Computational Fluid Dynamic Cell-DEVS M&S of Coronary Heart Disease

Michael Van Schyndel Gabriel Wainer
System & Computer Engineering
Carleton University
Ottawa, ON, Canada

Azam Khan Rhys Goldstein
2Autodesk Research
210 King St. East, Toronto
ON M5A 1J7, Canada

Keywords
Computational Fluid Dynamics, Cell-DEVS, Cellular Models

Abstract
Computational Fluid Dynamics (CFD) deals with computing the equations of fluid flows using numerical methods instead of partial differential equations. The Discrete-Event System specification (DEVS) theory has already been used to approximate various continuous systems by applying a quantized state system approach. In this research, we experimented with a method based on Cell-DEVS theory and CFD, building a uniform set of rules for to apply to each cell and execute the state changes of the cells asynchronously. We show how this harmonized set of state changes can effectively render the dynamics of the fluid. To do so, we show an application of the model in the narrowing of the coronary arteries due to plaque buildup.

1. INTRODUCTION
Computational Fluid Dynamics (CFD) solving is the process for calculating and describing the physics of the movement and interaction of fluid flow with the use of numerical methods [1]. Currently there exist no analytical solution; however, various numerical methods have been proposed, including Cellular Automata (CA) [2]. The behavior of the motion is defined with the Navier-Stokes equations, which are a representation of Newton's Second Law of motion. These cells are solved for discrete durations of time and the results rendered to provide results that are more meaningful.

Cell-DEVS is a derivative of the DEVS formalism that implements CA. The cells execute, which reduces unnecessary processing burden, with a continuous time base. Each cell is treated as a DEVS atomic model [4] were the state changes are event driven. Cell-DEVS was originally introduced for modeling and simulation of spatial systems however, there has been no research on adopting it for CFD. In n this research, we propose using the Cell-DEVS methodology to implement CFD equations to simulate fluid dynamics. The rule-based nature of cellular model behavior definition provides a platform for area-wise behavior definition, leading to easier and faster adoption and implementation of CFD solver algorithms. Additionally, simulating with DEVS allows interfacing CFD models with other models defined in different formalisms with ease.

The research and solver presented here are an improved derivative of the solver presented in [5]. By using a new solver and implementing a new strategy for defining the rules, it was possible to increase the computational efficiency drastically. We will discuss the framework that can be used in a real-case application about Coronary Artery Disease (CAD), which is trying to narrow the coronary arteries due to plaque buildup will affect the flow of blood to the heart muscles.

2. RELATED WORK
Fluid dynamic solvers have been used for a wide variety of purposes. The goal is to create a realistic representation of a naturally occurring fluid system such as rising smoke or blowing dust. The flow of fluids can be viewed as solid particles interacting with a velocity field, or as a density of particles. There are different methods for solving the evolution of these fields and densities, such as, Lattice-Gas [6], Navier-Stokes Equations [7] and Riemann Solvers [8]. In [9], Stam proposed a new method of resolving the Navier-Stokes equations. A cell lattice is spanned over the simulation window with each cell holding unique information regarding that particular area. Each cell stores a density value and the horizontal and vertical components of velocity. The cell spaces are updated simultaneously at discrete time intervals. This algorithm provided realistic results with limited computational effort by utilizing a rather basic set of rules.

We are interested in adapting the algorithms presented by Stam [9] and to use the models to define a discrete-event CFD solver developed according to the conventions of the DEVS and Cell-DEVS formalism. DEVS [1] has several characteristics which aide in the development of such simulations. The general nature of the specifications of DEVS allows it to be used in a wide range of simulation methods [2] (parallel, distributed, real-time). DEVS provides a hierarchal approach for coupling models to create larger, complex models with different methods. DEVS separates the model code, which contains the real-world system, from the simulator and the time advance functions, which allows models to be coupled that have different time advance functions. This makes the DEVS formalism a powerful tool for simulating large complex systems, such as biological systems.

A Cell-DEVS is an extension defined as a lattice of cells holding state variables and a computing apparatus, which is in charge of updating the cell states according to a local rule. This is done using the current cell state and those of a finite set of nearby cells (called its neighborhood), as done in CA. Each cell is defined as a DEVS atomic model, and it can be later integrated into a coupled model representing the cell space (Figure 1).

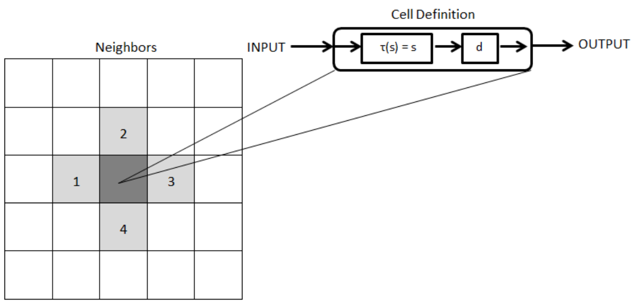

Figure 1. Cell-DEVS Model.

