**Hydraulic Fracturing Modelling with Cell-DEVS using the Lopez simulator**

*Ngozi Silas Echegini (101032933) and Joseph Boi-Ukeme (101053347)*

Department of Systems and Computer Engineering

Carleton University

December 22, 2017

**ABSTRACT**

Hydraulic fracturing in oil and gas reservoirs often leads to a development of complex fracture networks. It is an important venture for the oil and gas industry to understand the nature and extent of the complexity of ensuing fractures, to optimize the strategies for designing and completing exploration exercises, having an idea of possible outcomes.

Fractures come about as a result of the action of existing high pressure fluids in underground reservoirs called wellbores, which exert a force proportional to their inherent pressure on surrounding rock bodies. The extent of their complexity are analogous to a nerve network structure of varying length, width and height combinations. Several researchers like Sergio Zlotnik have proposed methods of modeling of hydraulic fracture, using Cell – DEVS to simulate the propagation of a complex fissure network with both natural and induced cracks. This proposed method was then implemented by Christopher Burt and Ifeoluwa Oyelowo improved on Burt’s model in Cell-DEVS as well, while attempting to implement the same model in the Lopez simulator.

Ifeoluwa experienced difficulties in this implementation and proposing a few ideas as future work. Not neglecting these ideas, we build on the previous author’s existing implementation and proffer our own solution towards translating into Lopez and creating a better visual of the fracture pattern. Our improved model and translation into Lopez will be discussed in this report.

1. **INTRODUCTION**

Hydraulic fracturing is one of the techniques for producing cracks by pumping the fluid at a relatively high speed and pressure into existing or artificially made rock crevice’s. This process is widely adopted in the oil and gas industry for extracting oil and gas resource, trapped in underground reservoirs called wellbores. The size of these fractures can range from several meters to hundreds of meters, and their cost is often an important part of the overall cost of oil exploration exercises.

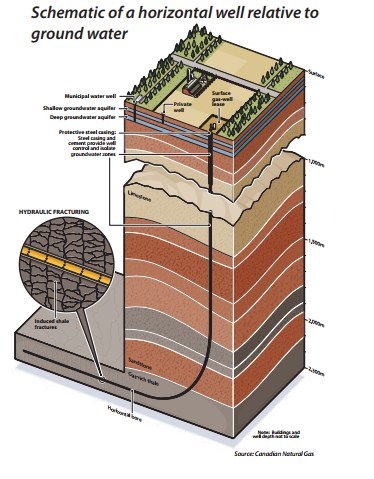


Figure 1: hydraulic bore showing the action of hydraulic fracturing.

The diagram above shows the extent of the depth of oil reservoirs and the ensuing fracture from hydraulic fracturing process. Possibly, the geometric accuracy of hydraulic fractures can be predicted and controlled in locations where the *in-situ* stress field, the direction and the position of one of the wells is known. The model proposed by Sergio Zlotnik as was earlier on referred, was implemented by Christopher Burt in Cell-DEVS and another Student, Ifeoluwa Oyelowo, improved on this implementation using Cell-DEVS and attempted translating the same implementation to the Lopez version which was not successful. The aim of this project is to build on the existing work done by Ifeoluwa, addressing the open issue related to translating the existing model to a Lopez version and improving on the work done, exploring some other aspect of hydraulic fracturing which are relevant and were not investigated by previous authors. For example, the width and extent of fracture, when a fracture should stop or is likely to stop.

The rest of the report is arrayed as follows: In Section 2, we present the background information of the relevant work done in the literature; in Section 3, we present the conceptual and formal definition of this model, in Section 4 and 5, we discuss our implementation of the project, and the results from simulations, Section 6 includes our discussions, Section 7 and 8 include our remarks, future work and conclusions.

1. **BACKGROUND**

The hydraulic fracturing implementation by Burt was an underground hydraulic fracturing event, designed to be of two-dimensional cross section, and in a 15x15 cell spacing. This implementation is a simple model which considers the fluid pressure, ignoring the pressure exerted by the wellbore. Each cell at each moment had a state that represents the pressure of that cell. An unfractured cell (rock) had zero pressure and a fractured cell had a pressure greater than zero. Included also, was a special cell called a well sprout that constantly supplied hydraulic pressure in the system.

The improved version implemented by Ifeoluwa in Cell-DEVS was of three-dimensional cross section and in a 25x25 cell space. The implementation was an improvement on Burt’s implementation and considering, important parameters like the well-sprout pressure which Burt did not consider. The author succeeded in implementing a model using derived pressure formulas and direction generators. In the results from this implementation, the fracture patterns are much closer in behavior to the real-life fracture patterns. However, implementing the same model using the Lopez simulator did not yield the desired results. This forms the bases of our work. Asides the above discussed works, some other authors have done research in this area as it is a broad research area. However, these authors concentrated on mathematical modelling which is complex. Cell DEVS provides a better approach of studying these concepts and Lopez simulator is an extended version of CD++ which allows the use of multiple state variables and multiple neighbor ports for each cell in a Cell-DEVS model.

