On the use of CD++/Maya for visualization of discrete-event models

Ayesha Khan
Gabriel A. Wainer
Wilson Venhola

Department of Systems and Computer Engineering

Michael Jemtrud

Carleton Immersive Media Studio

Carleton University
1125 Colonel By Drive
Ottawa, Ontario, K1S 5B6, Canada
{amkhan2@connect, gwainer@sce, wvenhola@connect, michael_jemtrud@}carleton.ca

Abstract – The CD++ modeling tool was created to study complex systems by using DEVS and Cell-DEVS formalisms. Although CD++ was successfully employed to define a wide variety of models for complex applications, visualization facilities were limited. We have introduced a 3D visualization engine, which creates virtual worlds using the Maya visualization environment with simulations running in CD++, to permit interaction with DEVS models. These tools are able to run in advanced environments based on OpenGL. We show different examples, and discuss current developments on the creation of a new visual environment based on open source OpenGL tools.

Keywords: DEVS, Cell-DEVS, Maya, OpenGL, 3D visualization.

1. INTRODUCTION

In recent years, the Discrete Event Systems Specification (DEVS) formalism [Zei, 00] has gained popularity to model a variety of problems. DEVS is a framework for the construction of discrete-event hierarchical modular models, in which behavioral models (**atomic**) can be integrated together forming a hierarchical structural model (**coupled**). The Cell-DEVS formalism [Wai, 01] extended the DEVS formalism allowing the simulation of discrete-event cellular models. The approach extends traditional Cellular Automata [Wol, 02] defining each cell as a DEVS atomic model and the space as a DEVS coupled model, including a flexible way of defining the timing of each cell.

The CD++ tool [Wai, 02] implements DEVS and Cell-DEVS theories. CD++ enabled us to solve successfully over 150 models of complex systems [Wai, 04; Wai, 05; Ame, 04, Ame, 01, Ame, 03]. CD++ also provides remote access to a high performance DEVS simulation server [Wai, 03]. The end user tools were organized as a simulation client applied to the CD++ simulator. Using these facilities, the users can now develop and test their models in local workstations, and submit them to be simulated in a remote CD++ server executing in a high performance platform. Then, they can receive, visualize and analyze the results on a local computer, improving model definition and execution.

Visualization tools are crucial in helping to understand better the behavior of these systems. Originally, CD++ only provided results on text files, and different visualization engines were incorporated. CD++ was recently provided with facilities for 2D and 3D visualization using VRML and Java [Wai, 03]. This 3D GUI enables sophisticated visualization of Cell-DEVS models only. These facilities were built on VRML, a technology becoming obsolete. These methods have some limitations. The Java applets use Java3D libraries and the VRML specification, both which are no longer actively developed. The current VRML viewers also lack functionality and ease of use. Hence, we have recently focused on new extensions that can be applied to both DEVS and Cell-DEVS, and which are able to run on OpenGL-based environments [Seg, 05]. The interface here presented was first introduced in [Kha, 05], and it is based on the Maya modeling environment [Ali, 05]. We will show how advanced DEVS models can be visualized using Maya facilities, giving a few examples of application, which permit discussing interoperability of a modeling and simulation tool based on DEVS and an advanced generic visualization environment like Maya.

2. BACKGROUND

DEVS is a systems theoretical approach that allows the definition of hierarchical modular models [Zei, 00]. A real system modeled using DEVS can be described as a set of atomic or coupled submodels. The atomic model is the lowest level and defines dynamics, while the coupled are structural models composed of one or more atomic and/or coupled models. An atomic DEVS model is defined as:

$$M = < X, S, Y, d_{int}, d_{ext}, ?, ta >$$

Each atomic model can be seen as having an interface consisting of *input* (X) and *output* (Y) *ports* to

communicate with other models. Every *state* (S) in the model is associated with a *time advance* (ta) function, which determines the duration of the state. Once the time assigned to the state is consumed, an internal transition is triggered. At that moment, the model execution results are spread through the model's output ports by activating an *output function* (λ). Then, an *internal transition function* (δ_{int}) is fired, producing a local state change. Input external events are collected in the input ports. An external transition function (δ_{ext}) specifies how to react to those inputs.

A coupled DEVS model is defined as:

$$CM = < X, Y, D, \{Mi\}, \{Ii\}, \{Zij\}, select >$$

Coupled models are defined as a set of basic components (atomic or coupled), which are interconnected through the models' interfaces. The models' coupling defines how to convert the outputs of a model into inputs for the others, and how to handle inputs/outputs from/to external models

Cell-DEVS has extended the DEVS formalism, allowing the implementation of cellular models with timing delays. A Cell-DEVS model is informally presented in Figure 1.