Each cell of the lattice contains information regarding its neighborhood and its local computing function. This local computing function has 3 main parts; a PostCondition, a Delay and a PreCondition. When defining the local computing function, we define the PreCondition that must be satisfied so that the PostCondition will be applied to that cell, and after the Delay has expired, that value is transmitted to other cells in the neighborhood (or to other DEVS models). Cell-DEVS allows for the cells to be calculated asynchronously and then updated all at once. This feature allows for the possibility of parallel computing.

The CD++ software [4] provides a development environment to create and navigate through the process of Modeling and Simulation (M&S) of a Cell-DEVS model. CD++ is an open-source framework that has been used to model environmental, biological, physical and chemical models as well as many other real-life simulations. The toolkit includes a high-level scripting language keyed to Cell-DEVS, a simulation engine, a testing interface and a basic 2D and 3D graphical interfaces. The following code is a sample of the implementation of a Cell-DEVS coupled model in CD++.

```
[cfd]
type : cell    width : 75   height : 25
delay : transport      border : wrapped
neighbors : cfd(-1,-1) cfd(-1,0) cfd(-1,1)
neighbors : cfd(0,-1)  cfd(0,0)  cfd(0,1)
neighbors : cfd(1,-1)  cfd(1,0)  cfd(1,1)
localtransition : Navier-Stokes
neighborports : value diffusion u v boundary p div
```

In this example, we see the definition of a Cell-DEVS coupled model named *cfd*. We define the model's dimensions, and the number of cells per dimension (in this example, two dimensions, 75x25). We then determine the behavior of the boundaries (border). In this example, the model is continuous along all the boundaries (*wrapped*). The other option is to have unwrapped borders, in which case special rules must be used for the cells in the borders. During the evolution of a model, the values used in the local computing function are taken from the defined set of surrounding (the neighborhood), defined next.

The user can employ a number of state variables in each cell, and input/output ports in the cell. Variables are useful for storing information locally, to be accessed by the cells during the local computing function calculations; however, the information stored within them could only be accessed by the cell to which it was linked and not by any neighboring cells. I/O ports allow transmitting information to any of the neighboring cells.

At this point, all that remains is defining the local computing function, that is, the set of rules located in each cell that governs cell behavior and determines the state changes. The following is an example of such a rule:

```
rule : { ~value := if((0,-1)~value = 1, 0,1); }
       100 { (0,0)~value = 1 }
```

Here the **PreCondition** is that the state of the *value* port must be equal to one. If this is true then the state of the *value* port is equal to zero if the neighboring state is one else it would retain the state value of one. This is the **PostCondition**. Finally, the **delay** is set; in this case a delay of 100ms was chosen. In the following section, we will introduce the definition of our CFD model, and will discuss the model implementation and simulation results.

3. MODEL DEFINITION

The Navier-Stokes equations, named after Claude-Louis Navier and George Gabriel Stokes, make use of Newton's Second law by applying it to fluid flow, assuming that the stress on the fluid is proportional to the diffusing viscous and the pressure terms [9].

$$\frac{\Delta u}{\Delta t} = -(u.\nabla)u + v\nabla^2 u + f$$
$$\frac{\Delta p}{\Delta t} = -(u.\nabla)p + k\nabla^2 p + f$$

Equation 1. Velocity and Density Equations

The first equation is for solving the velocity vectors; the sum of which is hereafter referred to as the velocity field. The equation is a re-arrangement of the incompressible flow of Newtonian fluids. The acceleration ($\frac{\Delta u}{\Delta t}$) is equal to the sum of the negative continuity equation $(u.\nabla)u$, responsible for the conservation of mass),the viscosity ($v\nabla^2 u$) and any body forces present (*f*). In other words, the change in the evolution of the velocity field is based on the viscosity and any other forces that may act upon it (such as a heating vent). While this is the most important part of any good CFD solver, it provides very little visually. To make it more useful, we must demonstrate particles moving through the velocity fields. To move objects, we must simply determine what forces are going to be acting on it, and in what direction. These forces are extracted from the velocity fields. Most of the objects we wish to move are relatively light, such as dust or smoke [9]. One could simply apply these forces to the particles, and see how they move; however, for more complex models, it would be taxing to perform these calculations for a large number of particles. Instead, we could treat the matter as a density of particles, where instead of either being 0 or 1 (no particle, particle respectively), we would treat it as a gradient value that ranges from no particles present to some maximum number of particles present. The forces on these densities are applied using the second equation, which is similar to the equation used for evolving velocities, but more simplified since the only forces present are solely generated from the velocity vector field.

The algorithm was broken into two parts: A *density solver* and a *velocity solver*. Each section represented one of the Navier-Stoker equations.