1. **MODEL DEFINITION**

In this section, we define the conceptual and formal definition of the hydraulic fracture model we implement. We started by replicating the Cell DEVS version of the existing model and then implemented the Lopez version of the model.

**3.2 CONCEPTUAL DEFINITION OF THE MODEL - Lopez simulator**

The previous author, “Ifeoluwa” had implemented this same model in CD++ and obtained functional results which is close to what an actual fracture will look like. We therefore delve into the translation of this model to a working Lopez model.

As initially stated, the Lopez simulator is an extension of CD++. The major advantage of the Lopez simulator over CD++ is the option which allows the for multiple state variables and multiple neighbor ports for each cell in a Cell-DEVS model. The CD++ implementation included two extra planes which represented state variable mappings to the cells in the plane-0, “the pressure plane”. These planes are compressed into a single plane in Lopez, each cell having state variables and multiple state ports. The state variables represent the planes in the Cell DEVS model.

This implementation has cell-spaces of 50x50 and 100x100 instead of the 25x25 in Ifeoluwa’s implementation. Each cell in the cell space piggybacks on the existing work done. However instead of using state variables, we simplified the model to include only neighbor ports. Each cell has three (3) neighbor ports which we describe below.

The three neighbor ports hold values for pressure, state and direction and are called as “pressure port”, “state port” and “direction port”. The direction of a cell is a random integer generated with a random generator function and integers are in the range corresponding to the number of neighbors a cell has. We consider the Moore neighborhood where each cell has eight neighbors. Thus, the direction of a cell is an integer in the range 0 to 8. One advantage of this wide band of randomness is to reduce the likelihood of having a cluster of integers which then define the same direction and will not depict a proper fracture pattern.

Cells that are Rock cells have a threshold at which they now become fractured cells. This threshold is called the maximum permissible pressure. The formula used to calculate this value was obtained from the Ifeoluwa’s paper in [2] and will be presented in a later chapter. The formula represents the depth below the surface of the earth, which in the cellular space corresponds to the vertical position of the cell in space. In this model, the maximum pressure supported by a rock is multiplied by the strength factor of that rock whereas the resistance factor increases with the number of neighboring fractures. This represents the rock that becomes stronger because it is compressed by fractures that open nearby. The use of the resistance factor was proposed by Burt and applied by Ifeoluwa as well. The direction is another factor that influences the fracturing of a cell. Thus, in simple terms, when the direction of a cell is equal to the direction of any of its neighbors, and the pressure exerted by the neighbors exceed its maximum permissible pressure, the cell becomes a fractured cell.

**Neighbor Ports**

Each cell in the model has 3 neighbor ports which hold values that provides some information about the cell to its neighbors. More details concerning the neighbor ports are provided below:

* **State port**: Holds a value corresponding to the three possible occurrences of a cell. A cell will either exist as a rock, a fractured cell or a well sprout. A value of 0 represents that a cell is a rock, 1 represents that a cell is a fractured cell and 2 represents that a cell is a well sprout.
* **Pressure port**: holds the value corresponding to the instantaneous pressure of a cell per time, and makes this available to neighboring cells. The range of values of a cells pressure are in the region [0,]. 0 indicates that the cell is yet a rock and the pressure for a rock and none of its neighbors have been able to overcome its resistance factor.
* **Direction port**: holds the value corresponding to the direction of a cell. This value is a random integer output from a random generator function and is made available to a cells neighborhood cells. The range of values of a cell’s direction are in the region [0, 8]. Each cell – a rock cell especially, has a direction which is used in conjunction with other parameters (pressure port and state port) to determine if a rock cell will be fractured or not.

**3.3 Model Modifications**

**Direction port modification**

In previous fracturing models done, the direction plane was made to have 4 possible directions (0,1,2 3) in a random fashion, but here was not justification why we should have four (4) possibilities out of 9 cells, hence we did an analysis of various possibilities for randomizing the integers for direction and found out that with 8 we had the best results in terms of closely mimicking what obtains, hence for the direction plane a random integer 8 was used.

In addition to the modification of the direction port, the following cases briefly discuss other model modifications done to improve the hydraulic fracture model

**Case 1: Translation of fracture model using same rules to run Lopez simulator**

At the start of this project, we inherited a fracture model that worked on CD++ and behaved completely differently, hence our first task was to translate the fracture model rules to the syntax of Lopez.

**Case 2: Modification of fracture rules to include porosity**

Porosity can be defined as the ratio of pore volume (in this case the spaces because of fracturing) to the total volume (both pores and rock volume). Analyzing the neighborhood of the cells (9 cells) to achieve typical porosity values obtainable in Canada (from 15% to 30%) [national resources Canada] we ensured that the number of fractured cells in the neighborhood must not exceed three (3).

