Agent Based Visualization of *C. Elegans* Embryogenesis at Cellular Resolution

-by Kison Osborne

<u>Introduction</u>:

The main goal of this project is to be able to accurately simulate the embryonic stages of the *C. Elegans*, using RepastHPC as a way to enhance computing through parallelization and a way to visualize the data generated by the RepastHPC code. The ultimate end goal of the project is be able to learn more about diseases that plague human beings.

<u>Background</u>:

C. Elegans is a primitive multicellular organism (worm) that shares many important biological characteristics that arise as complications within human beings.[1] It begins as a single cell and then undergoes a complex embryogenesis to form a complete animal. Using experimental data, the early stages of life of the cells are simulated by computers. The goal of this project is to use this simulation to compare the embryogenesis stage of *C. Elegans* cells with that of human cells. Since the simulation involves the manipulation of many files and large amounts of data, the power provided by supercomputers and parallel programming is required.

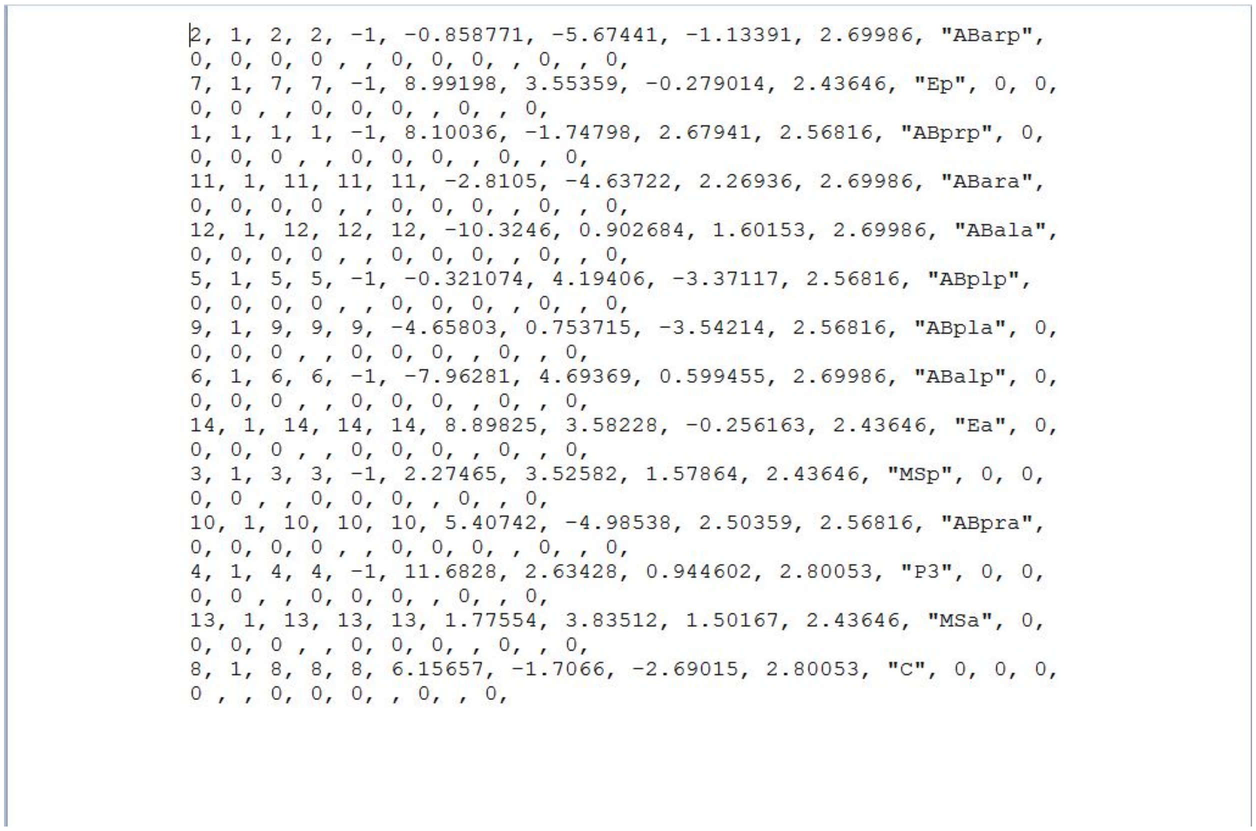
<u>Current Status</u>:

Thus far, I have managed to take the files generated through by the old Netlogo code and the new RepastHPC code (I call them Nuclei files) and parse them for the data that is most relevant for the visualization (such as location, size, and name) through the use of python scripts. This parsed data is written to point3d files, which VisIt can read and produce images of. Depending on the python script used to parse the Nuclei files I may implement other features into the visualization, such as cell tracking and coloring each cell depending on their parent cell.

I have also started to create a graphical user interface (GUI) that will streamline the RepastHPC code, python code and VisIt display into one user-friendly interface.

