Aircraft Evacuation DEVS Implementation & Visualization

Patrick Castonguay & Gabriel Wainer
Department of Systems and Computer Engineering
Carleton University
1125 Colonel By Drive
Ottawa, ON K1S 5B6
pcaston3@connect.carleton.ca ; gwainer@sce.carleton.ca

Keywords: Cell-DEVS, CD++, Blender, Visualization

Abstract

Emergency evacuation from new, higher than normal, aircraft is believed to be affected by a new variable: passenger's hesitation at the exit door. This paper discusses how a previous simulation studying this effect was replicated in Cell-DEVS using the CD++ toolkit and how the results were visualized using the Blender Python interface. We will show how one can relatively easily apply advanced visualization techniques to any DEVS simulator results. As well, we noticed that the initial design of the model and simulation can limit the level of visualization which can be achieved if that design is not intended for visualization in the first place.

1. INTRODUCTION

With the advent of new twin-decks Very Large Transport Aircraft (VLTA), concerns about the speed at which they may be evacuated have emerged [1]. Many factors come into play when conducting an aircraft emergency evacuation [2, 3, 4]. This paper focuses on the possible effects passengers hesitation could have on total evacuation time when exiting from an abnormally high elevation [5]. Amos and Wood developed and applied a model of the emergency exit procedures from the Airbus A380 to study the effect of individual delays at the emergency door when exiting a plane [6].

Here, we will show the definition of these models using the Cell-DEVS methodology [7]. Cell-DEVS is an an extension to DEVS [8] (Discrete Event Systems Specifications) which enables efficient execution of cellular models. The approach extends traditional Cellular models 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++ toolkit [9] was used to implement the model and generate the initial results. Then, the Blender toolkit [10] was used to integrate the simulation results into a 3D visualization environment, based on previous experiences in this area [11]. Results from DEVS simulation can be visualized in many different ways. The CD++ toolkit provides DEVSView as well as the CD++ Modeler add-on. These tools provide a quick representation of the model and simulation behavior in the 2D space. 3D visualization of CD++ results have previously been done using VRML, Atlas, Maya and Blender [12, 13, 14]. We extended previous results in order to demonstrate the flexibility and portability of both the CD++ tool kit as well as Blender.

2. BACKGROUND

Emergency evacuations of any aircraft have to meet certain regulations; all passengers and crew have to be evacuated within 90 seconds, with only half of the emergency exits available. In [6], it was argued that, due to the dangerous and expansive nature of the evacuation, real demonstrations are normally limited to a single instance. As well, for commercial reasons, the results are normally kept secret within the company. The effect of the delay at the door was suspected to be of importance due to the nature of the second deck of the A380 which was much higher than other aircraft at the time. The model was initially implemented using NetLogo programming language [6]. Figure 1 represents the floor layout of the A380 which was used as a construct for the original work.

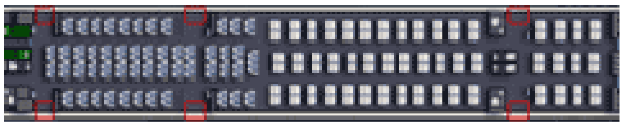

Figure 1: A380 upper floor plan

We implemented this agent based cellular automata using Cell-DEVS [7]. A cell grid representing the floor plan is defined. A single cell represents a wall, a passenger, an exit or an empty space. Each cell contains information relative to the nearest exit (*id* & *distance*). Each agent (passenger) has the following attributes:

- p_i: grid position;
- h_i: heading;
- m_i: max speed
- s_i: current spead; and
- d_i: delay at door.

The system variables needed are defined as:

D: mean passenger door delay; and

O: exit opening time (fixed at 14 seconds).

d_i is defined as a Poisson-distribution of positive real numbers with mean D. Each passenger has a minimum speed of 0 and a random (poisson-distributed) max speed between 0.3m/s and 1.05m/s. Furthermore, each cell of the grid is assumed to have a dimension of 0.5m * 0.5m.

