

Diffusion-Limited Aggregation

Kinetic Growth Phenomena

Many structures in nature are formed by random addition of particles. Starting with a seed particle, the structure can capture freely moving (kinetic) particles that happen to collide with it. This growth mechanism gives rise to fascinating shapes such as ice crystals, snowflakes, frost patterns on a window, and dendritic (tree-like) structures like the metallic cluster and Lichtenberg figure shown in Fig. 1.

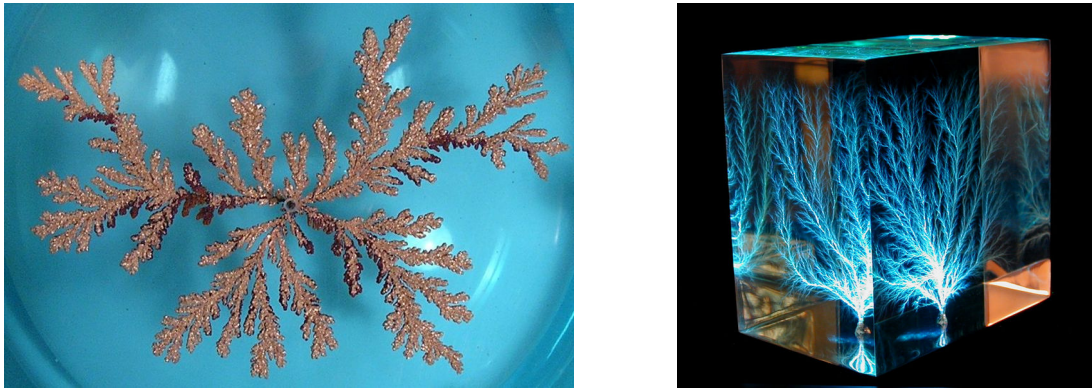


Figure 1: Left: Dendritic growth of metallic copper from CuSO_4 solution, http://en.wikipedia.org/wiki/Diffusion-limited_aggregation. Right: Lichtenberg figure caused by dielectric breakdown in acrylic http://en.wikipedia.org/wiki/Lichtenberg_figure.

These structures have fractal properties: they are not simple geometric objects like lines or surfaces or volumes with integer dimension, but lie somewhere in between!

Random Walks and Cluster Growth

It would be too complicated to simulate dendritic cluster growth by solving the equations of motion for all of the molecules involved. Surprisingly, dendritic growth can be simulated quite well by very simple random models based on random walks. Witten and Sander[1] the concept of diffusion-limited aggregation to model the growth of dendritic structures in various experimental systems. Niemeyer et al.[2] modeled the growth of Lichtenberg figures (dendritic structures due to dielectric breakdown) using random walks.

Diffusion is a process in which particles move through a system making numerous collisions with other particles. At each collision, a particle's velocity changes randomly.

If a particle were to move with a constant velocity, then its displacement would increase linearly with time

$$\mathbf{r} = \mathbf{r}_0 + \mathbf{v}t . \quad (1)$$

This is called ballistic motion.

In diffusive motion, the displacement increases much more slowly:

$$\langle |\mathbf{r} - \mathbf{r}_0|^2 \rangle \sim Dt, \quad (2)$$

where $\langle \dots \rangle$ denotes average, and D is the *diffusion constant*.

It can be shown mathematically that the random walking of an ensemble of independent walkers on a lattice reduces to the partial differential equation

$$\frac{\partial \rho(\mathbf{x}, t)}{\partial t} = D \nabla^2 \rho(\mathbf{x}, t) \quad (3)$$

for diffusion of the density n of walkers in the limit that the lattice spacing and time step tend to zero.

Approximating diffusion using random walkers

To model the diffusive motion of kinetic particles in two dimensions, define a simple random walk as follows:

- Assume that time is discrete $t = 0, \delta t, 2\delta t, \dots$
- Assume that space is discrete and that the particle is located at the vertices of a two-dimensional square lattice.
- Start the particle at some lattice site at $t = 0$.
- At each time step, choose one of the four nearest neighbor sites at random and move the particle to that site.

Diffusion limited aggregation algorithm

The diffusive motion of a random walker is “limited” or stopped by an encounter with the growing dendrite. This is modeled by the following algorithm:

- Start with a lattice with one particle fixed at a central lattice point. This will be the *seed* from which the dendritic crystal will grow.
- Repeat the following steps until the crystal is large enough:
 - Add a particle at a random empty site on the boundary of the lattice.
 - Allow the particle to diffuse through the lattice until it strikes the crystal and sticks to it. Diffusion is modeled using a two-dimensional random walk.