<u>Python Scripts</u>:

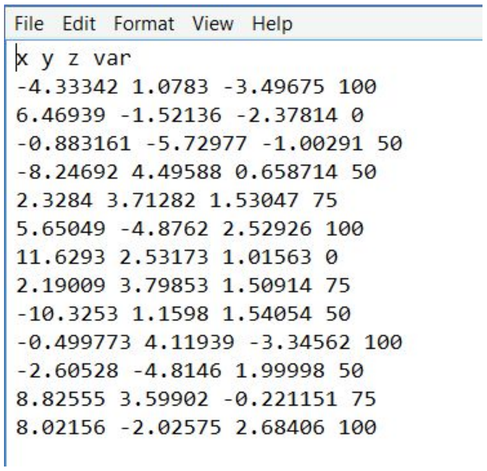
The files that come from the RepastHPC code is unable to be read easily by VisIt for visualization. These nuclei files (as I refer to them) are formatted such that there is a lot of information that is unimport when it comes to the visualization of that cell when using VisIt. The nuclei files are formatted as shown, where each line of information represents one cell and each column (separated by commas) represents a certain quality of the cell.



```
2, 1, 2, 2, -1, -0.858771, -5.67441, -1.13391, 2.69986, "ABarp",
0, 0, 0, 0 , , 0, 0, 0, , 0, , 0,
7, 1, 7, 7, -1, 8.99198, 3.55359, -0.279014, 2.43646, "Ep", 0, 0,
0, 0 , , 0, 0, 0, , 0, , 0,
1, 1, 1, 1, -1, 8.10036, -1.74798, 2.67941, 2.56816, "ABprp", 0,
0, 0, 0 , , 0, 0, 0, , 0, , 0,
11, 1, 11, 11, 11, -2.8105, -4.63722, 2.26936, 2.69986, "ABara",
0, 0, 0, 0 , , 0, 0, 0, , 0, , 0,
12, 1, 12, 12, 12, -10.3246, 0.902684, 1.60153, 2.69986, "ABala",
0, 0, 0, 0 , , 0, 0, 0, , 0, , 0,
5, 1, 5, 5, -1, -0.321074, 4.19406, -3.37117, 2.56816, "ABplp",
0, 0, 0, 0 , , 0, 0, 0, , 0, , 0,
9, 1, 9, 9, 9, -4.65803, 0.753715, -3.54214, 2.56816, "ABpla", 0,
0, 0, 0 , , 0, 0, 0, , 0, , 0,
6, 1, 6, 6, -1, -7.96281, 4.69369, 0.599455, 2.69986, "ABalp", 0,
0, 0, 0 , , 0, 0, 0, , 0, , 0,
14, 1, 14, 14, 14, 8.89825, 3.58228, -0.256163, 2.43646, "Ea", 0,
0, 0, 0 , , 0, 0, 0, , 0, , 0,
3, 1, 3, 3, -1, 2.27465, 3.52582, 1.57864, 2.43646, "MSp", 0, 0,
0, 0 , , 0, 0, 0, , 0, , 0,
10, 1, 10, 10, 10, 5.40742, -4.98538, 2.50359, 2.56816, "ABpra",
0, 0, 0, 0 , , 0, 0, 0, , 0, , 0,
4, 1, 4, 4, -1, 11.6828, 2.63428, 0.944602, 2.80053, "P3", 0, 0,
0, 0 , , 0, 0, 0, , 0, , 0,
13, 1, 13, 13, 13, 1.77554, 3.83512, 1.50167, 2.43646, "MSa", 0,
0, 0, 0 , , 0, 0, 0, , 0, , 0,
8, 1, 8, 8, 8, 6.15657, -1.7066, -2.69015, 2.80053, "C", 0, 0, 0,
0 , , 0, 0, 0, , 0, , 0,
```

As stated before, not all of this information is needed to visualize the cells. For the method I use, the only information needed for each cell is the cell's validity, x coordinate, y coordinate, z coordinate, size, and name. This information is represented by the 2nd, 6th, 7th, 8th, 9th, and 10th columns respectively in the nuclei file. Now that I had an idea of what information I would need, I needed to decide what kind of file could be read by VisIt and incorporated the most important of this data. What I eventually decided on was the Point3d file format.

Point3d files are usually used to show information about points on some type of 3d grid. The files are formatted similarly to the nuclei files where each line in the file represents one point (cell in our case) and each column represents an aspect of that cell. Unlike the nuclei files, the point3d files only consist of four columns of information. The first three columns are reserved for the cell’s x, y, and z coordinates while the fourth/final column is left to hold that cell’s variable. This variable can be used to illustrate anything about the cell while the first three columns will always represent the cell coordinates. For my purposes, I used this variable to show the cell’s size or color. Last but not least, this type of file is able to be easily read and displayed by VisIt.



```
x y z var
-4.33342 1.0783 -3.49675 100
6.46939 -1.52136 -2.37814 0
-0.883161 -5.72977 -1.00291 50
-8.24692 4.49588 0.658714 50
2.3284 3.71282 1.53047 75
5.65049 -4.8762 2.52926 100
11.6293 2.53173 1.01563 0
2.19009 3.79853 1.50914 75
-10.3253 1.1598 1.54054 50
-0.499773 4.11939 -3.34562 100
-2.60528 -4.8146 1.99998 50
8.82555 3.59902 -0.221151 75
8.02156 -2.02575 2.68406 100
```

To extract data from the nuclei files and store it into point3d files, I needed to use python scripts. To be exact, two python scripts. One script allowed for the implementation of cell tracking, while the other allowed each cell in each file to be colored differently based on its parent, these features will be explained more later.[1]