When the emergency procedure is activated, the passengers start heading to the nearest exit at max speed. They adjust their speed to reflect the passenger directly in front of them if applicable (i.e. they do not pass, they follow in trail, but they run to catch up). We used the same ideas than the ones in the original work, where the authors only modeled the second level as they identified that inter-deck travel during emergency evacuation is not permitted. As well they only modeled passengers in their seats.

The evacuation model was implemented using a 3D Cell-DEVS model representing different variables on two different planes. The first plane is a representation of the seating arrangement. The states of the first plane are presented in table 1. The second plane is used to calculate the distance to the closest emergency exit. This is done at initialization only and states are presented in table 2. Each plane needs to access each other in order to determine if movement will be allowed to the desired cell. The speed of travel of passenger is encapsulated in the cell delay for the calculation of the new states. For the initial development of the system a speed of 1000 ms. was used as this represent a one second evaluation interval on each cell of 0.5m section and therefore a speed of 0.5m/s which fall in the average selection provided by [6]. Table 1 also shows the different colours used for 2D visualization in the CD++Modeler tool.

The second plane stores information on distances to the nearest functioning emergency exit, therefore keeping this information separate from the configuration of the aircraft.

Table 1: Dimension 1 State Description

Name	**Val**	**Color**	**Description**
Passenger	-2	Blue	Occupied (passenger)
Wall	-1	Black	Wall or Obstruction
Empty	0	White	Empty space
Exit	9	Green	Emergency exit

Error! Not a valid bookmark self-reference. illustrates the different states that a cell in the distance dimension can take and Figure 2 shows the visual representation of that dimension after system initialization for a subset of the aircraft. As the variable under study is the hesitation of passengers at the door, no other factors like panic behavior or secondary routing seeking behavior were implemented.

Table 2: Dimension 2 State Description

Name	**Val**	**Color**	**Description**
Wall	-1	Black	Obstruction
Exit	0	White	Empty space
Distance	1..∞	Green shades	Emergency exit

The system initially calculates distances from the exit on second plane for each cell. If the corresponding floor plane cell is a wall, assign distance as wall which will give information not to allow travel to this cell. One the system is initialized; each occupied cell will want to travel toward the next unoccupied cell with smaller distance to the exit. Priority is given to window seats and then to center isle. This means that for the sector defined for [port seats] the priority will be given to cells traveling down and for the [starboard exit] the priority is given to the cells traveling up. In order for a passenger to move it has to verify that other cells with lower priority will not be moving to the same wanted cell. Currently there is no panic or erratic behavior modeled and when a passenger cannot move, it patiently waits for the cell to become available. The model could be made more interesting by implementing behaviors where passengers would choose alternate routes if the one wanted is not available or have erratic movement like jumping over seats or pushing other passengers.

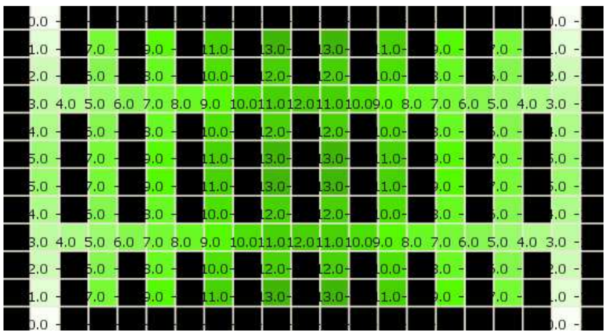

Figure 2: Distance Test plane.

Once the system is initialized, each occupied cell will want to travel toward the next unoccupied cell with smaller distance to the exit. Priority was given to window seats and then to center isle. This means that for the sector defined for [port seats] the priority will be given to cells traveling down and for the [starboard exit] the priority is given to the cells traveling up.

CD++ defines behaviors via the description of mathematical rules which are evaluated sequentially until a valid statement is found. Therefore movement priority is mplemented in the order in which the evaluation of cell state is described.