The diffusion function is responsible for calculating the natural flow of the particles regardless of the forces exerted by the velocity fields. The density for the cell is calculated as the sum of the densities not exiting the cell to the surrounding area and the densities entering the cell from its neighboring cells, as seen in Equation 2.

$$\boldsymbol{x(i,j)} = \frac{\left[\boldsymbol{x(i,j)'} + \mathbf{a} * \left(\boldsymbol{x(i-1,j)} + \boldsymbol{x(i+1,j)} + \boldsymbol{x(i,j+1)} + \boldsymbol{x(i,j-1)}\right)\right]}{\boldsymbol{1+4a}}$$

Equation 2. Diffusion Calculation [9].

The rate at which the densities radiate between cells is referred to as the viscosity and is incorporated into the equation as the value for **a**. By incorporating the viscosity into the equation, it is possible to simulate particles with different behaviors. A low viscosity would cause the densities to have very little diffusion, similar to a liquid; while a high viscosity would rapidly radiate to the surrounding cells and take the shape of the container, which is similar to the behavior of a gas.

The advection function's role is to apply the forces generated by the velocity fields. The force acting on the density at any location is equal to the equivalent velocity vector of *u* and *v*. To apply the forces is significantly more complicated. The simplest approach would be to determine the destination based on the magnitude of the forces applied. However, since the system is treated as a cell space and not all densities will end up in the exact center of the cell after moving, this would cause problems. Instead, to move the density, one simply traces backwards from the cell center to compute where the density.

The diffusion function is responsible for calculating the natural flow of the particles regardless of the forces exerted by the velocity fields. This natural flow, or diffusion, is represented in the Navier-Stokes equations as the radiation term. In addition to resolving the radiation of the densities, it is also responsible for resolving the radiation of the velocity field. The implementation of equation 2 with the Cell-DEVS formalism is straightforward. The *x(i,j)* term is replaced with the port for which the state value is calculated. For example when the values stored within the *diffusion* port are being diffused, the *x(i,j)* term is replaced by (0,0)~diffusion, similarly the *x(i±1,j±1)* terms are replaced by the corresponding neighbors' *diffusion* port values. Finally, the *x'(i,j)* is replaced with the state value stored in the *value* port.

To calculate the new state value for the cell we must first determine two factors: the particles leaving the cell and the particles entering the cell from the immediately adjacent cells. Therefore the new cell value will be equal to the previous state value minus the densities leaving to the surrounding cells plus the particles entering the cell, as seen in equation 3.

$$D_{NEW}(i,j) = D(i,j) - D_{Leaving}(i,j) + D_{Entering}(i \pm 1, j \pm 1)$$

Equation 3. Calculating the new density values

After calculating the diffusion, we invoke an *advection* function whose purpose is to "move" the densities and velocity fields. This movement generated is caused by either the forces acting upon the density field from the velocity fields, or, in the case of the velocity solver, the momentum of the velocity fields. The velocity field is composed of velocity vectors which are stored in component form, hereafter referred to as velocity component vectors where *u* represents the horizontal component and *v* represents the vertical component. In other words, the advection function resolves the last term of the Navier-Stokes equations. While there are many methods for which the forces could be interpreted to resulting movements, the method used in this thesis is stable and works with the Cell-DEVS.

The approach to moving densities is to determine the densities entering the cell instead of where the densities currently in the cell will end up. To implement this process we must first ensure that the origin of the densities lie within the defined neighborhood.

If we can ensure that the velocity vectors will remain within a set range we can define a neighborhood with 100% certainty that the displacements will not exceed the neighborhood boundaries. In our model, the maximum absolute value of the magnitudes for the velocities is set to be one, therefore the neighborhood was defined as the 8 cells surrounding the cell being computed (called a Moore's Neighborhood). The magnitude of the velocity component vectors was limited to one for several reasons. First, if the modeler is not careful, large movements can cause instability in the model. The backwards tracing method is supposed to lead to a more stable model. Second, larger velocities would require a larger neighborhood; this would result in the need for additional cases and overall increase the computational effort.

The probability that the location in which the densities originated is at the cell center is rare. For this reason, the densities that will be "moved" to the new cell location will most likely come from 1 or more cells. Therefore, the new density state value is calculated as a weighted average of the four closest cells to the origin. For example, if we assume the velocity component vectors at the current cell are u = 0.6 and v = 0.4, the cell from which the densities originate from would be at (-1,1) (Figure 2).

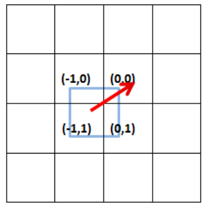

Figure 2. Results of the advection calculation

The amount of the densities coming from each cell is proportional to the area of each cell enclosed by the square. The following equation is used to determine these amounts:

$$D(0{,}0) = \left[\, u \times \left(\, v \times d'(-1{,}0) + (1-v) \times d'(-1{,}1)\right) + (1-u) \times \left(\, v \times d'(0{,}0) + (1-v) \times d'(0{,}1)\right)\right]$$

Equation 4. Weighted averages for new density

Using the values previously described the results would be as follows: 36% of the density originates from the cell location (-1,1), while 24% from both (-1,0) and (0,1) cells and the remaining 16% comes from (0,1) cell; this totals to 100% therefore the formula is mass conserving.