C++ Program for DLA Growth

_____ Program 1: <http://www.physics.buffalo.edu/phy410-505/topic2/dla.cpp> _____

```
#include <algorithm>
#include <cmath>
#include <cstdlib>
```

```

#include <fstream>
#include <iostream>
#include <vector>
#include <string>
using namespace std;

#include "random.hpp"
using namespace cpl;

Random rg; // random number generator object

const double pi = 4 * atan(1.0); // value of pi

int L; // number of lattice sites in x or y
vector< vector<bool> > cluster; // 2-d array of sites belonging to cluster

```

The two-dimensional boolean array `cluster` is maintained to keep track of which lattice sites belong to the cluster.

The following are some useful functions to

- test whether an x, y value belongs to the lattice
- test whether a site is occupied or not
- compute the distance of a site from the origin
- add site x, y to the cluster

Program 1: <http://www.physics.buffalo.edu/phy410-505/topic2/dla.cpp>

```

bool onLattice(int x, int y) { // test whether site is on lattice
    int i = y + (L - 1) / 2;
    int j = x + (L - 1) / 2;
    return i >= 0 && i < L && j >= 0 && j < L;
}

bool occupied(int x, int y) { // test whether site is occupied
    if (!onLattice(x, y))
        return false;
    int center = (L - 1) / 2;
    return cluster[center + y][center + x];
}

double rCenter(int x, int y) { // distance from center of lattice
    double rSq = x * x + y * y;
    return sqrt(rSq);
}

double rMax; // max distance of cluster sites from center

```

```

void addSite(int x, int y) {          // add site and adjust rMax
    int center = (L - 1) / 2;
    cluster[center + y][center + x] = true;
    double r = rCenter(x, y);
    if (r > rMax)
        rMax = r;
}

```

The following function initializes the lattice by

- estimating the diameter of a cluster with a guess for the fractal dimension
- setting the linear size of the lattice 50% larger than this diameter
- creating two dimensional boolean array with all elements = `false`
- adding a single cluster particle at the center of the lattice

Program 1: <http://www.physics.buffalo.edu/phy410-505/topic2/dla.cpp>

```

void initialize(int n) {              // cluster and lattice for n sites

    // choose lattice large enough for n sites
    double area = n;                 // area of fractal disk
    double df = 1.6;                 // under-estimate of fractal dimension
    double r =
        pow(area / pi, 1 / df);      // over-estimate of disc radius
    L = int(3 * r);                   // make L/2 50% larger than r
    if (L % 2 == 0) ++L;              // make L odd
    cout << " Using " << L << " x " << L << " lattice" << endl;

    // create two dimensional cluster array and add site at origin
    vector< vector<bool> > empty(L, vector<bool>(L, false));
    cluster = empty;
    rMax = 0;
    addSite(0, 0);
}

```

When the cluster is small and a random walker is started at the boundary of the system, the walker can waste a lot of CPU time wandering aimlessly around the lattice before colliding with the cluster.

The following function attempts to optimize the simple random walk algorithm:

- Start new walkers at r_{start} away from the origin with random polar angle $0 \leq \theta \leq 2\pi$.
- If the walker wanders farther than $1.5 \times r_{\text{start}}$ start a new walker.
- Increase r_{start} as the cluster grows, keeping it at least 5 units larger than the maximum cluster size.

- When the walker is far from the cluster take steps of length 2 to speed up the walk, and decrease the step size back to 1 as the walker approaches the cluster.