The following is the formal specification for the Cell-DEVS model:

AC_EVAC = < X, Y, I, S, θ, N, d, δ_{int}, δ_{ext}, τ, λ, D >

X = Y = Ø

S = plane 1: { 0, -1, -2, 9 }
// Empty, Wall or Obstruction, Occupied (passenger), Exit
plane 2: { -1, 0..inf }
// Wall, Distance to exit 0 being exit itself

N = neighbors :

```
% Modified Moore or VonNeuman
          (-2,0,0)
(-1,-1,0) (-1,0,0) (-1,1,0) % NW,N,NE
(0,-1,0)  (0,0,0)  (0,1,0) % W, Orig,E
(1,-1,0)  (1,0,0)  (1,1,0) % SW, S, SE
          (2,0,0)

%Distance plane to access floor plane
(-1,-1,-1)(-1,0,-1)(-1,1,-1) % NW,N,NE
(0,-1,-1)(0,0,-1)(0,1,-1) % W, Orig,E
(1,-1,-1)(1,0,-1)(1,1,-1) % SW, S, SE
```

Different Zones were defined in order to have different calculation done by the different planes as well as giving different prioritization to the different section of the aircraft. The zones are presented below and represent the distance plane as well as the port (left) side and starboard (right) side of the aircraft.

```
zone : prt_seats { (1,0,0)..(5,20,0) }
zone : stb_seats { (6,0,0)..(10,20,0)}
zone : dist_plane {(0,0,1)..(11,20,1)}
```

d = 1000 ms // default delay used to represent the speed 1 second per cell or 0.5m (or 0.5m/s)

type = transport

τ: N→S is defined by the different sets of rules introduced bellow:

The first set of rules calculates the distance to the nearest exit and stores this information in the second plane. This is only done at initialization and a zone describing the distance plane had to be defined.

```
rule: -1 0 { (0,0,-1) = -1 }
rule : 0 0 { (0,0,-1) = 9 }
rule : {(1,0,0)+1} 0 { (0,0,0)<0 and
                        (1,0,0)>= 0 }
rule : {(-1,0,0)+1} 0 { (0,0,0)<0 and
                        (-1,0,0)>=0 }
rule : {(0,1,0)+1} 0 { (0,0,0) < 0 and
                        (0,1,0) >= 0 }
rule : {(0,-1,0)+1} 0 { (0,0,0)<0 and
                        (0,-1,0)>=0 }
```

The second set of rules describes the exiting at the emergency door condition. When a cell is occupied and next to an exit the passenger will exit after a certain random delay. To be true to the initial model [6], also added a condition for the simulation time to be greater than a certain constant (14seconds).

```
rule : 0 #Macro(DoorDelay) {(0,0,0)=-2
            and (1,0,0)=9 } %exit down
rule : 0 #Macro(DoorDelay) {(0,0,0)=-2
            and (-1,0,0)=9 } %exit up
rule : 0 #Macro(DoorDelay) {(0,0,0)=-2
      and (0,1,0)=9 } %exit right
rule : 0 #Macro(DoorDelay) {(0,0,0)=-2
            and (0,-1,0)=9} %exit left
```

The DoorDelay Macro represents a random exit delay at the door and returns values (1000, 2000, 3000 or 4000). It has been defined in the *ac_eval.inc* file as such:

```
#BeginMacro(DoorDelay)
      { 1000 + 1000*randInt(4) }
#EndMacro
```

The third set of rules defines when a cell will be receiving passenger. This example is for the starboard seats and therefore the priority has been given to the cells traveling down. This will be true for the entire description of the rules. As well, for all transition, we have to verify the desired direction of movement manually. This is more complicated than having the direction of movement coded in the state but provides higher better flexibility.

```
%Entering cell
rule : -2 #Macro(Speed) {(0,0,0)=0 and
   (-1,0,0)=-2 and (-1,0,1)>=(0,0,1)}
%receive from up
rule : -2 #Macro(Speed) {(0,0,0)=0 and
       (1,0,0)=-2 and (1,0,1)>(0,0,1)}
%receive from down
rule : -2 #Macro(Speed) {(0,0,0)=0 and
    (0,-1,0)=-2 and (0,-1,1)>=(0,0,1)}
%receive from left
rule : -2 #Macro(Speed) {(0,0,0)=0 and
    (0,1,0)=-2 and (0,1,1)>(0,0,1) and
    (0,2,1)>(0,1,1)}
%receive from right
```