As previously mentioned, the magnitudes for the velocity component vectors were restricted to fall between 1 and -1. These forces would translate to the maximum distance traveled to the cell being 1. This means there are four different combinations for which the velocity components could be. Therefore, four versions of equation 4 must be defined to represent the possible outcomes.

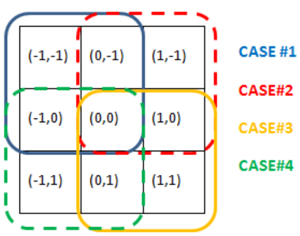

Figure 3. Possible sources of the density for advection

The first case is for when the velocity component vectors are between $0 \leq u \geq 1$ and $0 \leq v \geq 1$:

$$D(0{,}0) = \left[\, u \times \left(\, v \times d'(-1,-1) + (1-v) \times d'(-1{,}0)\right) + (1-u) \times \left(\, v \times d'(0,-1) + (1-v) \times d'(0{,}0)\right)\right]$$

Equation 5. Weighted Average for new density Case #1

The second case is for when the velocity vectors are between; $-1 < u > 0$ and $0 < v > 1$:

$$D(0{,}0) = \left[\, |u| \times \left(\, v \times d'(0,-1) + (1-v) \times d'(0{,}0)\right) + (1-|u|) \times \left(\, v \times d'(1,-1) + (1-v) \times d'(1{,}0)\right)\right]$$

Equation 6. Weighted Average for new density Case #2

The third case is for when the velocity vectors are between; $-1 \leq u \geq 0$ and $-1 \leq v \geq 0$:

$$D(0{,}0) = \left[\, |u| \times \left(\, |v| \times d'(0{,}0) + (1-|v|) \times d'(0{,}1)\right) + (1-|u|) \times \left(\, |v| \times d'(1{,}0) + (1-|v|) \times d'(1{,}1)\right)\right]$$

Equation 7. Weighted Average for new density Case #3

The final case, case 4, is for when the velocity vectors are between; $0 < u > 1$ and $-1 < v > 0$:

$$D(0{,}0) = \left[\, u \times \left(|v| \times d'(-1{,}1) + (1-|v|) \times d'(-1{,}0)\right) + (1-u) \times \left(\, |v| \times d'(0{,}1) + (1-|v|) \times d'(0{,}0)\right)\right]$$

Equation 8. Weighted Average for new density Case #4

The advection function is therefore represented by these 4 equations. These same equations are used for moving the velocity vectors as well, with the only difference being that instead of density values being used, the magnitudes of the velocity vectors are being averaged.

After the advection calculation, the *projection* function is responsible for calculating the first term of the Navier-Stokes equation. The main role of that term is to ensure the solution to the equation remains mass-conserving. The projection function is the most complex function of the entire model and therefore, the information generated during the execution is stored in two separate ports, while the remaining information from the function is stored within the two vector component ports. These two parts are hereafter referred to as the *div* and *p* functions. The *div* function is responsible for creating a gradient map. A gradient map shows changes in the velocity fields, with small values representing a uniform field with little variation and large values representing extreme fluctuations in the velocity field. To ensure the system remains stable, i.e. mass is not lost or created, we want to ensure that situations do not arise were the velocity vectors all converge to a point or diverge from a point, as seen in figure 4.

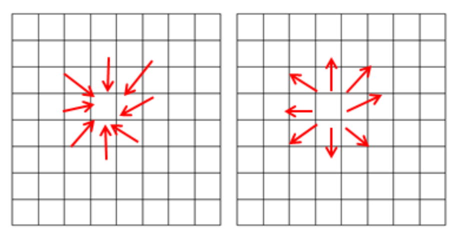

Figure 4. Convergent and Divergent Velocity Fields

Figure 4 shows a case of convergence and divergence, which could cause instability. For example, while the densities can theoretically exceed a value of 1 without causing instability this may not be a desired outcome. More importantly though is this may cause the magnitude of the velocity component vectors to exceed 1 at this location and that is not allowed. Similarly, with the divergent case, were all the velocities are leading away from the cell, the density at the center cell would follow each velocity vector without being divided amongst all of the vectors. This would result in more mass leaving the cell then was actually present in the cell, once again leading to instability. The equation for generating the gradient map is as follow:

$$div[i,j] = -0.5 * 1/h * (\, u(i+1,j) - u(i-1,j) + v(i,j+1) - v(i,j-1)\,)$$

Equation 9. Generating the gradient field

As we can see, the gradient field is calculated as the sum of the differences of the vertical and horizontal neighbors. Therefore, a small gradient value will occur when the change in values of the neighborhood is small. Large gradient values will occur when the values of the velocities make extreme changes but still maintained the same sign. The largest, and worst case, is when the values are opposite signs on either side of the cell. Once we have obtained the gradient field, the p function is executed. Similar to the *diffusion* function it helps average the values out, reducing large gradient changes. The equation for the *p* function is as follows:

$$p(i,j) = [\, div(i,j) + p(i-1,j) + p(i+1,j) + p(i,j-1) + p(i,j+1)\,]/4$$

Equation 10. Averaging the gradient field

This equation is run for several iterations. The final step is to subtract the gradient field from the velocity field. The result will be a more stable field, void of any situations such as those shown in figure 4. The following two equations represent the gradient field being subtracted from the velocity field.