**Case 3: Stopping conditions for the fracture.**

In previous models, there were no explicit stopping conditions for the propagation of the fractures and this was far from what is obtainable, hence to solve this problem we undertook two different approaches:

* Reduce the number of cells responsible for averaging the well spout pressure, this was achieved by modification of the neighborhood.
* Gradually reducing the well spout pressure as the fracture propagates.

**Case 4: Monitoring the effects of cell space**

In previous model a cell space size of 25 x 25 were used and the results were a bit mechanical, and we believe that some fluidity of the fracture was needed and one approach to achieving this was to increase the cell space to 100 X 100.

**3.4 FORMAL SPECIFICATION**

**Formal Specification for the Hydraulic Fracture model (Lopez simulator)**

CD = < X, Y, I, S, θ, N, d, δint, δext, τ, λ, D >

To represent the well spout, we made it a zone.

}

}

}

I = <η, μ, Px, Py>

η = 9

μ = 3

Px = {all in neighborhood}

Py = {all in neighborhood}

Θ = {

State port = [0 to 2]

Pressure port = [0 to ]

Direction port = [0 to 8]

}

N = neighborhood = {(0, 0), (0, 1), (0, -1), (1, 0), (1, -1), (-1, 1), (-1, 0), (-1, -1), (1, 1)}

d = 100 ms (transport delay)

τ: N🡪S: as defined in HydrauFractureLP.ma

**We explain the rules we used in more details below:**

**Rock rule:**

A rock cell will become a fractured cell if the pressure of the neighboring cells or any of the neighboring cells exceeds the product of its maximum supported pressure and resistance factor [2, 3]. Also, the neighborhood cell must have the same direction as the rock cell to fulfil the requirements for fracturing.

The maximum pressure support of the rock cell has the following formula: (Obtained from [2])

**Maximum Supported Pressure**

Where:

is the smallest principal stress;

is the pore fluid pressure; and

is the tensile strength of the rock.

The pore fluid pressure and tensile strength is exclusive to each cell. They are computed as described in Figure 3 below [2, 3].

Table 1: Definition of variables used to compute Maximum Pressure Support (from [2])

|  |  |  |  |
| --- | --- | --- | --- |
| **Symbol** | **Description** | **Units** | **Value** |
| ***Ρ*** | *Rock density* | *Kgm -3* | *2200* |
| ***Τo*** | *Rock tensile strength* | *Pa* | *15±1 \* 106* |
| ***G*** | *Gravity acceleration* | *Ms -2* | *9.8* |
| ***gP*** | *Pore fluid pressure gradient* | *Pa m -1* | *10500* |
| ***Pf*** | *Pore fluid pressure* | *Pa* | *Depth(m) \* gP* |
| ***Σ*** | *Stress* | *Pa* | *Depth(m) \* g \* ρ* |

and depend on depth, so they will have different values as depth increases [2]. In the Hydraulic Fracture model, we replicate the same assumptions in [2, 3], that the fracture is taking place at 10,000 ft (3048 m) below the ground. As a result, depth is computed as (3048 + cellPos(0)) where cellPos(0) is the row index of each cell.

The resistance factor of the rock cell has the following formula: (Obtained from [2, 3])

**[Resistance multiplier minimum] + [resistance multiplier range span] \* ([percentage of fractured non-self-neighbors])**

The above formula is interpreted as follows:

**1 + 10 \* (8 – ([number of cells which are zero or undefined])) / 8**

The interpretation of the above formula follows that, the number of cells with “state port value” zero or undefined are counted with a series of checks that return 1 if true or zero if false.

**Fracture rule:**

A cell becomes a fractured cell if the initial state port value of this cell is ‘0’ i.e. is a rock cell, any of its neighbors have the same direction as this cell and the pressure of its neighbor or neighbors exceed its maximum resistance factor.

If a cell is a fractured cell i.e. state port = 1, then the pressure of that cell is the average pressure of all the neighborhood cells that are fractured cells. Otherwise, the cell keeps its current value.

**Zone “wellspout” rule:**

The wellspout cell is a special case cell which is constantly incrementing its pressure by 10 MPa, generating the pressure required to create the fracture. It also follows the fractured cell rule.

**Direction rule:**

The direction port of each cell constantly holds a random value between 0 and 8 every 100 ms. We chose this range for a random number generator as there are 8 neighbors to each cell – Moore neighborhood and increases the randomness of the direction of each cell, reducing the probability of clustering.

The equations for the Hydraulic Fracture model were defined as macros and stored in *.inc* files for various cases.

**4.0 Implementation of the Hydraulic Fracture model in Lopez**

This implementation includes a single plane of varying cell spaces. There is one for 50 x 50 - 2500 cells and another for 100 x 100 – 10,000 cells. Burt in [3] used a cell space of 225 cells (15 x 15) and Ifeoluwa used a cell space of 625 cells (25 x 25). We subscribed to making the cell space in this model larger to have a rather clearer visual the fracture patterns as they propagate in a larger cell space.