The fourth set of rules describes a passenger leaving a cell. The one traveling down have priority and therefore the others have to evaluate is a cell with higher priority will be moving to its position.

```
%Leaving down(has priority), up next
rule : 0 #Macro(Speed) {(0,0,0)=-2 and
    (1,0,0)=0 and (1,0,1)<=(0,0,1)}
%move down

rule : 0 #Macro(Speed) {(0,0,0)=-2 and
((-1,0,0)=0 and (-1,0,1)<(0,0,1)) and
((-2,0,0)!=-2 or (-2,0,1)<(-1,0,1))}
%move up

%Leaving right giving way to up/down
trafic but not if heading away from
ilse
rule : 0 #Macro(Speed) { #Macro
(WantRight) and ((-1,1,0)!=-2) and
(1,1,0)!=-2}
%move right with give way to down/up

rule : 0 #Macro(Speed) { #Macro
(WantRight) and (1,1,0)!=-2 and
((-1,1,0)=-2 and (-1,1,1)<(0,1,1))}
%Next row right is exit row (moving
up)

rule : 0 #Macro(Speed) { #Macro
(WantRight) and (-1,1,0)!=-2 and
((1,1,0)=-2 and (1,1,1)<(0,1,1))}
%Next row right is exit row (moving
down)
%Leaving left giving way to up/down
trafic but not if heading away from
ilse

rule : 0 #Macro(Speed) { #Macro
(WantLeft) and ((-1,-1,0)!=-2) and
(1,-1,0)!=-2}
%move left; give way to down and up

rule : 0 #Macro(Speed) { #Macro
(WantLeft) and (1,-1,0)!=-2 and
((-1,-1,0)=-2 and (-1,-1,1)<(0,-1,1))}
%Next row left is exit row (moving up)

rule : 0 #Macro(Speed) { #Macro
(WantLeft) and (-1,-1,0)!=-2 and
((1,-1,0)=-2 and (1,-1,1)<(0,-1,1))}
%Next row left is exit row (moving up)
```

3. EXPERIMENTAL RESULTS

We executed different evacuation scenarios to test the simulation model. Figure 3 shows a basic test in which all the seats are taken and the plane is evacuated in an orderly fashion. We used multiple simulation tests, and the model evolved until the basic behavior was considered satisfactory.

Table 3 shows the simulated evacuation time for different tests varying the door delay. Time for a delay of 1 second is close to the results of the original work (taking into account that the model of the plane was smaller) which expected 54 seconds and got results of 57.9 seconds with a mean deviation of 1.5 second. One can expect that if the door delay could be set as per the original experiment, similar results would be obtained. The next step was to reproduce the full size second deck of the A380 and run the same type of scenarios. The next step was to implement a full scale replica of the A380 seating plan and run simulations with half of the emergency exits disabled. The total simulation time for a full plane layout with full load was found to be greater than expected (1min 14sec). This was originally done with a delay of 1sec but was adjusted to 500msec in order to account for the propagation delay. This adjustment brought the simulation time down to 59sec which can be considered acceptable.

Table 3: Simulation Results (seconds)

