) and even(cellpos(0) +
 cellpos(1)) }
rule : 5 100 { ( ( ( ( if( ( (0,-1) >= 8 ) , 8 , 0 )  = 8 ) and ( if(
( ( ( ( ( ( ( ( (1,0) = 2 ) or ( (1,0) = 3 ) ) or ( (1,0) = 6 ) ) or
( (1,0) = 7 ) ) or ( (1,0) = 10 ) ) or ( (1,0) = 11 ) ) or ( (1,0) = 14
) ) or ( (1,0) = 15 ) ) , 2 , 0 )  = 2 ) ) and ( if( ( ( (0,-1) > 11 )
or ( ( (0,-1) > 3 ) and ( (0,-1) < 8 ) ) ) , 4 , 0 )  = 0 ) ) and ( if
( odd((0,1))  , 1 , 0 )  = 0 ) ) and odd(cellpos(0) + cellpos(1)) }
rule : {{ ( ( if( ( ( (0,-1) > 11 ) or ( ( (-1,0) > 3 ) and ( (-1,0) < 8
) ) ) , 4 , 0 )  + if( odd((0,1))  , 1 , 0 )  ) + if( ( ( ( ( ( ( ( (
0,-1) = 2 ) or ( (0,-1) = 3 ) ) or ( (0,-1) = 6 ) ) or ( (0,-1) = 7 ) )
or ( (0,-1) = 10 ) ) or ( (0,-1) = 11 ) ) or ( (0,-1) = 14 ) ) or ( (0,-1
) = 15 ) ) , 2 , 0 )  ) + if( ( (-1,0) >= 8 ) , 8 , 0 )  )} 100 { t and
even(cellpos(0) + cellpos(1)) }
rule : {{ ( ( if( ( ( (0,-1) > 11 ) or ( ( (0,-1) > 3 ) and ( (0,-1) < 8 )
) ) , 4 , 0 )  + if( odd((0,1))  , 1 , 0 )  ) + if( ( ( ( ( ( ( ( (1,0)
= 2 ) or ( (1,0) = 3 ) ) or ( (1,0) = 6 ) ) or ( (1,0) = 7 ) ) or ( (1,0
) = 10 ) ) or ( (1,0) = 11 ) ) or ( (1,0) = 14 ) ) or ( (1,0) = 15 ) ) ,
2 , 0 )  ) + if( ( (0,-1) >= 8 ) , 8 , 0 )  )} 100 { t and odd(cellpos(0)
+ cellpos(1)) }
```

Figure 63. Coupled Model for gasdiffusiontri.ma

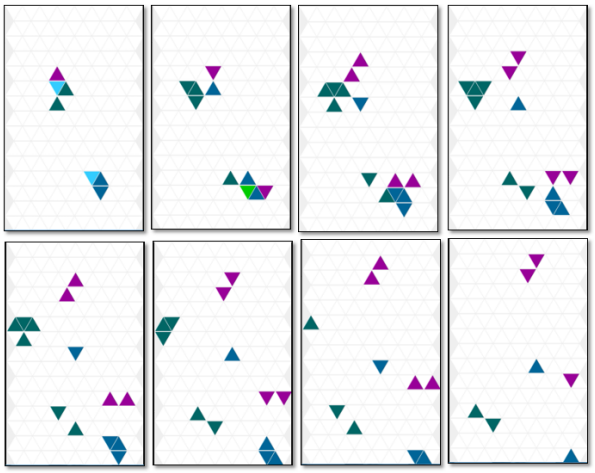

Figure 64. Simulation Result for gasdiffusiontri.ma

The model is now observed to understand the behavior of gas diffusion using a triangular topology, the coupled model referencing to this can be seen in figure 63. The translated rules are shown in figure 62 to build a square topology. Figure 64 shows the simulation results of gasdiffusiontri.ma.

4. CONCLUSION

The tool CD++ can be used to model multi-dimensional Cell-DEVS. Using the specification language for defining the rules for triangular and hexagonal topologies we can visualize the behavior of physical systems on a hexagonal and triangular grid provided in the CD++ modeler. As hexagonal and triangular topologies consider few neighbors the number of messages passed between the neighbors are low as compared to the regular square geometry, hence we lose a lot of information. On the other hand, the isotropic behavior of a hexagonal mesh and the fact that triangular mesh considers the limited number of neighbors are taken as a n advantage when implementing physical systems like diffusion and excitable media, as it gives us a better visualization.

5. REFERENCES

[1] G.Wainer: Application of Cell-DEVS Methodology for Modeling the Environment.

[2] Wolfram, S. 2002. A New Kind of Science. Champaign, IL: Wolfram Media

[3] B. Zeigler; T. Kim; H. Praehofer: Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems, Academic Press, 2000

[4] G. Wainer; N. Giambiasi: "Application of the CellDEVS Paradigm for Cell Spaces Modeling and Simulation", Simulation, Vol. 71, No. 1, pp. 22-39, January 2001.

[5] G. Wainer: "CD++: A Toolkit to Define DiscreteEvent Models", Software, Practice and Experience, Wiley, Vol. 32, No 3. pp. 1261-1306. November 2002.

[6] G. Wainer, Javier Ameghino: APPLICATION OF THE CELL-DEVS PARADIGM USING N-CD++.

[7] G. Wainer, Alejandro Troccoli: Implementing Parallel Cell-DEVS

[8] G.Wainer: Sample Models Webpage