We present the basic definitions for the model below:

This implementation includes five (5) different models which are all variations of the first model – translating the CD++ model version to the Lopez version. We present one of these model as a reference to the work done. The rest of the models can be found in the files with *.ma* extension whereas the formulas used in the model are defined as macros in the files with the *.inc* extension.

#include(HFMacrosCase4a.inc)

[top]

components : HydrauFracture

[HydrauFracture]

type : cell

dim : (100,100)

delay : transport

defaultDelayTime : 100

border : nowrapped

neighbors : HydrauFracture(-1,-1) HydrauFracture(-1,0) HydrauFracture(-1,1)

neighbors : HydrauFracture(0,-1) HydrauFracture(0,0) HydrauFracture(0,1)

neighbors : HydrauFracture(1,-1) HydrauFracture(1,0) HydrauFracture(1,1)

initialValue : 0

%Ports for stateport, pressureport and directionport variables

neighborports : stateport pressureport directionport

localtransition : hfrule

zone : wellspout { (40,1) }

***The “well sprout rule” talks about the cell which behaves like a fractured cell from its initial condition, and continuously increases its pressure by 10MPa***

[wellspout]

%Rule for the well spout cell

rule : { ~directionport := randInt(8); ~pressureport := #macro(FracturedCellpressure) + 1000000; ~stateport := 2;} 100 {t}

***the HFrules define the general rules for a cell which is yet a rock and will become a fractured cell once certain conditions are satisfied. This rule checks for these conditions and executes accordingly.***

[hfrule]

rule : { ~stateport := 1; ~pressureport := 1; ~directionport := randInt(8); } 100 { ((0,0)~stateport = 0) and ((-1,-1)~pressureport > #macro(MaxpressureSupport)\*(1 + 10 \* ((8 - #macro(ResistanceFactor))/8))) and ((-1,-1)~directionport = (0,0)~directionport) and (statecount(1, ~stateport) < 2) }

rule : { ~stateport := 1; ~pressureport := 1; ~directionport := randInt(8); } 100 { ((0,0)~stateport = 0) and ((-1,0)~pressureport > #macro(MaxpressureSupport)\*(1 + 10 \* ((8 - #macro(ResistanceFactor))/8))) and ((-1,0)~directionport = (0,0)~directionport) and (statecount(1, ~stateport) < 2) }

rule : { ~stateport := 1; ~pressureport := 1; ~directionport := randInt(8); } 100 { ((0,0)~stateport = 0) and ((-1,1)~pressureport > #macro(MaxpressureSupport)\*(1 + 10 \* ((8 - #macro(ResistanceFactor))/8))) and ((-1,1)~directionport = (0,0)~directionport) and (statecount(1, ~stateport) < 2) }

rule : { ~stateport := 1; ~pressureport := 1; ~directionport := randInt(8); } 100 { ((0,0)~stateport = 0) and ((0,-1)~pressureport > #macro(MaxpressureSupport)\*(1 + 10 \* ((8 - #macro(ResistanceFactor))/8))) and ((0,-1)~directionport = (0,0)~directionport) and (statecount(1, ~stateport) < 2) }

rule : { ~stateport := 1; ~pressureport := 1; ~directionport := randInt(8); } 100 { ((0,0)~stateport = 0) and ((0,1)~pressureport > #macro(MaxpressureSupport)\*(1 + 10 \* ((8 - #macro(ResistanceFactor))/8))) and ((0,1)~directionport = (0,0)~directionport) and (statecount(1, ~stateport) < 2) }

rule : { ~stateport := 1; ~pressureport := 1; ~directionport := randInt(8); } 100 { ((0,0)~stateport = 0) and ((1,-1)~pressureport > #macro(MaxpressureSupport)\*(1 + 10 \* ((8 - #macro(ResistanceFactor))/8))) and ((1,-1)~directionport = (0,0)~directionport) and (statecount(1, ~stateport) < 2) }

rule : { ~stateport := 1; ~pressureport := 1; ~directionport := randInt(8); } 100 { ((0,0)~stateport = 0) and ((1,0)~pressureport > #macro(MaxpressureSupport)\*(1 + 10 \* ((8 - #macro(ResistanceFactor))/8))) and ((1,0)~directionport = (0,0)~directionport) and (statecount(1, ~stateport) < 2) }

rule : { ~stateport := 1; ~pressureport := 1; ~directionport := randInt(8); } 100 { ((0,0)~stateport = 0) and ((1,1)~pressureport > #macro(MaxpressureSupport)\*(1 + 10 \* ((8 - #macro(ResistanceFactor))/8))) and ((1,1)~directionport = (0,0)~directionport) and (statecount(1, ~stateport) < 2) }

%rule for a fractured cell

rule : { ~stateport := 1; ~pressureport := #macro(FracturedCellpressure); ~directionport := randInt(8); } 100 { (0,0)~pressureport > 0 }

rule : { ~stateport := 0; ~pressureport := 0; ~directionport := randInt(8); } 100 { t }

**5.0 SIMULATION RESULTS**

In this section, we briefly present the results obtained from all our simulations of the different variations of the hydraulic fracture model which we have implemented and investigated in the Lopez simulator. The results pictures and explanations of what these pictures represent.

