APPLICATION OF THE CELL-DEVS PARADIGM USING N-CD++

Javier Ameghino Gabriel Wainer.

Departamento de Computación. Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires.
Planta Baja. Pabellón I. (1428) Buenos Aires. ARGENTINA
gabrielw@dc.uba.ar
http://www.dc.uba.ar/people/proyinv/celldevs

ABSTRACT

Cell-DEVS is an extension of the DEVS formalism that allows the definition of cellular models. We have developed a tool implementing these theoretical concepts, making easy the definition of complex cell spaces with explicit timing delays. Here, we show the use of the paradigm through different application examples. Complex applications can be implemented in a simple fashion, and they can be executed effectively.

Keywords: Modeling methodology: DEVS models, cellular automata, Cell-DEVS models; Simulation tools.

INTRODUCTION

The DEVS formalism, defined in (Zeigler 1976, Zeigler 1984, Zeigler *et al.* 2000) is a modular and hierarchical approach for discrete-event simulation. A DEVS model can be described as composed of several submodels, which, once tested, can be reused.

We have interested in describing real systems that can be represented as cell spaces. Cellular Automata (Wolfram 1986) is a well-known formalism to describe these systems. They are defined as an infinite n-dimensional lattice of cells whose values are updated according to a local rule. This is done simultaneous and synchronously, using the present cell state and those of a finite set of nearby cells (called its **neighborhood**).

Cellular automata usually require large amounts of compute time. The use of a discrete time base also constrains the model precision. Timed Cell-DEVS solves these problems by using the DEVS paradigm to define a cell space where each cell is defined as an atomic model (Wainer and Giambiasi, 2000). The goal is to build discrete-event cell spaces, improving their definition by making the timing specification more expressive. Cell-DEVS atomic models can be specified as:

$$TDC = < X, Y, I, S, N, delay, d, \delta_{INT}, \delta_{EXT}, \tau, \lambda, D >$$

In this case, **X** is the set of external input events, **Y** is the set of external output events, and **I** is the model's modular interface. **S** is the set of sequential states for the cell, and **N** is the set of input events. **Delay** defines the kind of delay for the cell, and **d** its duration. Finally, there are several functions: δ_{INT} for internal transitions, δ_{EXT} for external transitions, τ for local computations, λ for outputs and **D** for the state's duration function.

Each cell uses a set of N inputs to compute the future state. They are received through the model interface, and are used to activate the local function. A delay can be associated with each cell, allowing deferring the transmission of the execution results. A **transport** delay allows us to model a variable commuting time for each cell with anticipatory semantics (every scheduled event is executed). Using **inertial** delays, the semantics is preemptive: some scheduled events are not executed due to a small interval between two input events. Therefore, the outputs of a cell are not transmitted instantaneously, but after the consumption of the delay. The model advances through the activation of the internal, external, output and state's duration functions, as in other DEVS models.

Once each cell is defined, they can form a coupled model:

$$GCC = < Xlist, Ylist, I, X, Y, n, \{t1,..,tn\}, N, C, B, Z >$$

Here, **Xlist** is an input coupling list, **Ylist** is an output coupling list and **I** represents the model interface. **X** is the set of external input events and **Y** the external output events. The **n** value defines the dimension of the cell space, **{t1,...,tn}** is the number of cells in each dimension and **N** is the neighborhood set. **C** is the cell space, **B** is the set of border cells, and **Z** a translation function.

This work was partially supported by ANPCYT Project 11-04460, UBACYT Project JW10, and COMPAQ Latin America Corporation.

This specification defines a coupled model composed of an array of atomic cells. Each of them is connected to its neighborhood. As the cell space is finite, the borders should be provided with a different behavior than the rest of the space. Otherwise, the space is wrapped, meaning that cells in a border are connected with those in the opposite one. Finally, the Z function defines internal and external couplings.

Using these formal specifications, a simulation tool was developed. The following sections will be devoted to show how the tool can be used to develop cellular models.