$$u(i,j) = (\, u'(i,j) - 0.5 * h * \big(\, p(i+1,j) - p(i-1,j)\big)$$
$$v(i,j) = (\, v'(i,j) - 0.5 * h * \big(\, p(i,j+1) - p(i,j-1)\big)$$

Equation 11. Subtracting the gradient field from the horizontal and from the vertical fields

Again, for small changes in the velocity field the *p* values will be equal and therefore the final velocity field will not be changed. Even for large differences that occur between velocities of the same sign, the *p* value will have been averaged such that the final difference in the *p* values will not be that large, and will not affect the final field significantly.

The *projection* function is repeated twice for the velocity solver in order to ensure that changes made to the velocity field during the *diffusion* or *advection* functions do not cause any convergent or divergent behaviors in the velocity field. For example, if two "fronts" are about to collide, the *diffusion* process could bring them close enough together. This would cause problems to occur during the *advection* process. Therefore the *projection* function is used to help negate this. However, this would only effect the leading edges of these "fronts" and once again when advected the fronts may come near enough that they are convergent. During the *diffusion* process these fronts would be averaged and the result could be that they cancel each other out. This is not the desired effect; therefore, we call the *projection* function once more to ensure this does not occur.

After, we execute the *boundary* function is to define the behavior of fluid-boundary interactions. The no-*slip condition* states that the velocity should average to zero along the boundaries. To implement this, we added an additional function to both solvers called the *boundary* function. The main goal is to ensure the system remains mass conserving and provides the behavior of the fluid-boundary interactions. The most important role of the *boundary* function is to control the behavior of the velocity fields around the boundaries. As previously stated, the *no-slip condition* states that the average of the velocity field should be zero along the edge of the boundaries. What is nice about this model is that the velocities are stored in component form. This mean the only boundaries that are of interest to the u vectors are the surfaces that run perpendicular to the vectors, in this case vertical boundaries, while the *v* vectors look at the horizontal boundaries. When running the *boundary* function for the vector ports, you trace along the boundary and set the vector ports for those boundary cells to be equal to the negative of the neighboring non-boundary cell's corresponding vector port. That way if you were to average the two values the result would be zero. This zeroing of the velocity field will stop the densities from interacting with the boundaries. By ensuring that boundaries are more than one cell wide, we reduce the loss that would be generated from having to average the values of more than two cells. When a boundary is in contact with only one non-boundary cell, it is equal and opposite to the state value of the non-boundary cell. When in contact with two non-boundary cells, it needs to assume a state value that is equal and opposite to the average of the adjacent cell's state values. This introduces the possibility for some loss of mass to occur, however it is minimal.

As previously mentioned, the other role of the boundary function is to ensure the system remain mass conserving, i.e. that the presence of boundaries does not negatively impact the system. Therefore, the *boundary* function is integrated when the other functions are called. For example, to ensure no mass is lost during the diffusion of the densities, the boundary cells assume a value that is equal to the average of the surrounding non-boundary cells, as seen in equation 13.

$$D_{Boundary}(i,j) = \frac{(Sum\ of\ Non\ Boundary\ Cells)}{\#\ of\ Non\ Boundary\ Cells}$$

Equation 12. Boundary Equation

In figure 5, we can see a density focus encountering a fixed obstacle. With the viscosity set relatively low for both the velocity and the density (0.05), we can see that the focus splits into two distinct clouds, mostly by the velocity field. With the viscosity of the velocity field being low, we see a space of zero velocity directly behind the obstacle, as we would expect.

The projection function was responsible for ensuring that the fields remain mass conserving and for adding visual effects. First, as can be seen in figure 5, the velocity vectors never exceed their allowed size. As we expected the velocities were slowed by the presence of barriers. More importantly though is how the projection function handled the collision between the velocity field and the obstacle. As described earlier the velocity value for the "boundary" cells should be equal to the negative of the neighboring cells, as seen in the first frames of the figure. This would cause a convergence in the velocity field, which causes instability. It is to handle these situations that this function exists. As seen in the second set of frames from figure 5, it handles this convergence of forces by taking the forces entering the area directly in front of the obstacle from the velocity field, and diverting the horizontal forces up and down.