At the beginning of the simulation, the cells in the cell space assume their initial values which are values held by the state ports as follows: state port = 0, pressure port = 0 and direction port = 0. The wellspout cell is a special cell as initially stated. It initially assumes a rock status (i.e. state port = 0). We faced the same issue as the previous author “Ifeoluwa” - where the Lopez simulator was not able to open the “*.stvalues”* file [2]. For the same reason as in [2], the Lopez simulator did not recognize the wellsprout cell being on the border of the cell space. We thus adopted the same assumption in [2] and selected cell (11,1) as the wellspout cell, which is the cell in the row following the 0th row. Figure 3 below shows the initial values at time 0. We then loaded the relevant files into the ARSLab - CellDEVS Viewer. The resulting visual is as represented in Figure 4.

The plane showing the pressure is labelled as Layer 0- [Pressureport]. In this layer, blue is a representation of cells which have translated from being rocks to fractured cells and dark brown represents cells that are still rock cells. The different shades of blue differentiate higher pressure values from lower pressure values in this order:

dark blue cells: mean higher pressure cells light blue: cells mean lower pressure cells.

For all representations and for all cells in the region of cell space, the extreme left cell space shows the values held by the state port, the center cell space shows the values held by the pressure port and the extreme right cell space shows the values held by the direction port.

To begin with, we present the comparison between the attempted implementation by Ifeoluwa and our Implementation which is translating the CD++ model to a working Lopez model.

Figure 2 shows Ifeoluwa’s implementation and Figure 3 shows the working improvement.

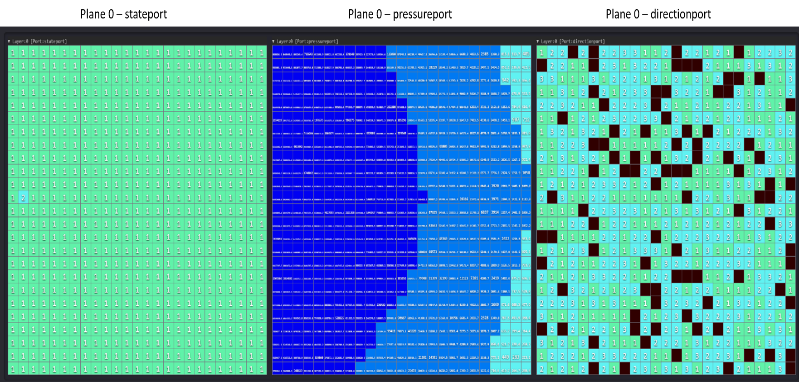


Fig. 2: Results from Ifeoluwa's Implementation

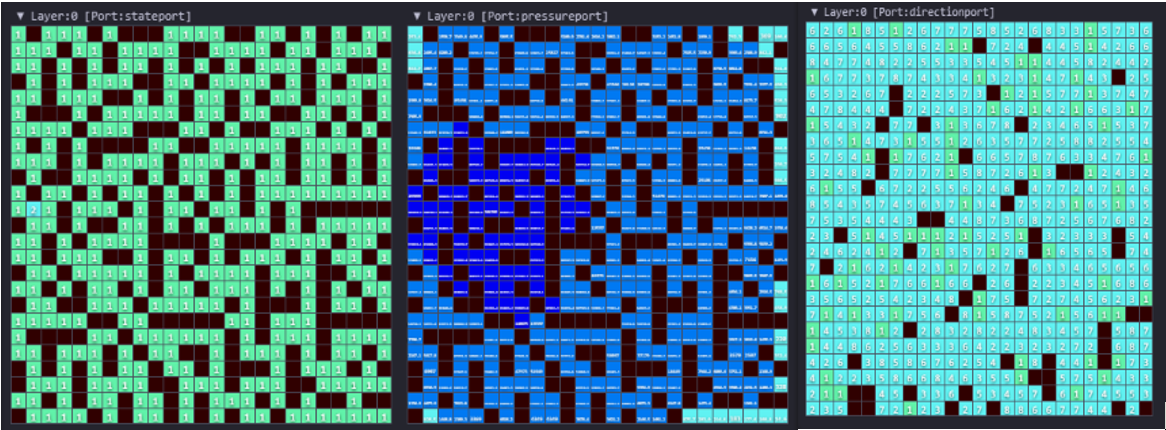


Figure 3. Working improved version of Fig 2. Above

**5.1 PROBLEM:**

With the previous author, in the person of Ifeoluwa, the problem the author faced was translating the working CD++ version of the hydraulic fracturing implementation into a working Lopez simulator version. The author successfully ran simulations, but the result was not in any way, a replication of what was obtained in the CD++ version. The main challenge was that in the extreme left plane which the author described as the state variable plane, and the middle plane which the author described as the pressure plane, cells where changing state in a fashion which did not replicate the thought and intention of the author from the perspective of the results from the CD++ version. The pressure state-variable follows the state state-variable. Thus, if the state state-variable behaves in an unexpected way, even if the direction state-variable behaves in an acceptable or expected way, the resulting pressure state-variable will follow the state state-variable.