A SIMULATION TOOL BASED ON CELL--DEVS

N–CD++ (Rodríguez and Wainer, 1999) is a tool built following the formal specifications of Cell-DEVS. The tool includes a specification language that allows describing the behavior of each cell. A cell space is defined including its size, influencees, neighborhood and borders. Using these parameters, a complete Cell-DEVS is built using the formal specifications.

The behavior of a cell is defined using rules with the form:

VALUE DELAY { CONDITION }

Each rule indicates that if the *CONDITION* is satisfied, the state of the cell will change to the designated *VALUE*, and it will be *DELAY*ed the specified time. If the condition is not valid, the next rule is evaluated (according to the order in that they were defined), repeating this process until a rule is satisfied.

The specification shown in Figure 1 shows the rules for the "Life" Game (Gardner, 1970). They indicate that if a cell will remain active when the number of active neighbors is 3 or 4 (using a transport delay of 10 ms). If the cell is inactive and the neighborhood has 3 active cells, the cell activates. Otherwise case, the cell is inactive.

```
Rule: 1 10 { (0,0) = 1 and ( truecount = 3 or
                truecount = 4 ) }
Rule: 1 10 { (0,0) = 0 and truecount = 3 }
Rule: 0 10 { t }
```

Figure 1. Definition for the Life game.

The language permits to manipulate n-dimensional references. Likewise, a neighborhood can be composed of non-adjacent cells, and the neighborhood's dimension can be similar or inferior to the model's dimension.

The most common operators are included: boolean (*AND*, *OR*, *NOT*, *XOR*, *IMP* and *EQV*), comparison (=, !=, <, >, <= and >=), and arithmetic (+, -, * and /). In addition, different types of functions are available: trigonometric, roots, power, rounding and truncation, module, logarithm, absolute value, minimum, maximum, G.C.D. and L.C.M. Other existing functions allow to check if a number is integer, even, odd or prime.

Some functions allow to query the cell state of the neighborhood: *truecount, falsecount*, *undefcount* and *statecount*(*n*).

Some common constants are defined: *pi*, *e*, certain constants used in the domains of the physics and the chemistry (gravitational constant, acceleration, light speed, Planck's constant).

The *Time* function returns the global simulated time. Functions *RadToDeg* and *DegToRad* are used for the conversion of angles expressed in radians to degrees and viceversa. There are functions for the conversion of polar and rectangular coordinates and temperatures in Celsius, Fahrenheit or Kelvin degrees.

Other functions allow to obtain values depending on the evaluation of a certain condition. *IFU*(*c*, *t*, *f*, *u*) evaluates the *c* condition, and if it is *True*, it returns the *t* value. If it is *False*, it return *f*, and *u* if it is *undefined*. On the other hand, the function *IF*(*c*, *t*, *f*) returns *t* if *c* evaluates to *True*, and *f* otherwise.

Finally, several functions are used to generate pseudorandom numbers using different probability distributions. We have included Uniform, Chi Square, Beta, Exponential, Φ, Gamma, Gaussian, Binomial and Poisson distributions. The introduction of random conditions presents new problems. For example, in the rule:

10 100 { random >= 0.4 }

the condition is evaluated to *True* in the 60 % of the cases, and to *False* in the rest. Therefore, the model could return all the rules evaluated to *False*, even when the model is well specified. The tool automatically identifies these cases, and assigns the *Undefined* value to the cell, informing this situation and continuing with the simulation.

Space *zones*, defined by a cell range, can be associated with a set of rules different than the rest of the cell space.

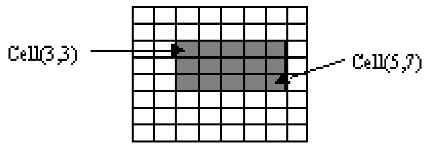

Figure 2. A Zone defined by the range {(3,3)..(5,7)}

Cellular models can be integrated to a standard DEVS hierarchy. Therefore, input/output ports can be defined for the cell space. The *portInFunction* directive is used to query the value of a message arrived in an input port. On the other hand, output ports are activated using the *send* function.