Door Delay	Evacuation time
0 (no delay)	38
1	56
2	1:14
1 + 1*randInt(4)	1:04

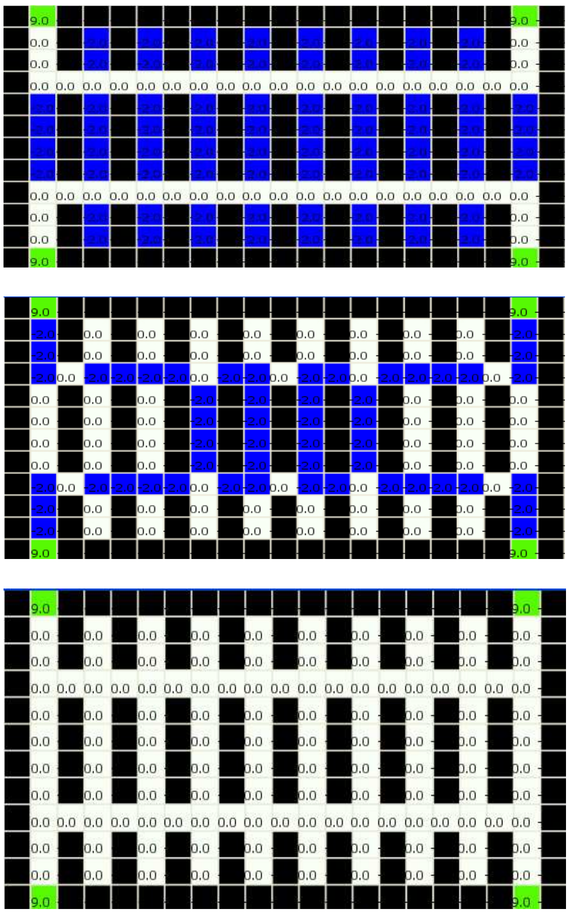
Figure 3: Test Simulation Execution

Observing the behavior of the passenger once half of the exits have been disabled, it becomes obvious that in order to have a fully representative simulation, the choice of alternate route for the passengers under certain circumstances should be added to the system. Only one freely available demonstration of such evacuation was found. It was carried out in Europe on the A380 which took 74 seconds to exit 853 passengers and 20 crew members. A video of the evacuation can be found on YouTube [15] but no other documentation was found. Of note is the interesting fact that no children, pregnant woman, elderly or obese passenger can be seen in the video. This is of course to be expected as the airline company would not want to endanger the life of these people, but strengthens the argument that simulation can give a more complete representation of these situations.

4. 3D VISUALIZATION OF THE EVACUATION PROCESS

We created a 3D visualization model based on previous work, where we used Blender and Maya for creating advanced visualization engines [8]. Blender is a free open source software 3D content creation suite released under GPL [7]. It can create 2D or 3D graphics as well as movie quality animations. Blender supports Python scripting, is free, and provides powerful 3D animation capability. Python provides a high-level dynamic object-oriented programming language. For this project we have not made any modifications to Blender itself but have used its native support for Python scripting in order to read and animate the results from CD++. Figure 4 shows the architecture of the visualization engine. A Python Script in Blender is in charge of integrating the visualization engine and the CD++ simulation results. CD++ uses a model (.MA) file to define the behavior of Cell-DEVS models. In turn this model file can be given varied initialization data (using a .VAL file) to set the initial values of each cell of the model in the simulation.

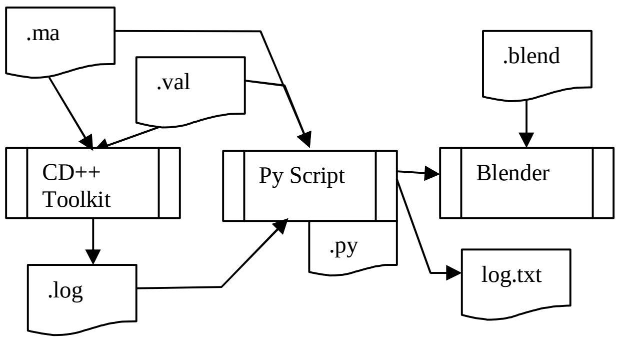

Figure 4: Visualization Architecture

These two files are used at run-time to define the behavior of the simulation and a .LOG file is generated containing the state of each cell for every time-advance for the simulation. In order for the script to be most flexible, it had to be extended to support the existence of a .VAL file and login of the script behavior itself. The cell space is represented in Blender by a direct coordinate transformation of the cell position to a coordinate inside Blender. Therefore the size of the passengers, walls and seats have to be no more than 1x1, this can be controlled by scaling the 3D model with the grid shown in Blender.