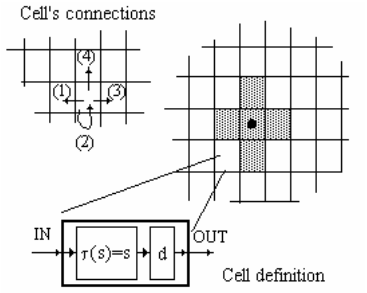


Figure 1: Description of a Cell-DEVS coupled model.

Once the cell behavior is defined, a coupled Cell-DEVS can be created by putting together a number of cells interconnected with their neighbors.

A cellular model is a lattice of cells holding state variables and a computing apparatus, which is in charge of updating the cell state according to a local rule. This is done using the present cell state and those of a finite set of nearby cells (called its neighborhood). Each cell is defined as a DEVS atomic model, and it can be later integrated to a coupled model representing the cell space. Each cell uses N inputs to compute its next state. These inputs, which are received through the model's interface, activate a local computing function (τ). A delay (**d**) can be associated with each cell. The state (**s**) changes can be transmitted to other models, but only after the consumption of this delay.

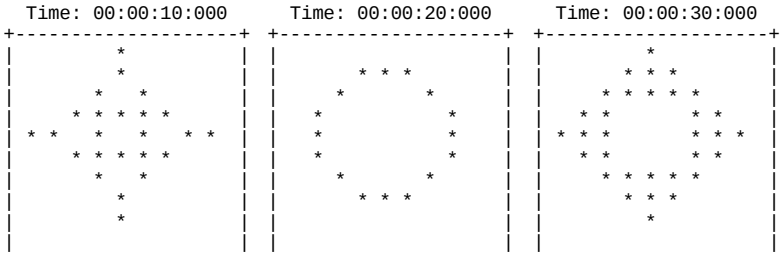
CD++ [Wai, 02] is a modeling and simulation toolkit that implements DEVS and Cell-DEVS theory. Atomic models can be defined using a state-based approach (coded in C++ or an interpreted graphical notation), while coupled and Cell-DEVS models are defined using a built-in specification language. We will show the basic features of the tool through an example of application. CD++ also includes an interpreter for Cell-DEVS models. The model specification includes the definition of the size and dimension of the cell space, the shape of the neighborhood and borders. The cell's local computing function is defined using a set of rules with the form: POSTCONDITION DELAY {PRECONDITION}. These indicate that when the *PRECONDITION* is satisfied, the state of the cell will change to the designated *POSTCONDITION*, whose computed value will be transmitted to other components after consuming the *DELAY*. Figure 2 shows the definition of a very simple example of the definition of such models.

```
[life]
size:(20,20) delay:transport border:wrapped
neighbors : (-1,-1)(-1,0)(-1,1)(0,-1) (0,0)
            (0,1) (1,-1)  (1,0)  (1,1)
localtransition : life-rule

[life-rule]
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 2: Definition of the Life game.

The rules in this example say that a cell remains active when the number of active neighbors is 3 or 4 (*truecount* indicates the number of active neighbors) using a transport delay of 10 ms. If the cell is inactive (`(0,0) = 0`) and the neighborhood has 3 active cells, the cell activated (represented by a value of 1 in the cell). In every other case, the cell remains inactive (*t* indicates that whenever the rule is evaluated, a *True* value is returned).