APPLICATION EXAMPLES

This section is devoted to describe how to implement different Cell-DEVS models using N-CD++.

We first show an example of excitable media, a phenomenon appearing in several real systems. For instance, the nervous tissue of the heart muscle is an excitable medium where a wave travels through the heart in every heartbeat. Magnetic fields, forest fires, etc. also can be represented as modifications of these models.

Three states can be recognized. For instance, in a heart tissue these states represent a cell that is resting, excited or recovering. In a forest fire, the states could represent a cell without fire, burning or burnt. Any of these models are defined as:

```
[ExMedia]
type : cell          dim : (9,9)
delay : transport    border : wrapped
neighbors : (-1,-1) (-1,0) (-1,1) (0,-1)
neighbors : (0,1) (1,-1) (1,0) (1,1) (0,0)
localtransition : Ex-rules

[Ex-rules]
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 3. Definition of an excitable media model.

We first define the Cell-DEVS coupled model and its parameters: size, neighborhood shape, kind of delay and borders. The *Ex-rules* section represents the local computing function for the model. Here, the first rule defines that when the cell and its neighbors are not excited (value *0*), the cell must keep resting. The second rule means that if the cell is resting and there are excited neighbors (value *2*), the cell is excited. Third and fourth rules mean that the cells remain in a specific state. In every other case (*t* means "True"), the cell keeps its present state.

Figure 4 shows the results obtained when this model executes. It shows the evolution of this excitable media model using different neighborhoods. Figure 4a uses all the adjacent neighbors, as defined in Figure 3 (Moore neighborhood). Figure 4b uses four adjacent cells (N-S-E-W), and Figure 4c shows a hexagonal lattice on a square grid.

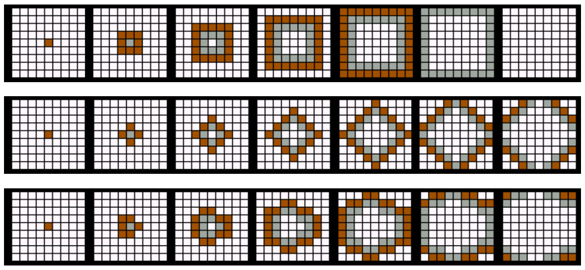

Figure 4. Results of ExMedia with different Neighborhoods: (a) Moore; (b) Von Neumann; (c) hexagonal grid.

The following example represents a surface tension model.

```
[Tension]
type : cell          dim : (40,40)
delay : transport    border : wrapped
neighbors : (-1,-1) (-1,0) (-1,1) (0,-1)
neighbors : (1,-1) (1,0) (1,1) (0,0) (0,1)
localtransition : Ten-rules

[Ten-rules]
rule : 0 100 { statecount(0) >= 5 }
rule : 1 100 { t }
```

Figure 5. Surface tension model specification

We see the definition of the Cell-DEVS coupled model parameters: a grid of 40x40, Moore neighborhood, transports delays and wrapped borders. We have two states: the presence (value *1*) or absence (value *0*) of particles. This model can be represented as a "majority vote" system. In each step, the new state depends of most neighbors. It remains in the cell if at least 5 of the 9 are occupied; otherwise it becomes empty. The following figure shows how particles concentrate where there is bigger tension. The resulting behavior of the surface is a high level representation of the majority vote rules defined earlier.

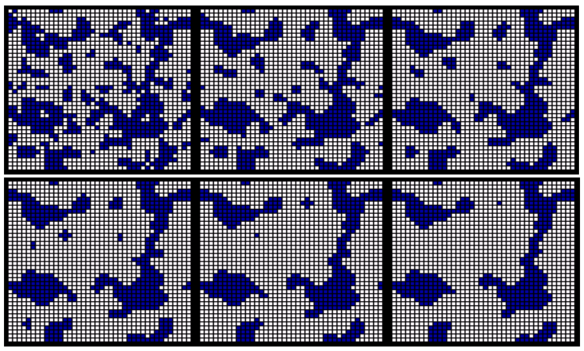

Figure 6. Execution results of the Surface Tension model.

The next example is an ecological model of ants moving in the ground. The ants eat grass in two steps: first, they eat the leaves and then the root. Hence, we have three grass states: *1* when there is grass in the cell, *2* when the leaves have been eaten, and *3* or *4* for empty cells. When an ant find grass, she eat it and rotates 90º to the right. When the leaves have been eaten, she eats the root and rotates 90º to the left. If there is no grass, she continues moving ahead.