Once loaded, the script will read and parse the .MA file and look for a .VAL file if needed. The .VAL file will be processed before the .LOG file in order to adequately initialize the simulation behavior. Figure 5 shows the visual representation of our aircraft evacuation simulation after initialization (a full airplane), which represents the same initialization case than the one showed earlier in Figure 3. As Blender provides a Python API that covers most of the functionality of the Blender GUI, the script only has to calculate time and position from the log file provided by the Cell-DEVS simulation in order to control the display of the entities. When parsing the .LOG file, passengers will be displayed at the appropriate coordinate within Blender. Passengers will appear and disappear is sequence representing their movement.

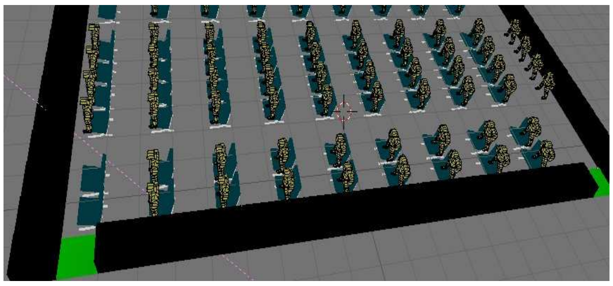
Figure 5: System Initialization [15].

A capture of the passengers exiting the airplane in a queue and waiting for the passenger in front of them to move is shown in Figure 6.

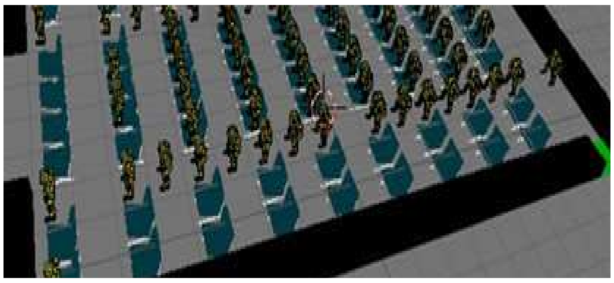
Figure 6: Passenger In Queue

With the basic 3D objects already defined in a .blend file, the script simply has to duplicate, link and unlink specific objects to the scene, in accordance to the cells state, in order to control the visual flow of the simulation. This is shown with the code excerpt that follows.

```
if (logValue == -2): # Occupied cell
try:
  scn.objects.link(ob)
except:
   human = Blender.Object.Get('Human')
   Blender.Object.Duplicate()
   …set position
elif (logValue == 0): # Empty cell
   scn.objects.unlink(ob)
```

The simulation will execute and passengers will appear and disappear representing their movements until they have all exited the aircraft. Figure 7 shows the four basic 3D model elements that were used in the visualization of our results (from left to right): a chair; an emergency exit; a wall or obstacle; and a passenger. In the execution of our simulation on the passengers are dynamic.

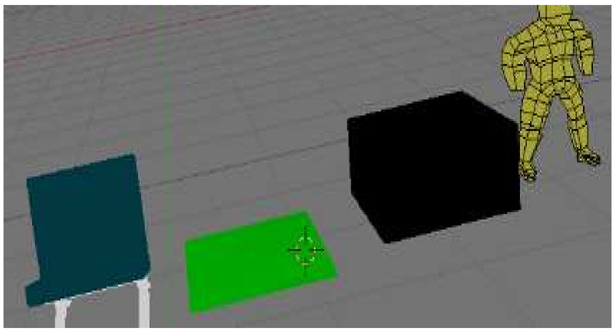
Figure 7: 3D Models

When the model was first developed, 3D animation was not something the author had in mind. This led to some decisions at the atomic model level that were later considered limiting when developing the 3D animation script. Having the state of each cell on the first dimension contain limited information (empty, passenger, wall) led to all motion information being calculated by the rules within CD++. This increased the complexity of representing the passengers' movement in a smooth transition fashion within the Blender environment. Having movement information such as direction and speed available directly within the state of model is considered important when doing 3D animation in Blender but is not needed for 2D representation inside CD++. This can be seen in Figure 8 where the passengers travelling to the back of the airplane are actually facing forwards. The fact that the basic CD++ toolkit keeps each of the cells independent from one another and messages cannot be passed between them was at the root of some of the limitations motioned previously. There exists an advanced CD++ toolkit that is more object-oriented and permits these types of complexes cellular behaviors. We believe that using this advanced framework could permit a more complete and complex representation of the behavior of passengers inside an aircraft during an emergency evacuation scenario.