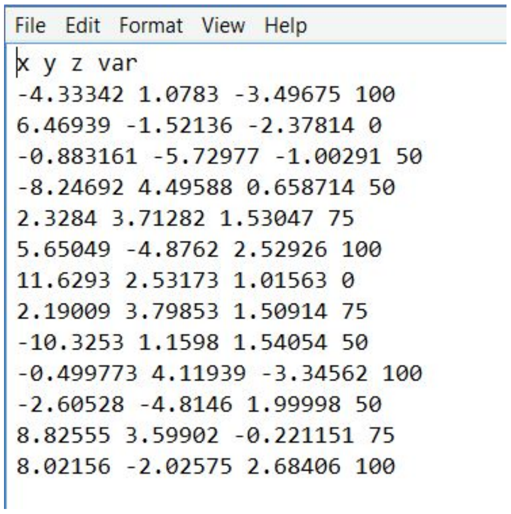
This script takes in as input a division table (which is basically a list of all the cells that will ever be created in the simulation), a location for the point3d files to be stored and the name of a cell, all provided by the user. The script takes the given cell name and searches for it in the division table. When it is found, the program takes the name of the parent and children of the cell (which are given by the division table) and puts them in a list with the starting cell. The program then uses the 'Find_Children' and 'Find_Ancestor' functions to find all of the cells that preceded the starting cell and all the cell that spawn from it. Once that is done, the program loops through all of the given nuclei files and proceeds to transform each one into a point3d file. While doing this, the script gives each cell in each file a value based on if it is in the list with the starting cell. This essentially makes it so that when the point3d files are eventually displayed all of the starting cell's children and ancestors are shown differently from the other cells. This allows the user to track the lineage of a particular cell throughout the simulation.

The other python script mentioned uses very similar techniques. Only in this script, each of the starting cells in the simulation are given their own value, which is then passed down to its children. This makes it so that each cell in the simulation will be displayed in one out of *n* colors (where *n* is the number of starting cells), illustrating what cell it originally spawned from at the start of the simulation.

<u>VisIt</u>:

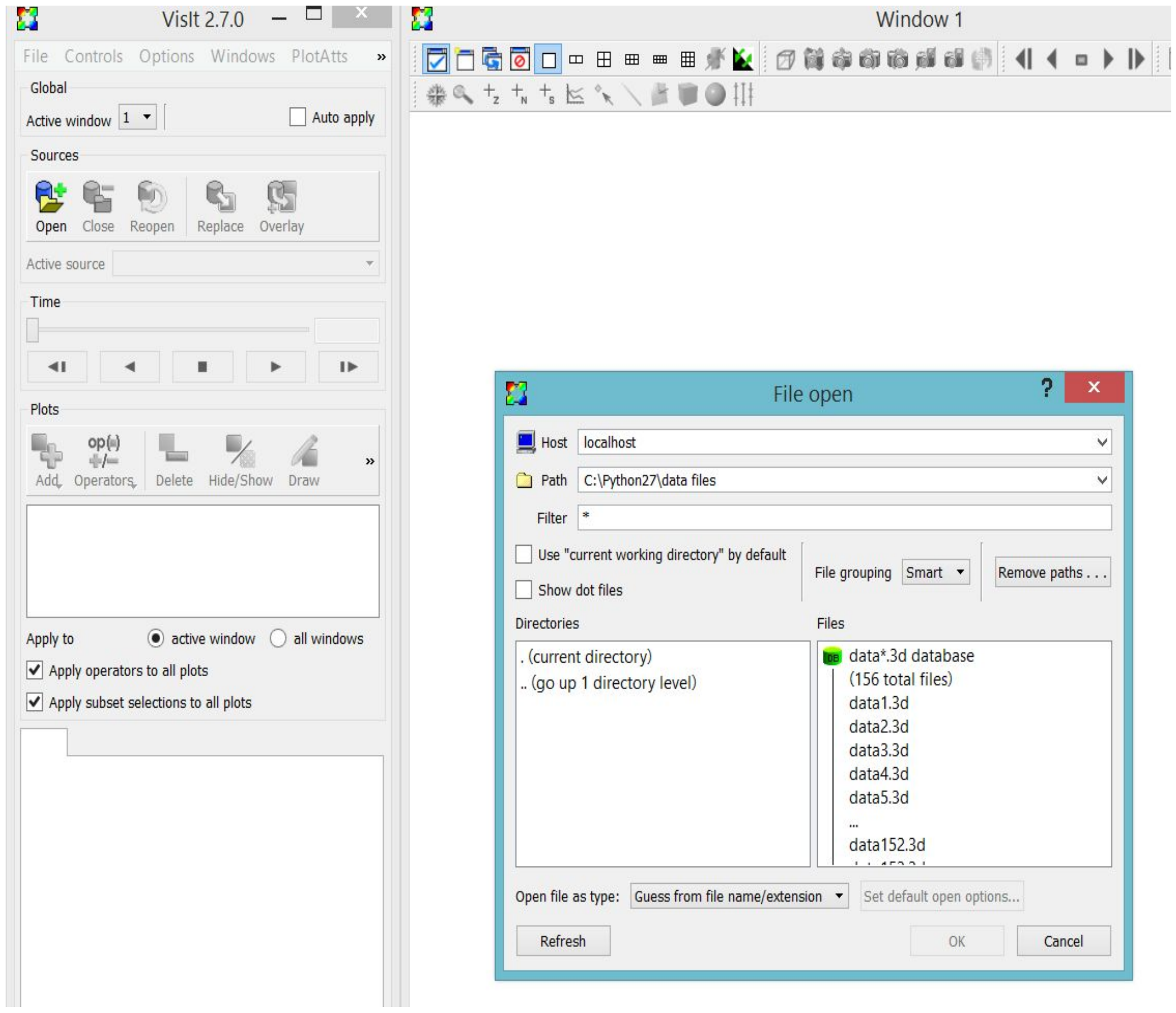
VisIt is an open-source program that can take in many different file types and display the data in said files in different ways (graphs, 3d models, maps, etc). To do this, VisIt reads the information in the file(s) that is(are) being opened depending on their file type and displays is depending on the plot and settings that the user sets for their data. This manual will show how to open a database (a bunch of files) in VisIt, set up a plot, and display through VisIts visualization window.