```
[vants]
type : cell              dim : (10,10,2)
delay : transport        border : wrapped
neighbors: (-2,0,0) (-1,-1,0) (-1,0,0)
neighbors: (-1,1,0) (0,2,0) (0,0,1)
neighbors: (0,-2,0) (0,-1,0) (0,0,0) (0,1,0)
neighbors: (1,-1,0) (1,0,0) (1,1,0) (2,0,0)

localtransition : calculus
[calculus]
...
rule:{if((0,0,0)=45,31,(trunc((0,0,0)/10)*10)+
1) } 100 { cellpos(2)=0 and (0,0,1)=2 and
((0,0,0)=25 or (0,0,0)=35 or (0,0,0)=45) and
(fractional((-1,0,0)/10)*10)!=7 and
((fractional((-1,1,0)/10)*10)=8 or
(fractional((-2,0,0)/10)*10)=7 or
(fractional((-1,-1,0)/10)*10)=6) }
...
rule : { (trunc(((0,0,0)/10))*10) + 1 } 100
{cellpos(2)=0 and (0,0,1)=4 and ((1,0,0)=25 or
(1,0,0)=35 or (1,0,0)=45)}
...
rule : { (0,0,0) + 1 } 100 {cellpos(2)=1 and
(0,0,0)<7}
rule : 1 100 {cellpos(2)=1 and (0,0,0)>=7}

rule : { (0,0,0) } 100 { t }
```

Figure 7. Specification of the ant's model

We use four states to represent the present direction, combined with the grass state. For instance, a value *13* means that there is grass (*1*) and the ant is going north (*1*: S; *2*: E; *3*: N; *4*: W). This model needs extra information to control the movement of the ants to avoid collisions. Then, we define auxiliary ant directions to analyze the possibility of collision (5: S; 6: E; 7: N; 8: W).

We have two surfaces, the first representing the grass and the ants, and the second used as collision flags. When a cell in the second surface have the value *1* we must analyze only rules that determine whether there are collisions or not. When the value of the cell is *2* we must analyze only rules to compute the next position of the ants. This is shown in the first rule in Figure 7. The following rule is used to move the ant once we know that we can move it. The *cellpos* function allows us to know which surface we are using, providing two different behaviors.

The following figure shows the execution of the model using the tool. The dark cells contain grass, and an ant moving in the lower rows is eating them. The ant behaves using the rules recently explained. We also can see the growth of the grass, represented as a change of state for the lighter cells, where a long delay is used.

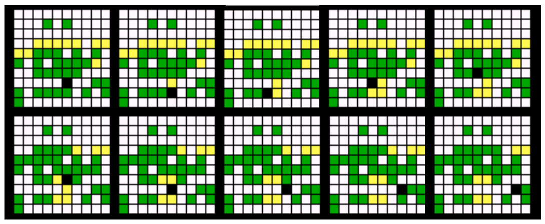

Figure 8. Execution results of an ant foraging model.

Our last example shows the simulation results for a watershed model. This model, previously introduced in (Moon *et al.* 1996), represents a hydrology system built as a cell space. The watershed is represented as small cells organized in several layers (air, surface water, soil, ground water, and bedrock). The rainfall input is partially retained by vegetation, and the rest infiltrates.