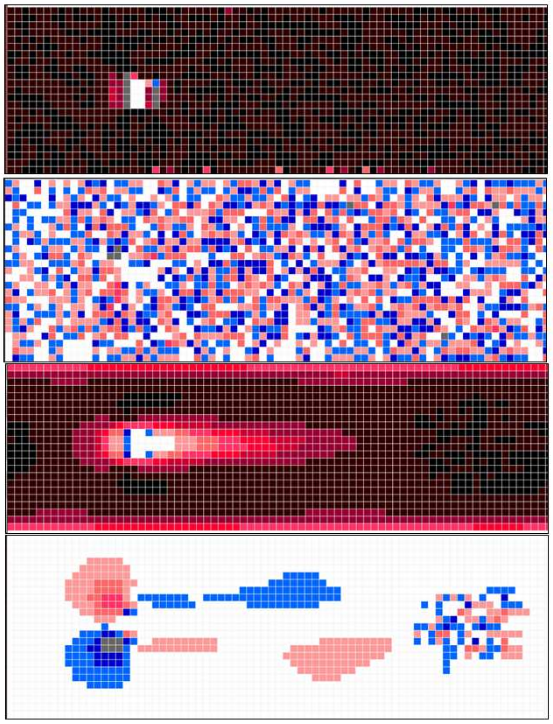

Figure 5. Velocity Component Vectors u and v: initial values and mid simulation values with an obstacle

This is a solution reflects the real-world behavior of such a fluid-boundary interaction. The final goal of the projection function is to create visual effects such as eddies. In the real world, forces would be drawn into the null region created behind an obstacle. As seen in figure 6, the projection function causes forces created to drive the densities into this space.

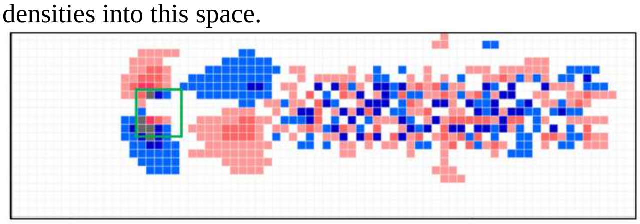

Figure 6. Vertical velocity component with obstacle.

Figure 6 shows the vertical forces acting on the densities. As we can see, upon initial contact with the obstacle the projection function creates a force that pushes the densities around the obstacles and pulls them back in behind the obstacle to the area of low pressure.

4. APPLICATION

In this section we discuss a real-world application for the model. We will look at the effect the narrowing of the coronary arteries due to plaque buildup will affect the flow of blood to the heart muscles. The simulations are uploaded and executed remotely on the RISE server [10] using the CD++ simulator presented in the introduction. The results are then downloaded and visualized using a 2D tool, as seen in the different figures presented in this section.

Coronary Artery Disease (CAD) is the leading form of heart disease and the leading cause of heart attacks, resulting in the most deaths world-wide. CAD happens when plaque builds up on the artery walls. This accumulation of plaque hardens, and thus narrows the arteries. This narrowing restricts blood flow through the arteries, and since these arteries are supplying blood to your heart, the restriction of blood flow weakens it. The dangerous part of CAD is that typically patients suffering do not show any immediate signs or symptoms. Eventually, the myocardial cells will become ischemic from the lack of oxygen and potentially cause a heart attack [11].

To detect for CAD, doctors often perform an angiogram. In an angiogram, dye is injected into the coronary artery via a catheter and a rapid series of x-ray images are used to track the flow of the dye through the arteries and detect any narrowing or blockages [12].

CFD is used to simulate the interaction between blood flow, artery walls, and the plaque that can lead to CAD [13]. This application of CFD may benefit from the possibility of generating patient-specific models of arterial geometry using angiogram data [14]. The example presented here differs from studies found in medical literature in that blood flow is modeled with a rule-based approach using Cell-DEVS methodology.

The following simulation is an attempt to demonstrate how the narrowing of the arteries affects the blood flow. Several scenarios will be run: a control test blockage (0%), a minor blockage (17%), a medium blockage (35%), a major blockage (52%), and a late stage CAD blockage (70%). A single bolus of dye will be "injected" into the artery that is initialized with a uniform velocity field.

Figure 7 shows the results for all the scenarios after 25 iterations have passed. There is no significant difference between 0% and 17% blockage manly due to the size of the bolus and the size of the blockage. The 35% blockage shows the bolus being more concentrated and not being allowed to diffuse as much, however there is still little effect on the velocity field. The 52% and 70% blockage show a decrease in the velocity field due to the narrowed arteries.

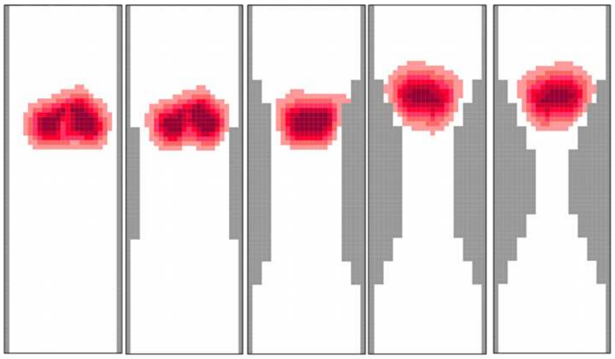

Figure 7. Simulation of CAD after 25 iterations: 0%, 17%, 35%, 53% and 70% blockage respectively

Figure 8 shows the results after 50 iterations have passed. Again there is no significant difference between the first three scenarios; however, the larger blockages are slowed significantly and have not passed through the blockage site yet. This is expected since the behavior of the fluid-boundary interactions is such that flow decreases near the boundary walls and with a narrow channel the flow is slowed across the width of it.