As mentioned before, VisIt can open many different file types but for this tutorial I will be using Point3d files (which are formatted as shown). This type of file represents the data for various points. Each line represents a point and the first three columns represents that points x, y, and z coordinates respectively. The fourth column represents a variable given to that point to show some quality (such as size). In this particular example, the 'var' column represents the color that point will be displayed as when displayed by VisIt.



```
x y z var
-4.33342 1.0783 -3.49675 100
6.46939 -1.52136 -2.37814 0
-0.883161 -5.72977 -1.00291 50
-8.24692 4.49588 0.658714 50
2.3284 3.71282 1.53047 75
5.65049 -4.8762 2.52926 100
11.6293 2.53173 1.01563 0
2.19009 3.79853 1.50914 75
-10.3253 1.1598 1.54054 50
-0.499773 4.11939 -3.34562 100
-2.60528 -4.8146 1.99998 50
8.82555 3.59902 -0.221151 75
8.02156 -2.02575 2.68406 100
```

When opening VisIt, you'll be greeted with two windows. One is the graphical user interface(GUI) window, where you will set your plot and the other is the visualization window (where the plots display). To open a file for display, click 'Open' in VisIts GUI window. Then, in the window that opens, navigate to the file you wish to open and click ‘OK’. To open a database, follow the same steps except you should make sure that the ‘File Grouping’ option in the window that appears when you click ‘Open’ is set to smart. Also, make sure that the group of you want to open are in the same folder and are named similarly.


VisIt 2.7.0
File Controls Options Windows PlotAtts »
Global
Active window 1
Auto apply
Sources
Open Close Reopen Replace Overlay
Active source
Time
Plots
Add Operators Delete Hide/Show Draw »
Apply to active window all windows
Apply operators to all plots
Apply subset selections to all plots
Window 1
File open
Refresh
OK
Cancel
Set default open options...
Remove paths . . .

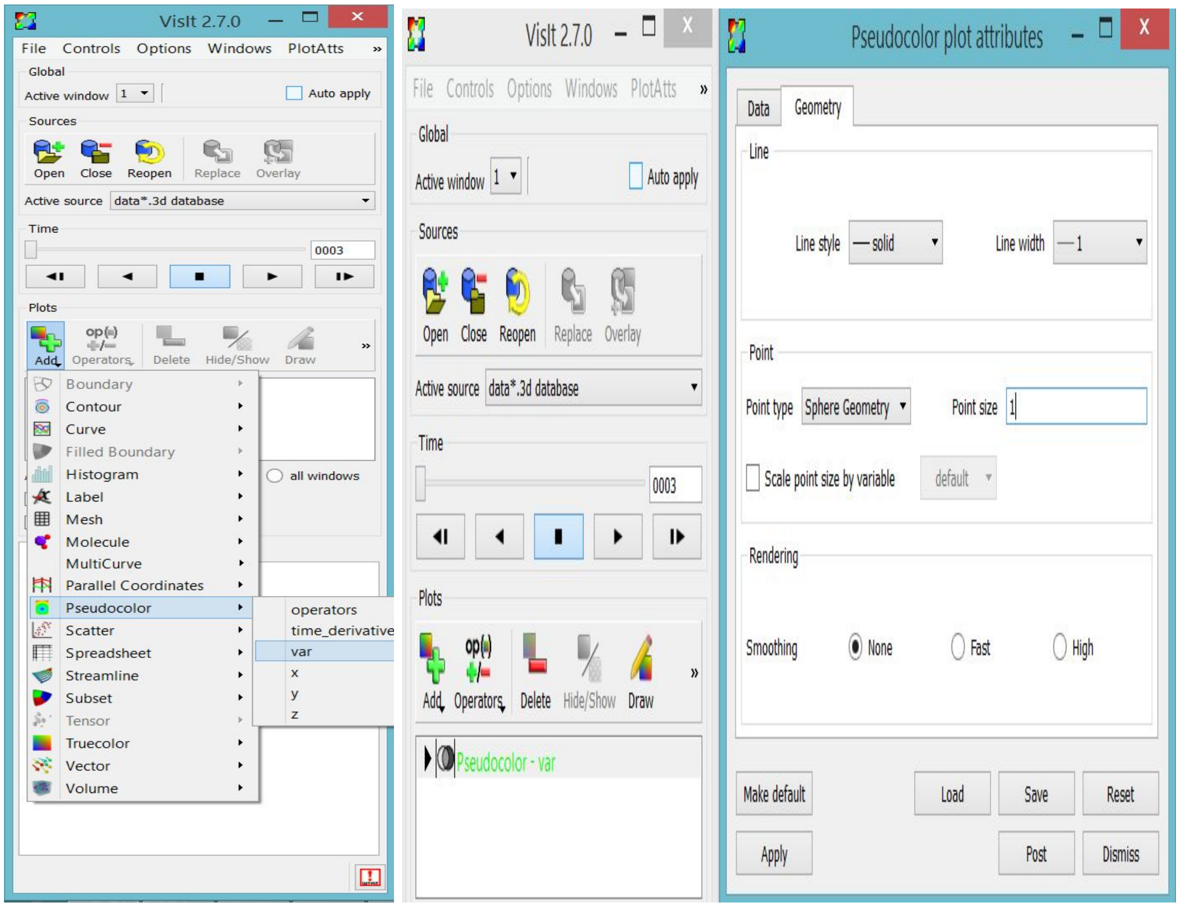
Host: localhost
Path: C:\Python27\data files
Filter: *
Use "current working directory" by default: [ ]
Show dot files: [ ]
File grouping: Smart