```
+--------------------+ +--------------------+ +--------------------+
```

Figure 3: Executing the Life game.

Figure 3 shows an excerpt from the execution of the Life game. As we can see, studying the simulation results based on these notations can be error-prone and cumbersome, and its use for training or on-line simulation is not adequate. Instead, the provision of a graphical environment can improve the results obtained, as discussed in the following sections

3. VISUALIZATION OF 3D MODELS IN MAYA

As mentioned earlier, we decided to expand our visual environment using Open-GL based tools. Maya [Ali, 04] is a powerful application for three dimensional modeling and animation, using special effects and rendering. It allows one to create digital imagery, three dimensional animation and visual effects. The Maya software interface can be extended by providing access to the Maya Embedded Language (MEL). Maya's modeling and animation tools were used to create three-dimensional environments for Cell-DEVS and DEVS models. To do that, a team with expertise in visualization can use Maya facilities to create visual scene files, while a modeler can create advanced models of the system. CD++/Maya is an application written in MEL that provides a graphical user interface that allows CD++ log files to interact with Maya, and to visualize the corresponding model in a 3D visual environment. This instantiates a MEL script specific to a particular model, and animates the three-dimensional world (scene file) in accordance with the CD++ log file [Kha, 05]. Figure 4 shows the relationships between these procedures.

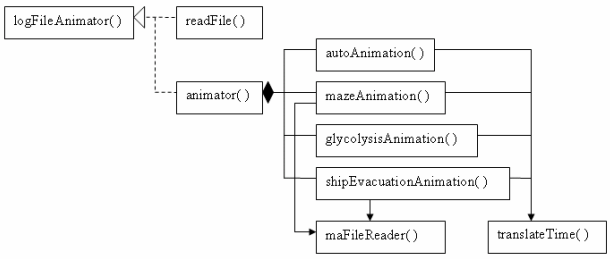


Figure 4: Architecture of CD++/Maya [Kha, 05]

This facility was first introduced in [Kha, 05]. As explained there, the *logFileAnimator* method acts as an interface requesting the user to select a particular model. The *readFile* method locates and opens the file corresponding to the file name provided and prints the contents to the Script Editor Window in Maya (allowing advanced users to analyze the detailed results found in the log files).

The *animator* method, instead, instantiates the animation procedure for that particular model, associating CD++ simulation results with graphic scenes defined in Maya. Each instance of the animation procedure opens the log File, reads it and stores pertinent information, which is then used to animate the objects in the three dimensional scene opened. All the information pertaining to a particular object from the log file is used to animate that same object in the scene file.

Finally, the *translateTime* method is in charge of accurately following the log File, and making the animation to match time with the time present in the simulation log. The *maFileReader* method is called to obtain the initial state values of each cell for Cell-DEVS models. This procedure parses the coupled model files and stores the initial state values of each cell. Then it animates the scene file for time 00:00:00:000 accordingly.

In the following sections, we will show how to use these facilities to visualize complex simulation models.

4. VISUALIZING FOREST FIRE SPREADING

Forest fires destroys important resources, hence, enormous efforts have been made to prevent them. Many forest fires models have been developed to study how the fire spreads under different environmental conditions, while one of the most popular ones is due to Rothermel [Rot, 72]. This model uses environmental and vegetation conditions to compute the ratio of spread and intensity of fire. When Rothermel's rules are applied to a given fuel model and environmental parameters, it can determine the spread ratio (i.e. the distance and direction the fire moves in a minute).

```
[ForestFire]
type : cell            dim : (20,20)
delay : inertial       border : nowrapped
neighbors : (-1,-1) (-1,0) (-1,1) (0,-1)  (0,0)
(0,1) (1,-1)  (1,0)  (1,1)
localtransition : FireBehavior