We fixed this issue in three steps:

1. We modified the rules which governed the translation of a cell from its initial existence of a rock cell to a fractured cell.
2. We simplified the current version to include only state ports and did not make use of state variables. We did this for simplicity.
3. We modified the macros for various model scenarios we implemented. For example, we implemented a model which took into consideration – porosity and another with fracture stopping conditions. We did these for 50x50 and 100x100 cell spaces. We discuss these models below.

**5.2 MODEL CASE 2**

The model represented in pictures by figures 4, 5 and 6 are the results of what we obtained when we considered the porosity of the rock cells. Porosity is another interesting factor to consider in the study of the complex fracture pattern in hydraulic fracturing. Porosity is normally expressed as the percentage of the total rock which is taken up by pore spaces. For example, a sandstone may have an 8% porosity. This means that, 92% is solid rock and 8% is open space containing oil, gas or water. 8% is about the minimum porosity required to make a decent porosity [cite]. For a fractured formation we expect porosity values of about 20 to 30 per cent. [5]

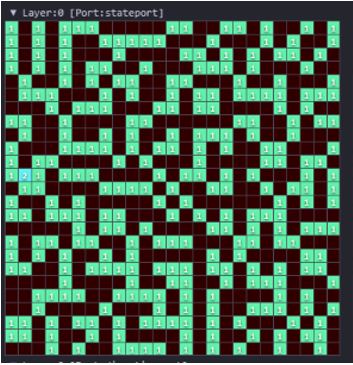


Figure 4: State Port of Plane0 - 25x25 and with porosity considerations

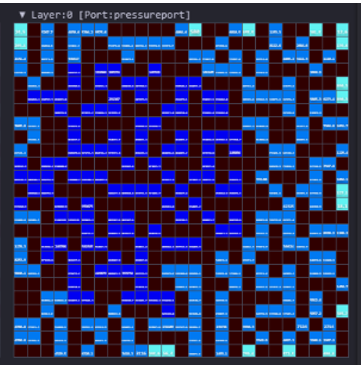


Figure 5: Pressure Port of Plane0 - 25x25 and with porosity considerations

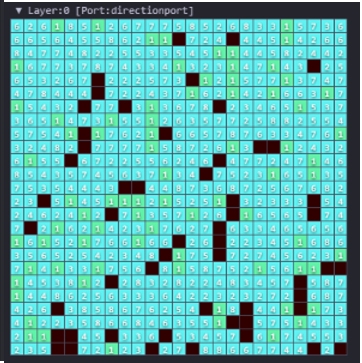


Figure 6: Direction Port of Plane0 - 25x25 and with porosity considerations

**5.3 Case 3: Visuals for the 50 x 50 cell space models**

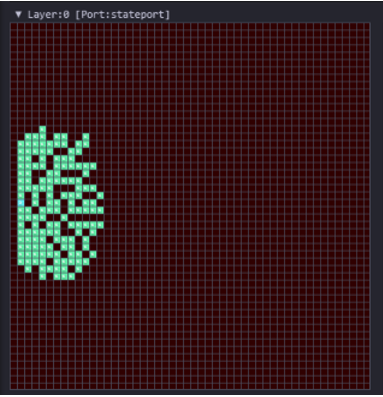


Figure 7 showing the model which includes fracture stopping conditions (Plane0 - State Port)

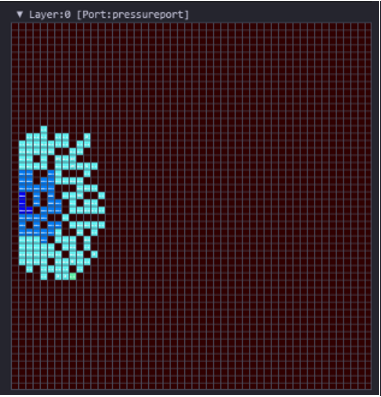


Figure 8: showing the model which includes fracture stopping conditions (Plane0 - Pressure Port)

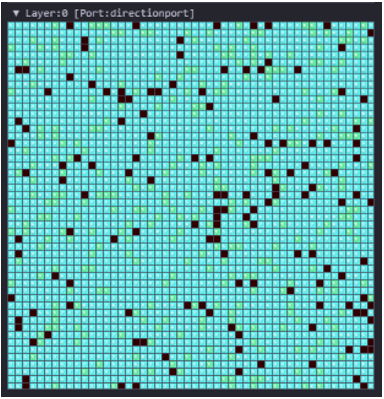


Figure 9: showing the model which includes fracture stopping conditions (Plane0 - Direction Port)

Figures 7, 8 and 9 represent the state, pressure and direction ports of plane 0. This version of the hydraulic fracture model implements fracture stopping conditions. The rules that implement this model interpret that, as the fracture propagates through the rock medium, the pressure decreases by a calculated amount and the fracture will terminate at the point where the pressure of the neighboring cells becomes less than the rock’s resistance factor.