```
[Watershed]
type : cell         dim : (30,30,2)
delay : transport   border : nowrapped
neighbors : (-1,0,0) (0,-1,0) (0,0,0) (0,1,0)
neighbors : (1,0,0) (-1,0,1) (0,-1,1) (0,0,1)
neighbors : (1,0,1) (0,1,1)
localtransition : Hydrology

[Hydrology]

rule : {0.0022 + (0,0,0) - if ((-1,0,0)!=?) and
((0,0,1)+(0,0,0) > ((-1,0,1) + (-1,0,0)),
((0,0,0)+(0,0,1)-(-1,0,0)-(-1,0,1))/1000)*
(0,0,0))/1000),0)-
if((1,0,0)!=?) and ((0,0,1)+(0,0,0))> ((1,0,1) +
(1,0,0)),((0,0,0) + (0,0,1) - (1,0,0) -
(1,0,1))/1000)*(0,0,0))/1000),0) - if((((0,-1,0)
!= ?) and
((0,0,1)+(0,0,0)) > ((0,-1,1)+(0,-1,0)) ,((0,0,0)
+ (0,0,1)-(0,-1,0)-(0,-1,1))/1000)
*(0,0,0))/1000),0) -
if((0,1,0) != ?) and ((0,0,1) + (0,0,0)) >
((0,1,1) + (0,1,0)),(((0,0,0) + (0,0,1) - (0,1,0)
- (0,1,1))/1000) * (0,0,0))/1000),0) + if((-
1,0,0) != ?) and ((-1,0,1) + (-1,0,0)) > ((0,0,1)
+ (0,0,0)),((-1,0,0) + (-1,0,1) - (0,0,0)-
(0,0,1))*(-1,0,0))/1000),0) + if((1,0,0) != ?)
and ((1,0,1) + (1,0,0)) > ((0,0,1) +
(0,0,0)),((1,0,0) + (1,0,1) - (0,0,0) - (0,0,1))
* (1,0,0))/1000),0) + if((0,-1,0) != ?) and ((0,-
1,1) + (0,-1,0)) >((0,0,1) + (0,0,0)), ((0,-1,0)
+ (0,-1,1) - (0,0,0) - (0,0,1)) * (0,-
1,0))/1000),0) + if((0,1,0) != ?) and (((0,1,1) +
(0,1,0)) > (0,0,1) + (0,0,0)),((0,1,0) + (0,1,1)
- (0,0,0) - (0,0,1)) * (0,1,0))/1000),0) } 1000 {
cellpos(2)=0 }
rule : { (0,0,0) } 1000 { t }
```

Figure 9. Watershed model specification

Figure 9 shows the implementation of the model using the tool. The rules represent the accumulation of water. It takes the present water of the cell, and the rain fell up to the present moment is added. Then, we consider how much water must be passed to the neighbors, and how much water is received from the inverse neighborhood. Several layers are represented as planes in a three dimensional model.