[FireBehavior]
rule : {(1,-1)+(21.552615/17.967136)}
   {(21.552615 / 17.967136)*60000} {(0,0)=0
         and 0<(1,-1)}
rule : {(1,0)+(15.24/5.106976)} {(15.24 /
      5.106976)*60000} {(0,0)=0 and 0<(1,0)}
rule : {(0,-1)+(15.24/5.106976)} {(15.24 /
     5.106976)*60000} {(0,0)=0 and 0<(0,-1)}
rule : {(-1,-1)+(21.552615/1.872060)}
          {(21.552615 / 1.872060)*60000}
              {(0,0)=0 and 0<(-1,-1)}
rule : {(1,1)+(21.552615/1.872060)}
  {(21.552615/1.872060)*60000} {(0,0)=0 and
      0<(1,1)}
rule : {(-1,0)+(15.24/1.146091)} {(15.24 /
     1.146091)*60000} {(0,0)=0 and 0<(-1,0)}
rule : {(0,1)+(15.24/1.146091)} {(15.24 /
     1.146091)*60000} {(0,0)=0 and 0<(0,1)}
rule : {(-1,1)+(21.552615/0.987474)}
{(21.552615/0.987474)*60000} {(0,0)=0 and
     0<(-1,1)}
rule : {(0,0)} 0 { t }
```

Figure 5: Rothermel's model in CD++ [Ame, 01]

In [Ame, 01], we presented the construction of a fire spreading model based on Rothermel's rules. The following figure shows the implementation of such model using CD++. This definition considers a fuel model group number 9 (based on the NFFL model, which classifies 13 different vegetation groups as fuel for fire), a SE wind of 24.135 km/h and a cell size of 15.24 x 15.24 m.

Figure 5 shows a 20 by 20 Cell-DEVS representing the terrain and vegetation. Every parameter defined corresponds to the specification of the coupled Cell-DEVS presented in section 2. In this case, the state variables use a 0 value to indicate the absence of fire and a value different to 0 indicates the time the fire has started on that cell. The rules define the behavior of the local computing function: if there is fire on a neighbor, the cell will burn. For instance, the first rule checks if the current cell is not burning (*(0,0) = 0*) and if the SW neighbor has started to burn (*0 < (1,-1)*). If this condition holds, the value will be (1,-1) + (21.552615/17.967136), which is the time the fire will start in the cell. As the spread ratio is 17.967136 mpm and a cell has a diagonal of 21.552615 m, it will take (21.552615/17.967136) minutes for the fire to reach the a cell once it has started in its SW neighbor. Therefore, we use a delay of (21.552615/17.967136)* 60000 ms after which the present cell state will spread to the neighbors.

In [Ame, 01], we showed the results of this model using a simple graphical tool, presented below.

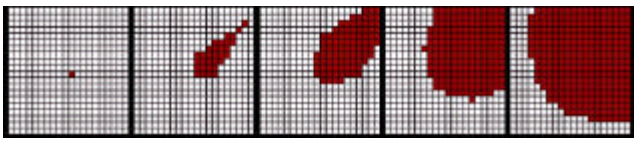
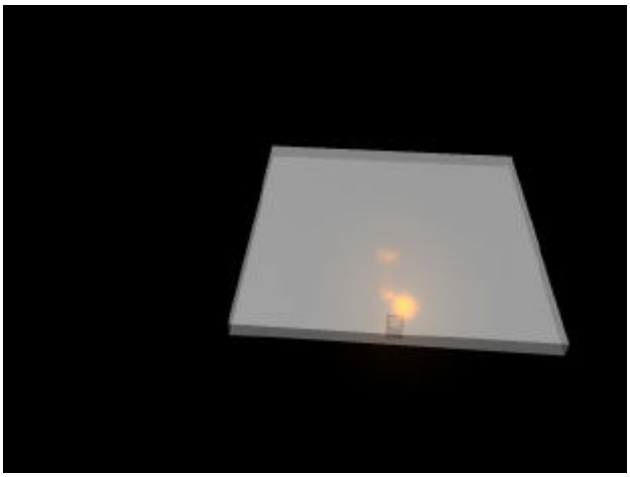
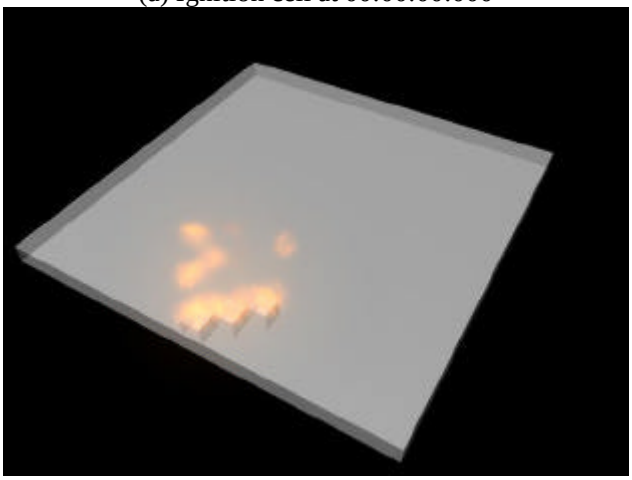


Figure 6: Execution Rothermel's CD++ [Ame, 01]

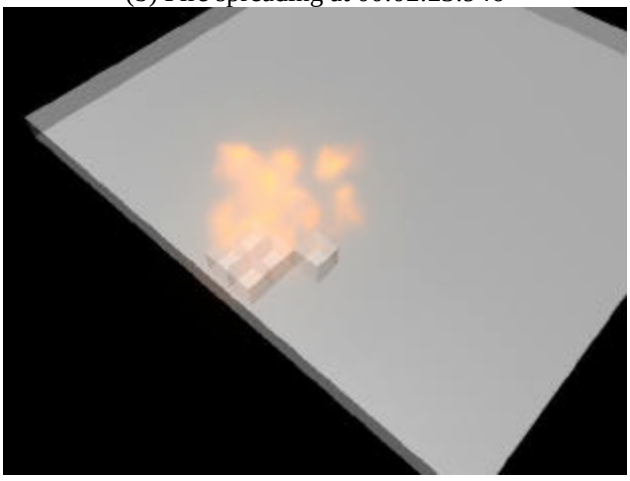
The following figure shows a 3D version of the execution results for this model using CD++/Maya. These visualizations are more realistic, and they can be easily expanded to include terrain and climate information, which could result in excellent facilities for training and online visualization under decision-making situations.



(a) Ignition cell at 00:00:00:000



(b) Fire spreading at 00:02:23:946



(c) Fire spreading at 00:02:59:049.

Figure 7: Visualization of the model in CD++/Maya

As we can see in Figure 7, at time 00:00:00:000 there is an ignition on a cell in the border. The fire spreads according to the rules defined in Figure 5, and in time 00:02:59:049 fire starts spreading in other directions, following the spread rules.

5. VISUALIZING EVACUATION PROCESSES