**5.4 Case 4a: Visuals for the 100 x 100 cell space models**

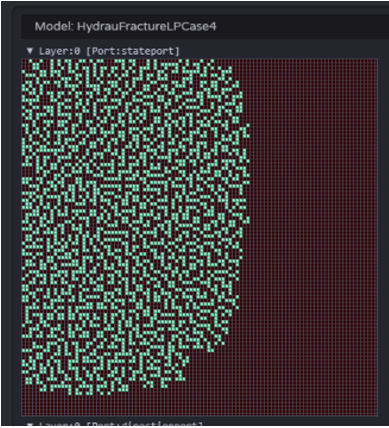


Figure 10: showing the 100x100 scale view (Plane0 - State Port)

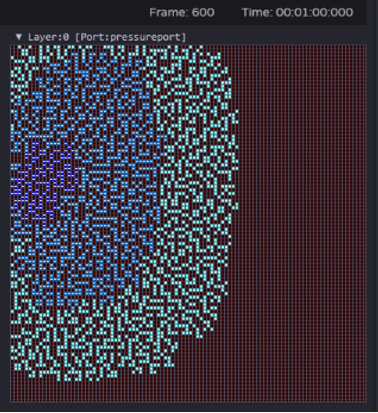


Figure 11: showing the 100x100 scale view (Plane0 - Pressure Port)

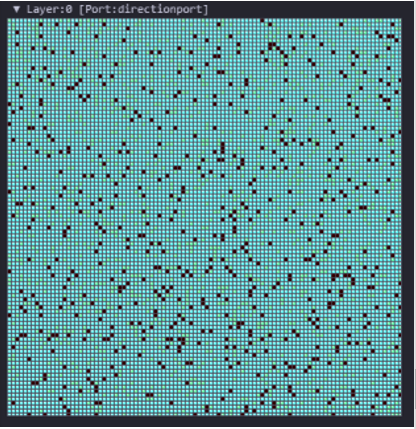


Figure 12: showing the 100x100 scale view (Plane0 - Direction Port)

**5.5 Case 4b: Visuals for the 100 x 100 cell space models with Fracture stopping conditions.**

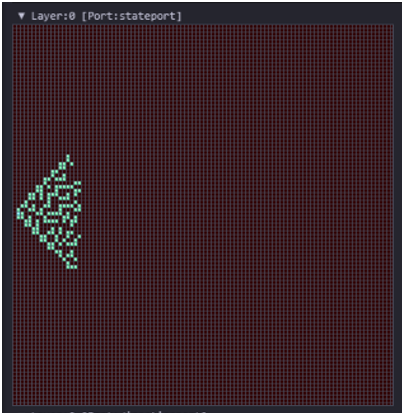


Figure 13: showing the 100x100 scale view (Plane0 - State Port) with fracture terminating conditions

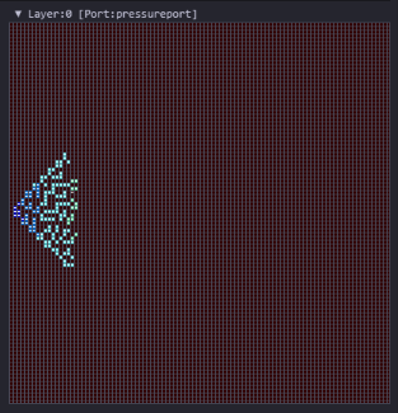


Figure 14: showing the 100x100 scale view (Plane0 - Pressure Port) with fracture terminating conditions

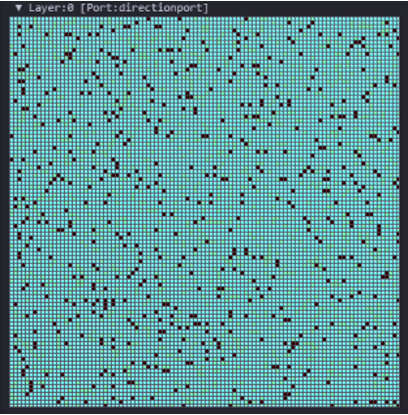


Figure 15: showing the 100x100 scale view (Plane0 - Direction Port) with fracture terminating conditions

**6.0 DISCUSSION**

The author of [2] had complained about the level of technicality in the issued faced in translating the improved version of the Hydraulic fracturing model implemented in CD++. These issues were related to difficulty in interacting with the interface at the time and a bit of complexity in syntax translation. Authors of this report repeatedly faced the same issues but with a diehard attitude, were able to achieve some results.