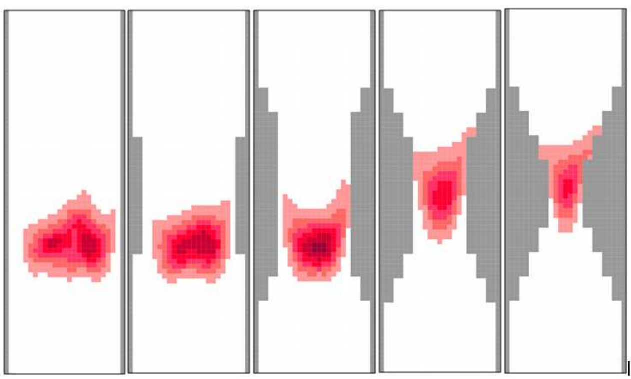

Figure 8. Simulation of CAD after 50 iterations

The following shows the vertical velocities at this point in time for the 53% blockage. During the second pass of the bolus, we start to see the most significant results.

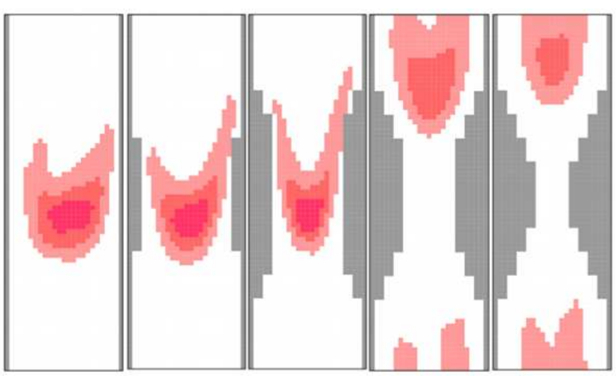

Figure 9. Simulation of CAD after 125 iterations

By the 100th iteration the first three scenarios are making their second pass through the blocked section of artery. The 35% blockage scenario is showing a decreased velocity field along the edges of the artery resulting in the bolus being longer. Also, the bolus has fallen behind from the control signifying a slight decrease in the flow rate. The last two scenarios show a large decrease in the flow rate and in the concentration of the bolus that has made it through. This demonstrates that at these levels of blockage we would most likely see a drop in the amount of oxygen being delivered to the nearby heart muscle which could result in a weakening of the heart function. In Figure 10 we see what would happen if there was a second blockage downstream of the first, of equal size.

In Figure 10 we see that with a second blockage of 35% there is a further decrease in the flow rate, resulting in less dye passing through the blockage. With 53% and 70% blockage would see very little of the bolus making it through the blocked region and the flow rate being further reduced.

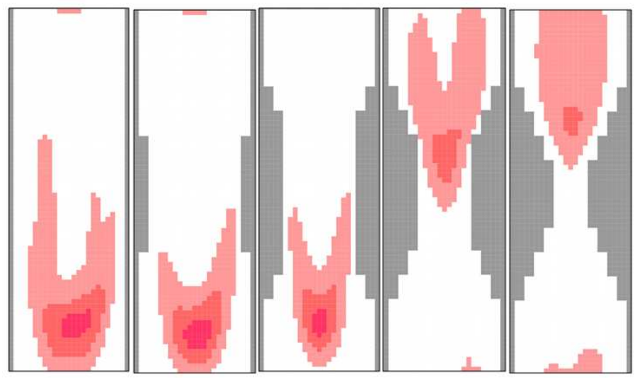

Figure 10. Simulation of CAD after 150 iterations

As stated before, there exists no analytical solution for fluid flow. The goal of all CFDs is to provide results that accurately portray reality. The goal of this research was to create a CFD solver that could be integrated to solve these flows for any system, specifically for use in biological systems. What we simulated here was a part of a larger system. We looked at how the model would behave if the passage width was decreased by a blockage; the real-world equivalent being CAD and its effect on blood flow through the arteries of the heart. The results generated matched what we expected to see and even helped provide a visual explanation of the dangers of CAD. As we increased the size of the blockage we saw progressively larger decreases in blood flow rate and in the amount of dye that passed through the blockage. However, it was never to the point that the flow was completely cut off and there was still a significant amount of flow passed through the blockages. In the real-world this would signify that oxygen is still being delivered to the muscles in the heart and an increase in blood pressure could offset the slight oxygen shortage due to the blockage. This decreased flow would still likely be enough to maintain regular heartbeat during restful activities, but what happens when the heart rate increases? The muscles will require more oxygen. However, as we see with the 53% and 70% blockages any increase in pressure upstream of the blockage would not result in a change in the rate of flow downstream of the blockage. This would result in the heart muscles becoming ischemic and cause a cardiac event to occur, such as a heart attack. We know that people suffering from CAD show no real symptoms until a cardiac event occurs, which would most likely occur after a period of high exertion.