In [Ame, 04] we defined an evacuation model based on the rules introduced in [Wei, 04], which represents

people moving through a room or group of rooms. People try to gather their belongings or related persons and to get out through an exit door. The goal is to understand where the bottlenecks can occur, and which solutions are effective to prevent congestion.
Every person involved in the evacuation process is modeled using Cell-DEVS, with a minimum set of rules to characterize their behavior:

- Persons normally go to the closest exit.
- A person in panic goes in opposite direction.
- People move at different speeds.
- If the way is blocked, people can decide to move away and look for another way.

The main rules of evacuation model can be found in [Ame, 04] and [Wai, 05]. We used two different layers to separate the rules that govern the people moving among walls or isles from the orientation guide to an exit. Each cell in the grid represents 0.4 m^2 (one person per cell). The orientation layer contains information that serves to guide persons towards emergency exits. We assigned a potential distance to an exit to every cell of this layer. People will move in the room trying to minimize this potential, which will quickly allow them to move to the closer exit.

In Figure 8, the state value "1" represents walls or obstacles, and the state value "2" represents exits. The even state values are occupied cells and the odd ones are empty cells. Each state value also represents the shortest direction to the exit [Kha, 04; Ame, 04].

Line : 4926 - Time: 00:00:00:000

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	5	5	5	5	5	4	4	4	4	3	3	4	10	10	9	10	10	3	1
2	1	5	5	7	1	1	5	5	5	6	5	6	3	1	1	1	1	1	3	1
3	1	1	1	1	1	1	1	1	1	1	1	1	4	1	1	1	1	1	4	1
4	1	4	1	3	4	3	4	3	4	3	3	1	3	4	3	3	4	3	4	1
5	1	3	9	9	10	10	9	5	6	6	6	6	4	9	9	10	9	10	4	1
6	1	4	1	1	1	1	1	1	1	1	1	1	3	1	1	1	1	1	3	1
7	1	3	10	9	10	9	10	6	6	5	6	6	4	9	10	9	10	9	4	1
8	1	4	1	1	1	1	1	1	1	1	1	1	4	1	1	1	1	1	4	1
9	1	2	10	9	9	9	10	6	5	5	5	6	2	10	9	10	10	10	9	1
10	1	8	1	1	1	1	1	1	8	1	1	1	1	1	1	1	1	1	8	1
11	1	7	10	9	10	9	9	5	8	10	10	10	9	9	9	9	9	1	7	1
12	1	7	1	7	1	1	1	1	7	8	8	8	7	8	7	8	4	1	7	1
13	1	1	1	8	1	1	1	1	1	1	1	1	7	7	7	3	4	1	7	1
14	1	6	6	7	10	9	1	1	1	1	1	1	7	8	6	6	5	5	8	1
15	1	5	6	7	8	7	10	10	9	10	10	5	8	1	1	1	1	1	8	1
16	1	5	6	8	7	8	7	8	8	8	8	5	8	10	9	9	10	1	7	1
17	1	7	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	8	1
18	1	7	10	9	9	10	9	10	9	10	9	6	6	5	5	6	5	6	7	1
19	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 8. Orientation layer [Kha, 04; Ame, 04].

The second layer is used to describe the movement of people being evacuated. The different values used to represent the movement of people are summarized as follows:
- Next movement direction. 1:W; 2:SW; 3:S; 4:SE; 5:E; 6:NE; 7:N; 8:NW
- Speed (cells per second: 1 to 5)
- Last movement direction (similar to the next movement)
- Emotional state: the higher this value is the lower the probability that a person gets in panic and finds a wrong path.
- Number of Movement that increase the potential of a cell
- Panic Level, represent the number of cells that a person will move increasing the cell potential.