The authors will however like to mention that it was not a trivial task as it were. The pending issue with the attempted translation of the working CD++ version of the model implemented by Ifeoluwa, into a working Lopez version, had to do with the rules guiding the state changes and coupled with the fact that the pressure port plane followed the state port plane hence, although the pressure port plane depended on both the direction and the state port planes, an unexpected result from the state port plane will replicate an unexpected result in the pressure plane. This issue was fixed, not altering the fact that the pressure port followed the state port which we saw as logical. We made modifications to the rules, in terms of the way a rock cell will respond to messages from its neighbors and what it will do with what message. For example, piggybacking on the existing rule, a rock cell will become a fractured cell based on the discussion in section 3 above and what the author of [2, 3] implemented. The issue we discovered was that, the fractured cell continued to behave like a cell that had not initially been fractured, and increased in pressure as well. This effect propagated in the path and was the reason why many other cells were getting fractures in unison. However, to keep the fracture propagating as it should, the fractured cell only needs to fracture once, and then build up pressure based on the effect of neighboring cells. Another modification was that, we made the direction generator more random to reduce the chances of cluster formation among the generated direction values. We did not just pick a number but assumed that with the Moore neighborhood, each cell will have the probability of existing in a different direction hence the 0 to 8 range of direction values. A second reason was piggybacking on the idea from [2], “to have a more continuous flow in the fracture pattern just like the real-life behavior of hydraulic fracture patterns.” In [2], the author used the same concepts adopted in the CD++ version to build the models for the Lopez simulator, the expected results were farfetched. In the authors words, “The author had a hard time working with the Lopez simulator.” However, we built on the existing code and made modifications to it were necessary as well as removed most things which we thought unnecessary, to the end that we achieved simplicity. The translation from the available CD++ version to the Lopez version of the model described in this report was successful, and we have the results in his report to prove this.

A lot of work has been done on the fracture model to match what obtains about fracture propagation, although there is a significant improvement in results obtained in this work when compared with other models, there is still a lot of improvements that can be made. Some of the areas of work that could greatly improve the hydraulic fracture model include:

* The direction port can be obtained using real life data that can be mapped to a statistical distribution.
* The use of rectangular meshes has limited applications especially when it comes to fluids and grains of sand and rocks, hence it would be important to consider unstructured meshes for the modelling of fractures.

**6.0 CONCLUSION**

This report presents the improvement as well as the implementation of the Lopez simulator version of the Hydraulic Fracturing process which was afore, performed by Christopher Burt [3], was improved upon and implemented in CD++ by Ifeoluwa Oyelowo [2] who also attempted to replicate the same in the Lopez simulator but proved abortive. A few improvements were added to the model. For example, we made the direction generator more random to reduce the chances of cluster formation among the generated direction values. We did not just pick a number but assumed that with the Moore neighborhood, each cell will have the probability of existing in a different direction hence the 0 to 8 range of direction values. A second reason was piggybacking on the idea from [2], “to have a more continuous flow in the fracture pattern just like the real-life behavior of hydraulic fracture patterns.” In [2], the author used the same concepts adopted in the CD++ version to build the models for the Lopez simulator, the expected results were farfetched. In the authors words, “The author had a hard time working with the Lopez simulator.” However, piggybacking again on the authors belief which was put in writing, that “she believes that the code she developed might be of use to future students who will further work on the project,” we built on the existing code and made modifications to it were necessary as well as removed most things which we thought unnecessary, to the end that we achieved simplicity. The translation from the available CD++ version to the Lopez version of the model described in this report was not a trivial task however. In any case, it was successful, and we have the results in his report to prove this. We advanced to extend the model to accommodate some other interesting scenario which we thought will be important to study. We also show these models in this report, and with explanations. When we deployed the version which had porosity, the visual showed better hydraulic fracture patterns than Ifeoluwa’s CD++ version based on the comparison with Ifeoluwa’s results and the results of the model in the report. In addition, we extended the cell spaces from 25x25 in the previous attempted implementation, to 50x50 and 100x100 in our implementation. In our research, we also looked at a few other works in the literature, relevant to this project. They were tailored to specific scenarios and offered interesting approaches. We have referenced one of this works [1]. We have made a few suggestions for improvement of the work in our discussion and we thus hope that, what has been presented thus far will set the stage for next level research for future students who want to explore Hydraulic Fracturing further.

**REFERENCES**

1. Carter, B. J., Desroches, J., Ingraffea, A. R., & Wawrzynek, P. A. (2000). Simulating fully 3D hydraulic fracturing. *Modeling in geomechanics*, *200*, 525-557.
2. Oyelowe, I., “Hydraulic Fracturing with Cell-DEVS using the Lopez Simulator and CD++ “ Carleton University (2016).
3. C. Burt, Simulating Hydraulic Fracturing Using Cell-DEVS.
4. S. Zlotnik, "Simulation of artificial hydraulic fracturing of rocks in boreholes using a Cell-DEVS approach," Ottawa, 2012.
5. C. L. Cipolla, E. P. Lolon, J. C. Erdle, and B. Rubin, “Reservoir modeling in shale-gas reservoirs,” SPE Reservoir Evaluation and Engineering, vol. 13, no. 4, pp. 638–653, 2010