Directories
. (current directory)
.. (go up 1 directory level)

Files
data*.3d database
(156 total files)
data1.3d
data2.3d
data3.3d
data4.3d
data5.3d
...
data152.3d

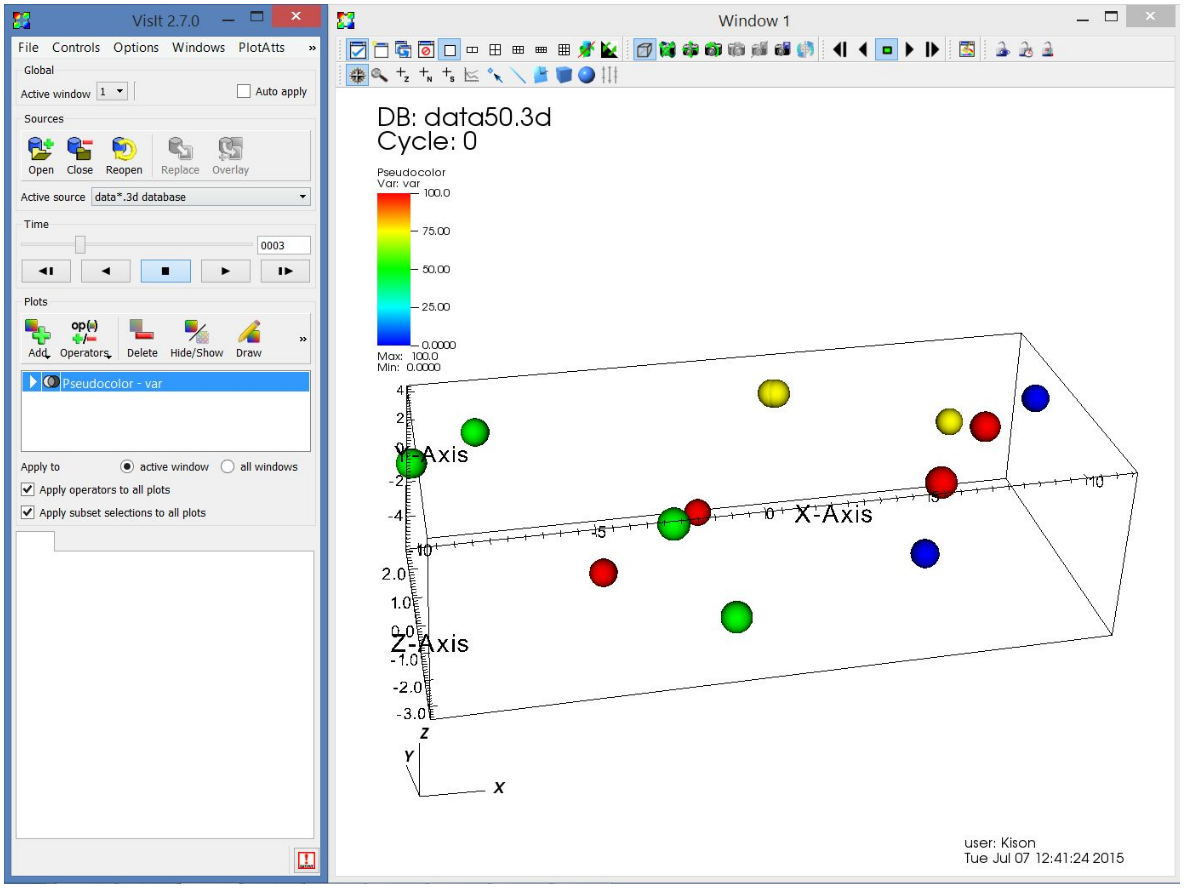
Open file as type: Guess from file name/extension

To add a plot to display the information stored in the file(s), simply click the ‘Add’ button in the GUI window. This brings up a list of numerous plots you can add, but for this tutorial we will add a pseudocolor plot. To this scroll down to ‘Pseudocolor’ and for this example we will click ‘vars’ in the menu that pops up to ensure that the plot that displays includes our points’ variable. After this is done, you should notice some text appear in the box under the ‘Add’ button in the GUI window. Clicking on the arrow will drop down a list of all the plots associated with your file (in this case all that would appear is ‘Pseudocolor’). Double clicking on the text will bling up that plots options for you to customize to your liking.



Active window 1
Auto apply
Active source data*.3d database
Time 0003
Pseudocolor - var
Line style — solid
Line width — 1
Point type Sphere Geometry
Point size 1
Scale point size by variable default
Smoothing [✓ None] Fast High

To draw your plot is as easy as clicking ‘Draw’ on the VisIt GUI window. If you opened a database you can use the arrows on the GUI window to navigate between the various files. You can also add operators to your plots to change its various aspects. This can be done by following similar steps to opening a plot, but you would click ‘Operators’ instead of ‘Add’.