We can see the execution results of this model in the Appendix. In the first figure we show an initial state, representing the slope of the terrain before raining. Each cell occupies 1m x 1m. The second figure shows the execution results after intense rain (0.0022 mm/s) after 10 minutes of simulated time. We can see that the rain is accumulated in the lower levels of the terrain, and a river is formed.

CONCLUSION

The tool N-CD++ can be used for modeling and simulation of n–dimensional Cell–DEVS. This approach improves the development, checking and maintaining phases, facilitating the testing and reuse of their components. Input/output port definitions allow defining multiple interconnection between Cell-DEVS or DEVS models. We also can develop multidimensional models, making the tool a general framework to define and simulate complex generic models.

The tool was built using a standard version of C++, which allowed is to provide different versions running in several platforms without additional cost. At present, N–CD++ has been successfully tested in Windows 95/NT, Linux, AIX, IRIX, HP-UX and Solaris. A simulation server has been built, and soon it will be available to do simulations using a Web-based approach.

The tool and the examples are the public domain and they can be obtained in:
"http://www.dc.uba.ar/people/proyinv/celldevs".

REFERENCES

Gardner, M., 1970. “The Fantastic Combinations of John Conway’s New Solitaire Game ‘Life’ ”. *Scientific American, 23* (4), pp. 120-123.

Moon, Y.; Zeigler, B.; Ball, G.; Guertin, D., 1996. "DEVS representation of spatially distributed systems: validity, complexity reduction". *IEEE Transactions on Systems, Man and Cybernetics*. pp. 288-296.