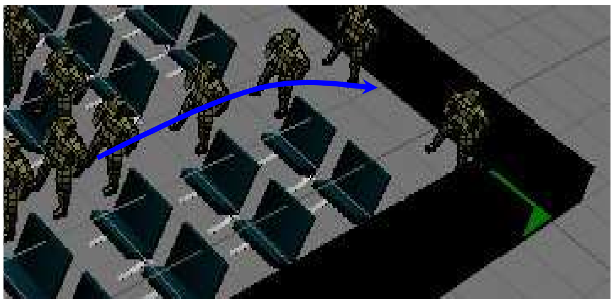
Figure 8: Passengers Travelling Backwards

Another important limitation of the current Blender visualization script is that actual rendering of the scenes as an animation was not possible. During the execution of the script, one can see the passengers moving inside of the Blender application. Blender allows users to render these frames to an animation that could be played outside of the application but only the last state of the system is represented with the passengers. All the other frames contain only the walls, chairs and exits (static items). We believe this to be due to the way the Python script allocates the passenger entities to the scene. An example of a single rendered frame (normally generated as a .jpeg image) is presented at Figure 9. This was achieved by cheating the script and providing an incomplete CD++ .LOG file leaving some passengers in the aircraft.

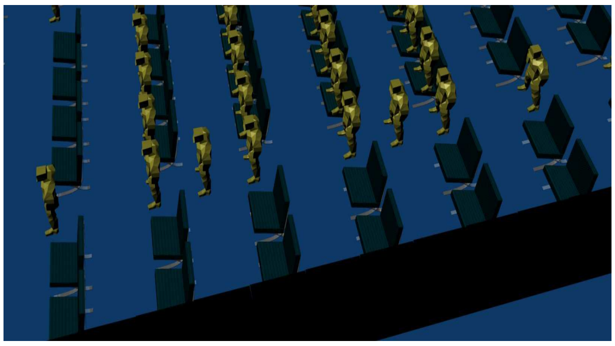

Figure 9: Rendered Frame

5. CONCLUSION

This project expanded on a previous implementation of a CD++ Python script generating 3D animations inside Blender. We effectively proved that with little or no expertise in the field of 3D animation, one can easily adapt a generic script to support visualization of results from a specific CD++ simulation. This was possible only due to the flexibility and portability of the CD++ toolkit, Python and Blender. Further interesting developments in this area would be to extend the script to enable mapping of the stateID to the 3D model so that it can be used to visualize results from CD++ without having to modify the script itself every time. Another interesting area to research would be the automatic generation of a rendered animation instead of having to view the raw animation processing inside of Blender.

Some assumptions and limitations were taken into consideration: no passenger behavior other than seeking the closest emergency exit was implemented. In order to broaden the scope and applicability of this project, one would need to implement behaviors such as the search of alternate valid routing and passenger panic leading to erratic movement. Also because of the original structure of the state of each cell, a maximum speed assigned to each passenger was not implemented. This would be needed in order to simulate having wide types of passenger such as elderly, obese, children and/or pregnant women. A possible solution would be to have a third dimension that keeps the max speed information or to have a state on the first dimension encode this information. Another limitation identified is that even though a random generation functions were used, the outcome was deterministic in that every passenger had a different delay at the exit but they were identical between all runs. This may have been caused by the way random generations of variables in handled in CD++. In order to make this simulation more stochastic, a way to change the seed of the random generation function would be needed within the CD++ toolkit.

References

[1] Muir H. & Thomas L. "Passenger Safety and Very Large Transportation Aircraft". Aircraft Engineering and Aerospace Technology 76(5), 479-486. 2004.