The model uses different rules define the person's behavior. We first define the path to follow using the information found on the orientation plane. The basic idea is to take the direction that decreases the potential of a cell, building a path following the lower value of the neighbors. The rules to control the people's movement in every direction. In all cases, the rule analyzes the eight near neighbors to understand what direction the person should take. The second set of rules governs the panic behavior: a person will take a wrong path or will not follow the orientation path. Figure 9 shows the simulation results for the model.

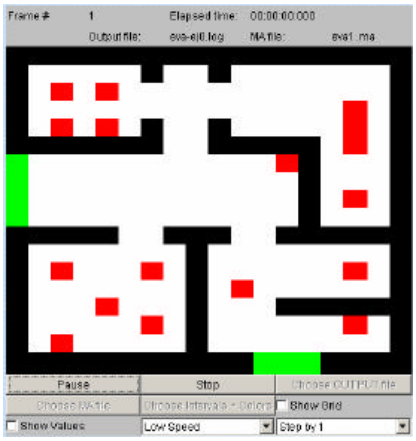


Figure 9. Simulation results [Ame, 04].

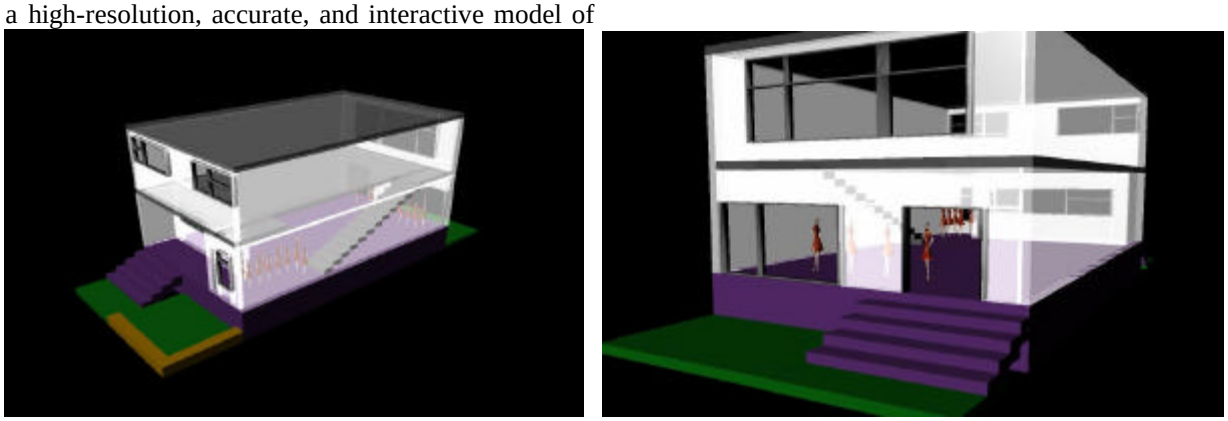
As in the previous cases, the results we can find in Figure 8 and Figure 9 are complex to interpret and understand. 3D visualization results for these problems can help in training, analysis of the evacuation problems, design upgraded solutions to improve the evacuation, while permitting online interaction with the simulation models.

In order to achieve these goals, we also defined an advanced visualization model of evacuation. Figure 10 shows different instances and visualization angles for the evacuation model. We have integrated two floors (each of them represented as explained before), interconnected through a stairway, which permits people moving from one floor to the next one.

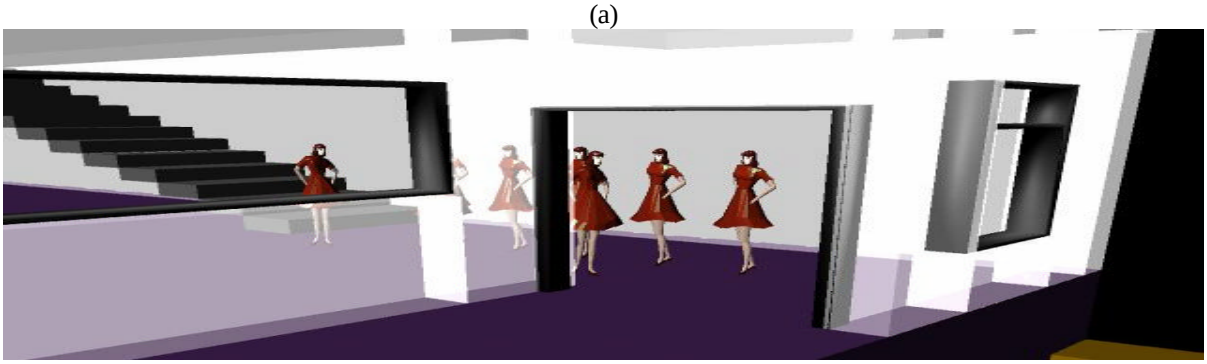
The figure illustrates the results in our 3D visualization using CD++/Maya. Initially (Figure 10 a), we show the state at time 00:00:08:000, in which people is concentrating in the two exits of the house (the figure is focused on the front door). Figure 10 b) shows a rotated version, in which we see the back door at 00:00:09:000.

Maya permits such rotations and close-ups, such as the one showed in Figure 10c). We finally see a close up view of the last person left in the building.

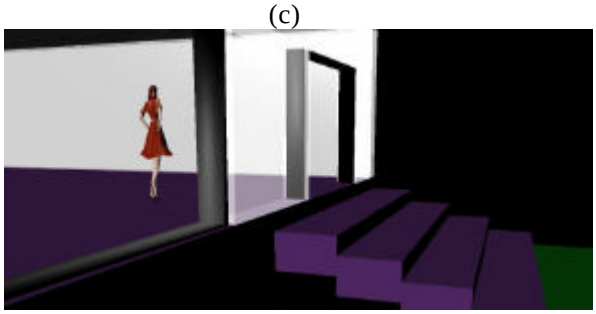
Currently, we are exploring the integration between Cell-DEVS models and advanced visualization models in Maya, like the two figures presented in the Appendix [Jem, 05]. This model represents a digital reconstruction of a high-resolution, accurate, and interactive model of the Chapel of the Convent of Our Lady of the Sacred Heart in Ottawa, Canada, using Maya for visualization. We will explore the definition of advanced behavioral models using CD++/Maya, in which we will analyze structural changes and its results under this highly advanced visual environment. Likewise, we will study advanced traffic models using exterior models like the one on the Appendix.



(a)



(c)



(d)

Figure 10. (a) View of exit1 (time: 00:00:08); (b) view of Exit2 (time: 00:00:09), (c) close-up and different angle at Exit1, (time: 00:00:08:500); (d) last person in the house (time: 00:00:08:500).

6. DEVSVIEW

Although Alias Maya is an excellent tool for creating environments and objects to visualize simulations, the installation size, workstation requirements, and licensing issues of the Maya software prevent it from being the optimal viewer for many projects. CD++/Maya can be used in larger scale applications, in which the licensing costs of the software and the hardware needs can be justified, while being able to obtain very advanced and highly precise results.

Nevertheless, we need to be able to provide alternative solutions for projects in which such costs cannot be justified. To permit increasing the potential base of users of DEVS models with advanced visualization, we developed DEVSView, an open-source visual engine also running on OpenGL.