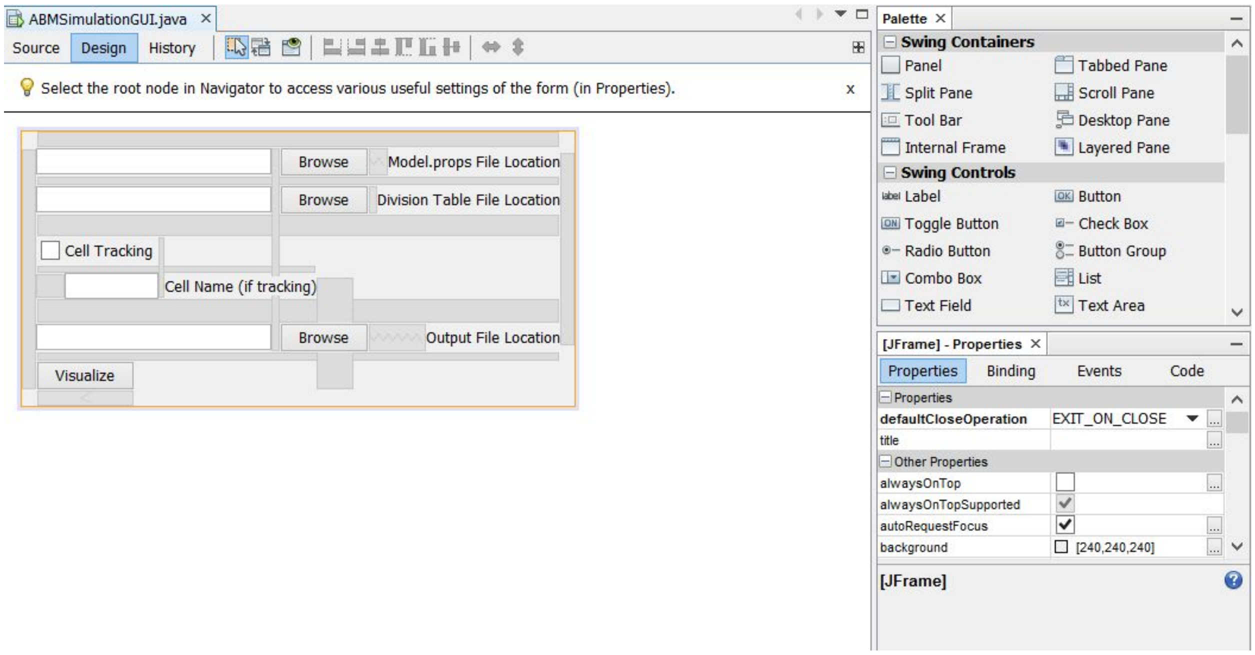

VisIt 2.7.0
File Controls Options Windows PlotAtts
Global
Active window 1
Auto apply
Sources
Open Close Reopen Replace Overlay
Active source data*.3d database
Time
0003
Plots
Add Operators Delete Hide/Show Draw
Pseudocolor - var
Apply to active window all windows
Apply operators to all plots
Apply subset selections to all plots
Window 1
DB: data50.3d
Cycle: 0
Pseudocolor
Var: var
100.0
75.00
50.00
25.00
0.0000
Max: 100.0
Min: 0.0000
Y-Axis
X-Axis
Z-Axis
user: Kison
Tue Jul 07 12:41:24 2015

<u>Netbeans IDE</u>:

So far the process of getting a VisIt display is something like this: running RepastHPC to create nuclei files, taking those nuclei file and run a python script to create point3d file, then finally opening VisIt and it display the point3d files. This can be a bit tedious to do this it involves running three different programs and a bit of managment. This is where the Netbeans IDE comes into play. Netbeans is used to create a java graphical user interface to streamline the process of running the RepastHPC simulation and displaying the results to VisIt.

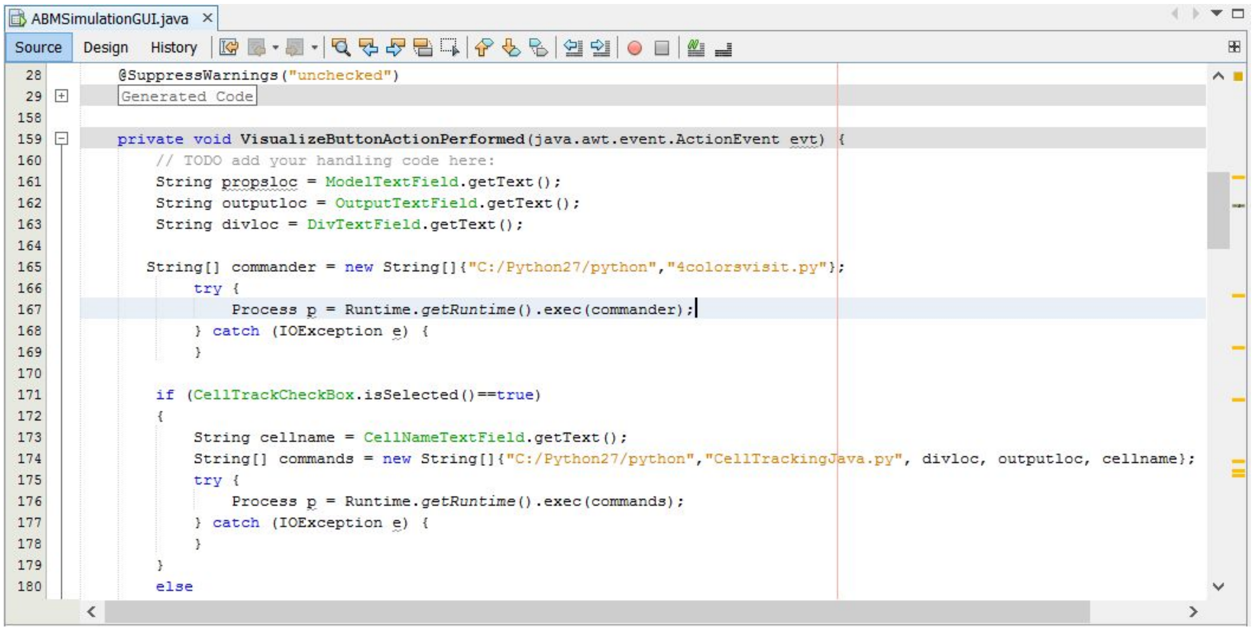
The reason for picking netbeans to create the GUI for now is simple: I know very little about java programming. Netbeans allows a user to create a java GUI without having to worry about the java code that goes into the actual design of the GUI. This allows the programmer to focus more on what the GUI actually does rather than what it may look like.