Rodríguez, D.; Wainer, G., 1999. "New Extensions to the CD++ tool". In *Proceedings of SCS Summer Multiconference on Computer Simulation*. 1-7.

Wainer, G.; Giambiasi, N., 2000. "Timed Cell-DEVS: modelling and simulation of cell spaces". In *Discrete Event Modeling & Simulation: Enabling Future Technologies*, to be published by Springer-Verlag.

Wolfram, S. , 1986. *Theory and applications of cellular automata*. Vol.1, Advances Series on Complex Systems. World Scientific, Singapore.

Zeigler, B., 1976. *Theory of modeling and simulation*. Wiley.

Zeigler, B., 1984. *Multifaceted Modelling and discrete event simulation*. Academic Press.

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

BIOGRAPHIES

Gabriel A. Wainer received the Licentiate degree (M.Sc., 1993) and Ph.D. degree (1998, with honours) of the Universidad de Buenos Aires, Argentina, and DIAM/IUSPIM, Université d'Aix-Marseille III, France. He is Assistant Professor at the Computer Sciences Dept. of the Universidad de Buenos Aires, Argentina. He has been working in the same department since 1988. He published more than 40 articles in the field of operating systems, real-time systems and Discrete-Event simulation. He has been head of several research projects related with computer modelling and simulation. He is author of a book on real-time systems and co-author of a book on Operating Systems.

Javier Ameghino is a Licentiate (M. Sc.) student at the Computer Sciences Department of the Universidad de Buenos Aires, Argentina. He worked in several private companies in Buenos Aires. At present he works at COMPAQ Latin America Corporation (Buenos Aires).

APPENDIX – Execution results of the Watershed model.

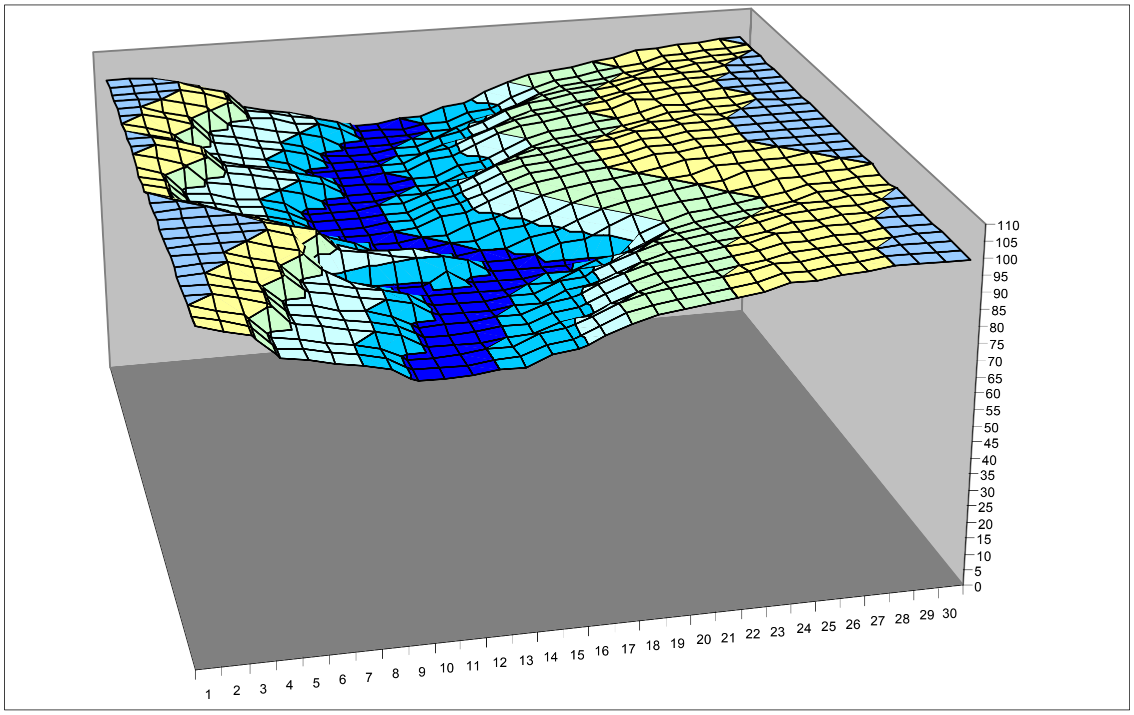

Figure 10. Initial height values for a watershed.

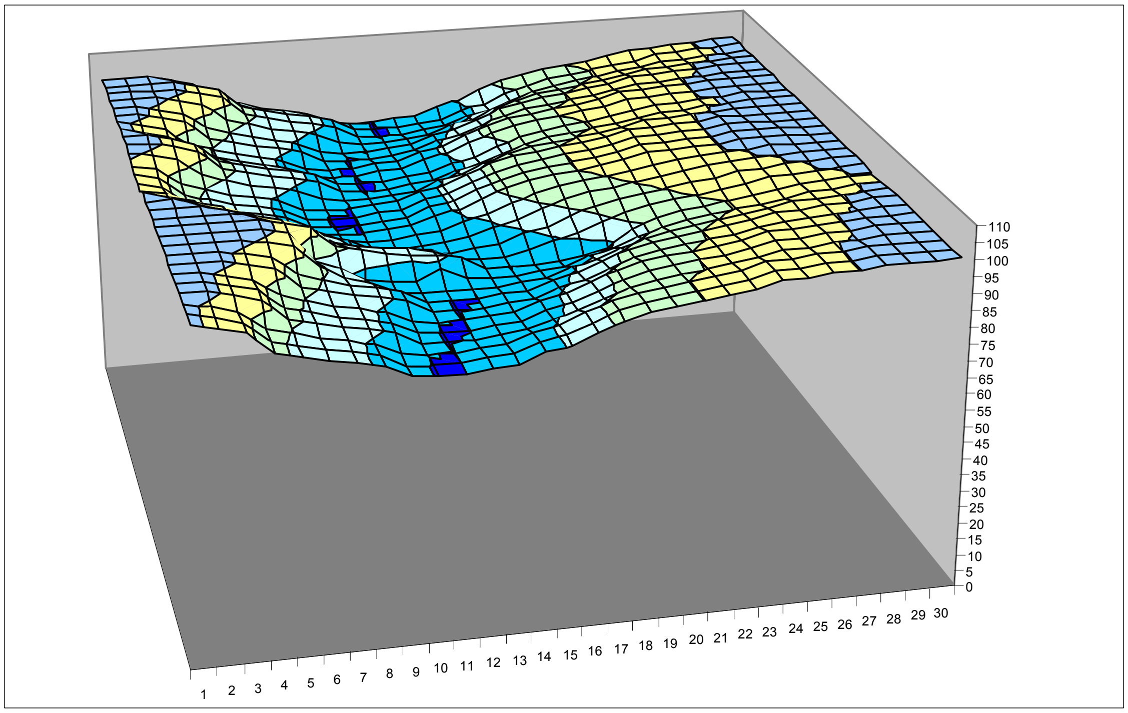

Figure 11. Height values after rain.