DEVSView, allows users to create visualizations from the simulation log files outputted by CD++. DEVSView has implicit support for Cell-DEVS models and uses OpenGL and the OpenGL Utility Toolkit for hardware accelerated rendering. DEVSView provides a graphical

user interface and a text file format for the creation of visualizations.

Visualizations in DEVSView, consist of visual models that translate CD++ log files into animations. Each visual model corresponds directly to an *atomic* or *coupled* model from a CD++ simulation. These visual models contain visual states and event animations, which are used to represent the simulation graphically. The user can set up rules to trigger state changes and event animations, within the GUI or in the visualization file, and the user can use the GUI to playback the visualization.

The DEVSView visualization tool provides basic services that enable simple visualizations, including:

- Design and Implementation of a windowing system based on the OpenGL Utility Toolkit
- A windowing system with buttons, text fields, list boxes, resizable windows, and other controls necessary for a GUI. The rendering of the controls is accelerated by OpenGL.
- Visual state transition, and event animation systems: a collection of visual states and transition rules defining what simulation events trigger state changes. The event animation system is a collection of rules to define which events trigger certain animations.
- Design and Implementation of an octtree scene database to enable efficient view culling: visual models are stored in an octary space partitioning tree. This data structure recursively divides the scene extents into eight regions, which enables efficient algorithms for rendering scenes, object selection, and other frequently used scene operations.
- Flexible file format allowing backwards compatibility with older versions: the file format is a hierarchical organization of information blocks. The parsing mechanism allows a program to skip blocks that it does not recognize.
- User Interface and Functionality to load, save, modify, and playback visualizations

The following figure shows the 3D visualization results of a model representing three balls that bounce of the walls. The image shows the motion of the balls in the grid.

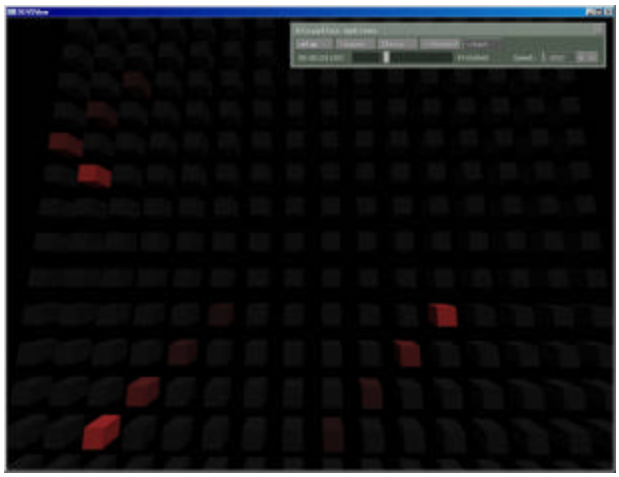


Figure 11: A bouncing ball simulation.

7. CONCLUSION

The CD++ tool allows the simulation of complex physical based on the DEVS and Cell-DEVS formalism. To facilitate the users to use the CD++ simulator, we extended its design to provide a number of services. The 3D visualization GUI enables sophisticated visualization to better understand the results. The current facilities have highly improved the use of the previously existing tools, thus enhancing the analysis experience of the modelers using the toolkit. The DEVSView tool provides facilities for creating visualizations within an open-source environment. The visual models have visual state transition systems, which define how the simulation models are graphically represented during visualization. The visual models also have event animation rules to create animations when certain events occur. Future work will include loading Maya model files for complex objects (like the figures in the Apendix), and more advanced model positioning capabilities. DEVSView could also benefit from many user interface improvements. The visualization facilities of the DEVSView tool are quite basic, but provide the beginnings of a powerful tool. The tools are open source and can be found in http://www.sce.carleton.ca/faculty/wainer.