Netbeans has two major components: a design component and a source component. The design component allows the user to drag and drop the components he/she wants for the GUI into place and name variables however they wish. Netbeans then sets up the class and variable declarations based on that the user's design of the GUI. The source component is where the programmer can implement what the GUI actually does. For example: in the GUI I created for the this project, there is only one button(for now). This button, when pressed, will take the text that is typed into the three main text fields and store them in variables. It then checks to see if the 'Cell Tracking' check box is selected. If it is, the program takes the text in the 'Cell Name' text box and stores it into another variable. The program then calls the appropriate python script (which are explained above) based on the user's choice. Lastly, the program executes a terminal command that will run VisIt's viewer window with the appropriate files, which will display to the screen. This function allows the user to create a set of point3d files and display them without having to open python or VisIt. It should be noted that the code that Netbeans generates for the user cannot be edited from the source component of the Netbeans GUI.



Select the root node in Navigator to access various useful settings of the form (in Properties).

Properties	
defaultCloseOperation	EXIT_ON_CLOSE
title	
Other Properties	
alwaysOnTop	☐
alwaysOnTopSupported	✓
autoRequestFocus	✓
background	☐ [240,240,240]



```
    @SuppressWarnings("unchecked")
    Generated Code

    private void VisualizeButtonActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        String propsloc = ModelTextField.getText();
        String outputloc = OutputTextField.getText();
        String divloc = DivTextField.getText();

       String[] commander = new String[]{"C:/Python27/python","4colorsvisit.py"};
            try {
                Process p = Runtime.getRuntime().exec(commander);
            } catch (IOException e) {
            }

        if (CellTrackCheckBox.isSelected()==true)
        {
            String cellname = CellNameTextField.getText();
            String[] commands = new String[]{"C:/Python27/python","CellTrackingJava.py", divloc, outputloc, cellname};
            try {
                Process p = Runtime.getRuntime().exec(commands);
            } catch (IOException e) {
            }
        }
        else
```

I am sure that this is not all that Netbeans is capable of, I am still learning to use the software as well as coding in java. The GUI is incomplete as well. As of now, there is no connection between the RepastHPC code and this interface, a problem I plan to remedy very soon.

Next Steps:

As I stated before, I plan to complete the GUI in order to create a user-friendly interface to run this simulation. Also, I plan to reduce the complexity of my python scripts so there little to no issues when larger data sets are run.

Resources

RepastHPC Tutorial and Download: http://repast.sourceforge.net/repast_hpc.php

Visit Tutorial and Manuals: https://wci.llnl.gov/simulation/computer-codes/visit/manuals

Caenorhabditis Genetics Center, College of Biological Sciences, University of Minnesota. "What is C. elegans?". *College of Biological Sciences, University of Minnesota*. July 22, 2015. https://www.cbs.umn.edu/research/resources/cgc/what-c-elegans

Acknowledgements

- Ben Ramsey (University of Tennessee)
- Dali Wang (Oak Ridge National Laboratory)
- Kwai Wong (University of Tennessee)
- Scott Simmerman (Oak Ridge National Laboratory)
- Zhirong Bao (Memorial Sloan Kettering Cancer Center)
- John Murphy (Argonne National Laboratory)
- Chung Ng (Morehouse College)
- National Science Foundation
- Joint Institute For Computational Sciences

