Diffusion-Limited Aggregation

Physics 3300, Weber State University, Spring Semester, 2012

In this project you will again use Monte Carlo methods to study a physical system. The system is extremely simple and you don't need to know any physics at all to understand it.

Imagine that a large number of small solid particles are suspended in a fluid, moving around randomly due to thermal motions ("Brownian motion"). In the center of the region is a cluster of these particles, all stuck together. Whenever a new particle touches the cluster, it sticks to it. After the cluster has grown to a fairly large size, what will be its shape?

Your task is to create a computer simulation to answer this question.

The Algorithm

While you could allow the particles to move around continuously, it's much simpler (and *pretty* accurate for our purposes) to restrict the particles to sites on a square lattice. Also, for simplicity, please take the space to have only two dimensions rather than three. Then the basic data structure for this simulation can be a two-dimensional array of boolean variables: `true` for each occupied site and `false` for each unoccupied site.

An algorithm to carry out the simulation then proceeds as follows. Start with a single "seed" particle that's fixed in the middle of the lattice, and let other particles wander toward it one at a time, forming a single cluster that grows outward from the seed. Keep track of the maximum "radius" of the cluster (that is, the maximum distance of any of its particles from the initial seed). Start each new wandering particle at a random point along a circle that lies just outside this radius. Move the wandering particle in a two-dimensional random walk, with equal probabilities of moving up, down, left, or right during each time step. After each move, test whether the wandering particle's location is adjacent to an already-occupied site and if it is, leave the particle there (by setting the lattice value to `true` at its current location).

Program Design

Call your program `DLA.java`. It should extend the `Canvas` class to display the current state of the lattice, using the background color for unoccupied sites and a contrasting color for permanently occupied sites. Your `paint` method will need to loop through all the lattice sites, drawing only those that are occupied. It should then use a different color to draw the currently wandering particle (whose coordinates are presumably stored in a couple of integer variables).

Write your `paint` method first, and test it using temporary "dummy" values for the boolean lattice variables and the current coordinates of the wandering particle. For now, draw each lattice site as a square that's at least a few pixels across, so you can easily see each square. Plan to reduce the square size (and correspondingly, increase the size of the simulated space) later.

Once your `paint` method is tested and working, write a method (call it `addParticle`) to add a single wandering particle to the cluster. This method should implement the algorithm described above, initializing the particle's location and then moving it randomly until it finds a permanent home adjacent to an already-occupied site. To make the simulation run faster, you can test whether the particle has wandered outside the initial circle and if so, move it back in to the circle (without changing its angle relative to the center). To calculate the angle unambiguously you'll need the `Math.atan2` function (look it up in the Java API reference, and make a note in your lab report describing how it works).

Create a separate thread to call the `addParticle` method repeatedly (until the cluster grows too large), and use the `Thread.sleep` method to slow down the simulation so you can see each move during the particle's random walk. Implementing the logic of the algorithm will require some thought and care, and probably some debugging. Take your time, and keep a record of your successes and failures in your lab report.

Once the algorithm seems to be working (or perhaps before), you'll want to add some buttons to control the simulation. At a minimum, create a "Run/Pause" button and also a checkbox to disable the sleeping (and thus speed up the simulation). (If you've never made a checkbox before, look it up in the Java API reference and again make notes in your lab report.) Optionally, you could add a button to clear the lattice so you can start over without quitting the program.

In general, try to keep a record in your lab report of anything you learn or figure out that you think would be helpful to someone else who is trying to reproduce what you did.

Results

Now increase the lattice size to several hundred across, and decrease the plotted size of each site to a single pixel. Run the simulation and admire the beautiful random shapes that it creates. Make a printout of one of your DLA clusters, by first making a screen capture (have someone show you how if you don't already know) and then printing from whatever software can open the screen capture file. Be sure to set the background color to white, to save toner! (You can, of course, use a different background color for screen viewing.)

How can we accurately describe this cluster's intricate shape—especially its "den-

sity"? If the particles had arranged themselves along a straight line, we would say they had made a *one*-dimensional shape. If the particles had filled the whole space, we would say the shape was *two*-dimensional. What we have *here* is something in between, and we can characterize it using a non-integer number for the "dimension."

To calculate this "dimension," you need to know the number of occupied sites as a function of the distance from the center. Create an array to hold this information (with the array's index representing the distance from the center, in units of the lattice spacing, rounded to the nearest integer). Add code to your `runParticle` method to increment the appropriate element of this array whenever a particle adheres to the cluster. Then, when the cluster is complete, print out all the elements of this array to the Terminal window. (You might want to add a button to activate this data dump.)

Once all this is working and you've collected the data for a large cluster, copy and paste the data into a spreadsheet. (In the instructions that follow, I'll assume that you're using Excel. I don't know whether other spreadsheet programs have the required features and if so, what the appropriate commands are called.) Add a column to the spreadsheet for the total number of particles *within* each radius (calculated appropriately), and make a log-log scatter plot of this quantity as a function of radius. The data should fall approximately along a straight line. Edit the data series in the plot to eliminate any values near the ends that don't lie near the line (due to edge effects and such). Then use the "Add Trendline" command to fit the data with a *power law* function, which should appear as a straight line on the log-log plot. Be sure to tell it to display the equation.

The exponent of this power-law fit is called the *fractal dimension* or *Hausdorff dimension* of the cluster. Explain carefully, in your lab report, what the exponent of this power-law fit would be if the cluster were (a) a simple straight line and (b) a completely filled space. Then explain why your result makes intuitive sense.

Repeat the simulation and analysis a couple of times, to see how consistent the fractal dimension is from one run to another.

Spend a little time on the Internet reading about fractals and the concept of fractal dimension. Be sure to browse through the marvelous Wikipedia article titled **List of fractals by Housdorff dimension**. Make a few notes on what you learn. For additional fractal fun, check out Vi Hart's doodling videos—especially the one titled Binary Trees (vihart.com/doodling).

Your finished lab report should include a running log of your notes (organized and legible), a printed image of your most beautiful fractal cluster, and a printout of your spreadsheet and graph, showing the calculation of your cluster's fractal dimension (for one simulation run). In addition, please clean up your source code for the benefit of human readers and submit it by email (as an attachment).