[2] Muir H.C., Bottomley DM & Marrison C. "Effects of Motivation and Cabin Configuration on Emergency Aircraft Evacuation Behavior and Rates of Egress". International Journal of Aviation Psychology 6(1), 57-77. 1996.

[3] Galea E.R., Owen M. & Lawrence P.J. "Computer Modeling of Human Behavior in Aircraft Fire Accidents". Toxicology 115, 63-78. 1996.

[4] Galea E.R. & Perez Galparsoro J.M. "A Computer-Based Simulation Model for the Prediction of Evacuation from Mass-Transport Vehicles". Fire Safety Journal 22(4), 341-366. 1994.

[5] Jungermann H. & Gohlert C. "Emergency Evacuation from Double-Deck Aircraft". Proceedings of ESREL 2000, 989-982, Rotterdam. 2000.

[6] Amos M. & Woods A. "Effect of Door Delay on Aircraft Evacuation time". Department of Computer Science, University of Exeter, Harrison Building, North Park Road, Exeter EX4 4QF, UK. 2005.

[7] Wainer, G. “Discrete-Event Modeling and Simulation: a Practitioner’s approach”. Taylor and Francis. 2009.

[8] B. Zeigler; T. Kim; H. Praehofer: "Theory of Modeling and Simulation: Integrating Discrete Event and

Continuous Complex Dynamic Systems". Academic Press, 2000.

[9] Wainer G. "CD++: a Toolkit to Define Discrete-Event Models". Software, Practice and Experience. 32(3), 1261-1306. November 2002.

[10] Blender Foundation, http://www.blender.org/. Accessed January 2009.

[11] Wainer G., Poliakov E., Hayes J. & Jemtrud M. "A Busy Day at the SAT Building". Proceedings of the International Modeling and Simulation Multiconference Buenos Aires, Argentina. 2007.

[12] Wainer G. & Chen W. "A Framework for Remote Execution and Visualization of Cell-DEVS Models". Simulation. 79, 626-647. November 2003.

[13] Wainer, G. & Liu, Q. "Tools for Graphical Specification and Visualization of DEVS Models". Accepted for publication in *Simulation, Transactions of the SCS* (accepted: October 2008)

[14] http://www.youtube.com/watch?v=XIaovi1JWyY. A380 Emergency Evacuation test. Accessed: January 2009.

[15] https://sourceforge.net/projects/devsacevacsim/. DEVS Aircraft Evacuation Sim. Accessed: January 2009.

Biographies

PATRICK CASTONGUAY is a student in the Department of Systems and Computer Engineering at Carleton University where he is pursuing a M.A.Sc. in Technology Innovation Management with specialization in Modeling and Simulation. Patrick graduated in 1998 from the Royal Military College of Canada with a Bachelor in Software Engineering. He then worked until 2000 as a team lead for CAE, maintaining and developing the CF-18 fighter aircraft mission computer software. He then specialized in Navigation and Communication with the Canadian Air Force and was employed with the CP-140 maritime patrol aircraft for six years. During that time he helped develop, maintain and administer the squadron scheduling and proficiency-management database.

GABRIEL WAINER received the M.Sc. (1993) and Ph.D. degrees (1998, with highest honors) of the University of Buenos Aires, Argentina, and Université d'Aix-Marseille III, France. In July 2000, he joined the Department of Systems and Computer Engineering, Carleton University (Ottawa, ON, Canada), where he is now an Associate Professor. He has held positions at the Computer Science Department of the University of Buenos Aires, and visiting positions in numerous places, including the University of Arizona, LSIS (CNRS), University of Nice and INRIA Sophia-Antipolis (France). He is author of three books and over 190 research articles, and helped organizing over 70 conferences. He is Special Issues Editor of the Transactions of the SCS, and the International Journal of Simulation and Process Modeling. He was a member of the Board of Directors of the SCS, a chairman of the DEVS standardization study group (SISO). He is Director of the Ottawa Center of The McLeod Institute of Simulation Sciences and chair of the Ottawa M&SNet, and one of the investigators in Carleton Unversity Centre for advanced Simulation and Visualization (V-Sim).