The model presented in this research met many of the goals we wished to achieve. The rule-based nature of the model code is significantly easy to understand. This means users will be able to easily adjust the parameters of the rules to modify the model so it best fits their needs. The model included a function for defining the behavior of fluid barrier-interactions which will ensure it can be used for a wide range of application. Finally, the method of simulation allowed for only the results at the end of each iteration to be stored, and only the density values stored, or whichever values you are interested in.

As previously mentioned, by restricting the velocity vector magnitudes it gave rise to the idea of varying time steps. What this means is that during periods of velocities with large magnitudes the equivalent time an iteration represents is decreased. During periods of low flow, the length of time an iteration represents is increased. This variance of time can be handled by the DEVS simulator, which is an important feature, and can help reduce the computational load when it is implemented into a biological system that has periods of high and low flow, a beating heart for example.

The results presented in this give a small glimpse into the possible application of the CFD model presented in this research.

Once again, the benefits of the DEVS formalism are such that, if a new simulator is developed, by keeping the model and the simulator separate, the model will not require any adjustments to run on this theoretical simulator.

5. CONCLUSION

Fluid dynamic solvers are used in a wide variety of application ranging from video games and entertainment to modeling of environmental events and biological systems. In this research, a CFD solver is proposed that uses the parameters of a CA in Cell-DEVS. The asynchronous and more efficient computing grid of Cell-DEVS with the continuous time-base allowed for more realistic simulation of fluid dynamics. We showed how CD++ toolkit was used to implement the Cell-DEVS model of the Navier-Stokes equations for CFD. We were able to create a fluid dynamic solver that met the requirements of a Cellular Automata, demonstrating that it is possible to create models of vary complex phenomenon using a relatively simple technique. The model was a significant improvement to the first version, in that it was able to provide results with a better resolution in a significantly shorter time. The model also improved the size of the log files generated which was a major concern of the last model, without sacrificing the ability to access the high level of detail generated during the evolution of both the density and velocity field. The results shown in this paper demonstrate that it is possible for a CFD model to be created and coupled to help resolve the physics of the fluid flow in any system; biological, environmental, etc.

REFERENCES

[1] Anderson, J. D. "Basic philosophy of CFD." Computational Fluid Dynamics, pp. 3-14, 2009.

[2] Ilachinski, Andrew "Cellular Automate: A Discrete Universe" World Scientific Publishing Co. 2001.

[3] Zeigler, B. P., Praehofer, H., & Kim, T. G. (2000) Theory of modeling and simulation. 2000.

[4] Wainer, G. A. Discrete-event modeling and simulation: a practitioner's approach.CRC, 2009.

[5] M. Van Schyndel, G. Wainer, M. Moallemi. Computational Fluid Dynamic Solver based on Cellular Discrete-Event Simulation. In Proceedings of SIMULTECH 2013.Rejkyavik, Iceland. 2013.

[6] Toro, Eleuterio F. "Rienmann Solvers and Numerical Methods for Fluid Dynamics: A Practical Introduction 3rd Edition" Springer-Verlag Berlin Heidelberg. 2009.

[7] Saleh, Jamal Mohammed.Fluid flow handbook. New York (NY): McGraw-Hill, 2002.

[8] Currie, I. G. Fundamental Mechanics of Fluids, McGraw-Hill, Inc., 1974.ISBN 0-07-014950-X.

[9] J. Stam, Real-Time Fluid Dynamics for Games, Proceedings of the Game Developer Conference, San Jose CA,2003.

[10] Al-Zoubi, K., & Wainer, G. (2010). RISE: Rest-ing heterogeneous simulations interoperability. In Simulation Conference (WSC), Proceedings of the 2010 Winter . IEEE.

[11] Mount Sinai Hospital, Fighting Coronary Disease, [Online] Available: http://www.mountsinai.org/patient-care/service-areas/heart/areas-of-care/heart-attack-coronary-artery-disease

[12] Mayo Clinic, Coronary Angiogram, [Online] Available: http://www.mayoclinic.com/health/coronary-angiogram/MY00541

[13] Chaichana, T., Sun, Z., & Jewkes, J. Computational Fluid Dynamics Analysis of the Effect of Plaques in the Left Coronary Artery. Computational and Mathematical Methods in Medicine. 2012.

[14] De Santis, G., Mortier, P., De Beule, M., Segers, P., Verdonck, P., & Verhegghe, B. Patient-specific computational fluid dynamics: structured mesh generation from coronary angiography. Medical & Biological Engineering & Computing. Volume 48, Issue 4, pp. 371-380. 2010.