ACKNOWLEDGMENTS

This work has been partially supported by NSERC (National Science and Engineering Research Council of Canada), the Canadian Foundation for Innovation, Ontario Innovation Trust and HP Canada.

REFERENCES

[Ali, 04] ALIAS Corp. "Maya 6 Features in Detail," [Online document accessed Oct. 2004], Available: http://www.alias.com/eng/products-services/maya/file/maya6_features_in_detail.pdf.

[Ame, 01] J. Ameghino; A. Troccoli; G. Wainer. "Modeling and simulation of complex physical systems using Cell-DEVS". In

Proceedings of 34th IEEE/SCS Annual Simulation Symposium,. Seattle, U.S.A., 2001.

[Ame, 03] J. Ameghino, E. Glinsky, G. Wainer. "Applying Cell-DEVS in models of complex systems". In *Proceedings of the 2003 SCS Summer Computer Simulation Conference*. Montreal, QC. Canada. 2003.

[Ame, 04] J. Ameghino; G. Wainer: "Application of the Cell-DEVS formalism for modeling cell spaces" In Proceedings of AIS'2004, Jeju Island, Korea, Lecture Notes in Computer Science, LNCS Vol. 3397. 2004.

[Jem, 05] M. Jemtrud et al. "An interactive model of the Chapel of the Convent of Our Lady the Sacred Heart". http://www.cims.carleton.ca/research. [Accessed: April 2005]

[Kha, 05] A. Khan, G. Wainer. "A visualization engine based on Maya for DEVS models". In *Proceedings of SISO Fall Interoperability Workshop*. San Diego, CA. U.S.A. 2005.

[Rot, 72] ROTHERMEL, R. "A mathematical model for predicting fire spread in wildland fuels". Research Paper INT-115. Ogden, UT: U.S. Department of Agriculture, Forest Service; 1972. 40 p.

[Seg, 05] Segal, M.; Akeley, K., "Open GL 2.0 spec", [Online document], Available: http://www.opengl.org/ documentation/ specs/version2.0/glspec20.pdf [accessed 2005, Apr. 22]

[Wai, 01] G. Wainer; N. Giambiasi: "Timed Cell-DEVS: modeling and simulation of cell spaces " In "Discrete Event Modeling & Simulation: Enabling Future Technologies" Springer-Verlag, 2001.

[Wai, 02] G. Wainer. "CD++: a toolkit to define discrete-event models" Software, Practice and Experience, Wiley, Vol. 32, No.3. pp. 1261-1306, November 2002.

[Wai, 03] G. Wainer and W. Chen. "A framework for remote execution and visualization of Cell-DEVS models". *Simulation*. Vol. 79, pp. 626-647. November 2003.

[Wai, 04] G. Wainer. "Modeling and simulation of complex systems with Cell-DEVS" In Proceedings of the Winter Simulation Conference, Washington, DC. IEEE Press, 2004.

[Wai, 05] G. Wainer. "On-line repository of DEVS and Cell-DEVS models". [Last checked: Apr 22, 2005.] http://www.sce.carleton.ca/faculty/wainer/wbgraf/.

[Wei, 04] J. R. Weimar. "Cellular automata model for ship evacuation" [Online document, accessed 2004, Oct. 19], Available on: http://www.jweimar.de/jcasim/schiff1.html).

[Wol, 02] Wolfram, S. "A new kind of science". Wolfram Media, Inc. 2002.

[Zei, 00] B. Zeigler; T. Kim; H. Praehofer: "Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems" Academic Press, 2000.

APPENDIX

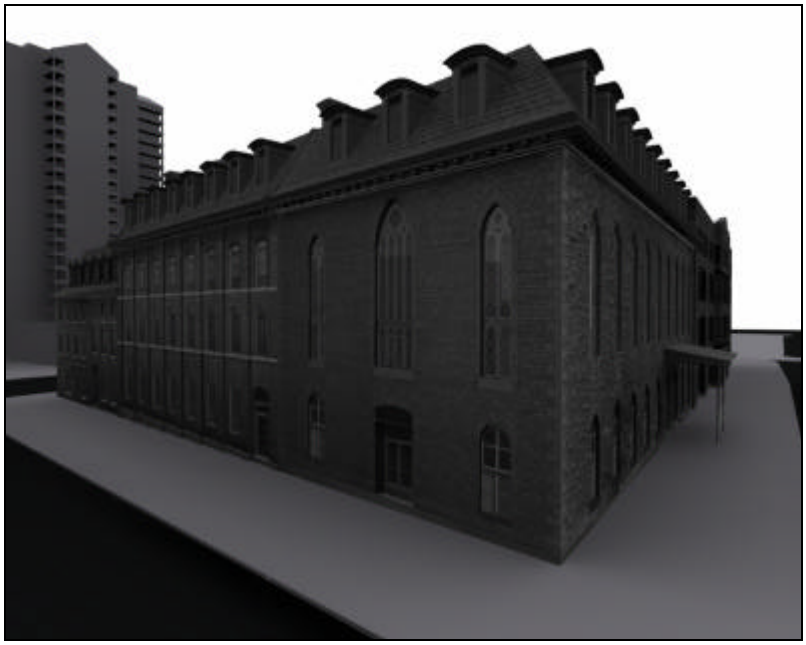


Figure 12: An interactive model of the Chapel of the Convent of Our Lady the Sacred Heart.