Appendix

1) Python Script 4 Color:

```
import sys, os, shutil

def java(divloc, outputloc):
        def ParseDivTable():
        DivTable = []
        with open(divloc) as inDiv:
        for line in inDiv:
        data = line.split('" ')
        parent = data[0]
        parent = parent[1:]
        daughter1 = data[1]
        daughter1 = daughter1[1:]
        daughter2 = data[2]
        daughter2 = daughter2[1:]
        listofdata = [parent, daughter1, daughter2]
        DivTable.append(listofdata)
        DivTable = DivTable[1:]
        return DivTable

        def Find_Children(User_Cell, DivTable, Children):
        search_cells = []
        search_cells.append(User_Cell)
        Children.append(User_Cell)
        while search_cells != []:
        search_cell = search_cells[0]
        for cells in DivTable:
        if search_cell == cells[0]:
                search_cells.append(cells[1])
                search_cells.append(cells[2])
                Children.append(cells[1])
                Children.append(cells[2])
        search_cells.remove(search_cell)

        def Highlight_Cells(DivTable, blue, yellow, green, red):
        rootdir = './nuclei'
        counter = 1
        folder = outputloc

        for files in os.walk(rootdir):
        for nfiles in files:
        if nfiles != rootdir and nfiles != []:
                for file in nfiles:
                sctr = str(counter)
```

```
            with open(folder + 'data' + sctr + '.3d', 'a') as infile:
            infile.seek(0)
            infile.truncate()
            infile.write("x y z var\n")
            with open(rootdir + '/' + file) as fp:
                    for line in fp:
                    data = line.split(', ')
                    #cID = data[0]
                    valid = data[1]
                    #pID = data[2]
                    #nID = data[3]
                    #dID = data[4]
                    xCOR = data[5]
                    yCOR = data[6]
                    zCOR = data[7]
                    #size = data[8]
                    name = data[9]
                    if '"' in name:
                    name = name[1:len(name) - 1]
                    if "Nuc" in name:
                    valid = 0
                    if valid == "1":
                    if name in blue:
                    color = 0
                    elif name in yellow:
                    color = 75
                    elif name in green:
                    color = 50
                    elif name in red:
                    color = 100
                    infile.write(xCOR + " ")
                    infile.write(yCOR + " ")
                    infile.write(zCOR + " ")
                    infile.write(str(color) + "\n")
            counter = counter + 1
for letter in divloc:
if letter == "\"":
letter = "/"
for letter in outputloc:
if letter == "\"":
letter = "/"


DivTable = ParseDivTable()
Starting_Cells = ['P2', 'EMS', 'ABa', 'ABp']
blue = []
yellow = []
green = []
```

```
        red = []

        Children = []
        Find_Children('P2', DivTable, Children)
        blue = Children
        Children = []
        Find_Children('EMS', DivTable, Children)
        yellow = Children
        Children = []
        Find_Children('ABa', DivTable, Children)
        green = Children
        Children = []
        Find_Children('ABp', DivTable, Children)
        red = Children
        Highlight_Cells(DivTable, blue, yellow, green, red)

java(sys.argv[1], sys.argv[2])
```

Python Script: Cell Tracking:

```
import sys, os, shutil

def java(divloc, outputloc, User_Cell):
        def ParseDivTable():
        DivTable = []
        with open(divloc) as inDiv:
        for line in inDiv:
        data = line.split('" ')
        parent = data[0]
        parent = parent[1:]
        daughter1 = data[1]
        daughter1 = daughter1[1:]
        daughter2 = data[2]
        daughter2 = daughter2[1:]
        listofdata = [parent, daughter1, daughter2]
        DivTable.append(listofdata)
        DivTable = DivTable[1:]
        return DivTable


        def Find_Ancestors(User_Cell, DivTable, Ancestors):
        found = False
        search_cell = User_Cell
        while search_cell != 'ABp' and search_cell != 'EMS' and search_cell != 'P2' and search_cell !=
'ABa':
        found = False
```

```
for cells in DivTable:
if not found:
        if search_cell == cells[1] or search_cell == cells[2]:
        found = True
        search_cell = cells[0]
        Ancestors.append(search_cell)

def Find_Children(User_Cell, DivTable, Children):
search_cells = []
search_cells.append(User_Cell)
while search_cells != []:
search_cell = search_cells[0]
for cells in DivTable:
if search_cell == cells[0]:
        search_cells.append(cells[1])
        search_cells.append(cells[2])
        Children.append(cells[1])
        Children.append(cells[2])
search_cells.remove(search_cell)

def Highlight_Cells(User_Cell, Ancestors, Children):
rootdir = './nuclei'
counter = 1
folder = outputloc

for files in os.walk(rootdir):
for nfiles in files:
if nfiles != rootdir and nfiles != []:
        for file in nfiles:
        sctr = str(counter)
        with open(folder + 'data' + sctr + '.3d', 'a') as infile:
        infile.seek(0)
        infile.truncate()
        infile.write("x y z var\n")
        with open(rootdir + '/' + file) as fp:
                for line in fp:
                data = line.split(', ')
                #cID = data[0]
                valid = data[1]
                #pID = data[2]
                #nID = data[3]
                #dID = data[4]
                xCOR = data[5]
                yCOR = data[6]
                zCOR = data[7]
                color_v = data[8]
                name = data[9]
                if "" in name:
                    name = name[1:len(name) - 1]
                    if "Nuc" in name:
                    valid = 0
                    if valid == "1":
                    if name in Ancestors:
                    color_v = 100
                    elif name == User_Cell:
                    color_v = 50
                    elif name in Children:
                    color_v = 50
                    else:
                    color_v = 0
                    infile.write(xCOR + " ")
                    infile.write(yCOR + " ")
                    infile.write(zCOR + " ")
                    infile.write(str(color_v) + "\n")
                counter = counter + 1

        for letter in divloc:
        if letter == "\"":
        letter = "/"
        for letter in outputloc:
        if letter == "\"":
        letter = "/"

        DivTable = ParseDivTable()
        Ancestors = []
        Children = []

        Find_Ancestors(User_Cell, DivTable, Ancestors)

        Find_Children(User_Cell, DivTable, Children)

        Highlight_Cells(User_Cell, Ancestors, Children)

java(sys.argv[1], sys.argv[2], sys.argv[3])
```