_____ Program 1: <http://www.physics.buffalo.edu/phy410-505/topic2/dla.cpp> _____

```
bool randomWalk() {
    // attempt to add a site using random walk
    // return true if attempt succeeds
    int nStart = min(5, (L-1)/2); // small number of spacings from cluster
    double rStart = rMax + nStart; // distance of starting point from origin
    double rStop = 1.5 * rStart; // stop walker if it gets too far away

    double theta = 2 * pi * rg.rand();
    int x = int(rStart * cos(theta));
    int y = int(rStart * sin(theta));

    while (true) {

        // check whether walker is too far away
        double r = rCenter(x, y);
        if (r >= rStop)
            break;

        // check whether walker is adjacent to an occupied site
        if (onLattice(x, y) && !occupied(x, y) &&
            (occupied(x + 1, y) || occupied(x - 1, y) ||
             occupied(x, y + 1) || occupied(x, y - 1) ))
        {
            addSite(x, y);
            return true;
        }

        // set the step size
        int stride = 1;
        if (r > rMax + 5) // more than 5 steps from cluster
            stride = 2; // double step size

        // take a random step
        const int EAST = 0, NORTH = 1, WEST = 2, SOUTH = 3;
        switch (int(4 * rg.rand())) {
            case EAST:
                x += stride;
                break;
            case NORTH:
                y += stride;
                break;
            case WEST:
                x -= stride;
                break;
            case SOUTH:
                y -= stride;
                break;
            default:

```

```

        break;
    }
}

return false;
}

void writeCluster(string filename) {
    ofstream file(filename.c_str());
    int xyMax = (L - 1) / 2;
    for (int x = -xyMax; x <= xyMax; x++)
        for (int y = -xyMax; y <= xyMax; y++)
            if (occupied(x, y))
                file << x << '\t' << y << '\n';
    file.close();
}

int main() {

    cout << " Random Walk Simulation of Diffusion Limited Aggregation\n"
         << " -----\n"
         << " Enter number of particles to simulate: ";
    int n;
    cin >> n;

    // set random number generator seed
    rg.set_seed_time();
    cout << " Using " << rg.get_algorithm()
         << " and seed " << rg.get_seed() << endl;

    initialize(n);           // create lattice and add site at center

    int sites = 1;           // cluster has one occupied site
    int failedWalks = 0;     // count number of failed walks
    while (sites < n) {
        if (randomWalk()) {  // walk succeeded
            ++sites;
            if (sites % 10 == 0)
                cout << sites << " " << flush;
        } else
            ++failedWalks;
    }

    cout << " Done\n Number of failed walks = " << failedWalks << endl;
    string name("dla.data");
    writeCluster(name);
    cout << " DLA cluster saved in file " << name << endl;
}

```

Fractals and Fractal Dimension

The term *fractal* was coined by Benoit Mandelbrot to describe geometrical objects and natural phenomena that appear to be intermediate between points and lines, or lines and areas, or areas and volumes, i.e., which appear to have noninteger dimensions.

Mathematically, a fractal is an object whose *geometric dimension* is different from its *Hausdorff dimension*.

The *geometric dimension* of an object can be described intuitively: a point or a collection of points has dimension zero, a line or collection of lines has dimension 1, an area dimension 2, etc.

The *Hausdorff dimension* of an object is computed by finding the minimum number $N(r)$ of *open balls* of radius r required to cover it. In one-dimensional space a ball is simply a line of length r , in two-dimensional space, it is a circle of radius r , in three-dimensional space, it is a sphere of radius r , etc. The Hausdorff dimension is then computed as the limit

$$d_H = \lim_{r \rightarrow 0} \frac{\log N(r)}{\log(1/r)}. \quad (4)$$

Examples of Fractals

The Cantor Set: Start with a straight line of length 1 and remove the middle one-third of the line. Do this repeatedly with the resulting segments. After the n -th repetition we have

$$r = \frac{1}{3^n}, \quad N(r) = 2^n, \quad d_H = \lim_{n \rightarrow \infty} \frac{\log 2^n}{\log 3^n} = \frac{\log 2}{\log 3} = 0.63092975 \dots \quad (5)$$

The Koch Curve: Start with a straight line of length 1, the diameter of a circle of radius $r = 1/2$ (the ball). Replace the middle one-third section with a “tent” constructed of two lines of length $1/3$. Do this repeatedly with each of the 4 resulting segments. After the n -th repetition we have

$$r = \frac{1}{2 \cdot 3^n}, \quad N(r) = 4^n, \quad d_H = \lim_{n \rightarrow \infty} \frac{\log 4^n}{\log 2 + \log 3^n} = \frac{\log 4}{\log 3} = 1.2618595 \dots \quad (6)$$

The Sierpinski Gasket: Start with an equilateral triangle inscribed in a circle of unit radius (the ball). Partition it into 4 equal equilateral triangles and remove the one in the middle. Do this repeatedly with each of the 3 resulting triangles. After the n -th repetition we have

$$r = \frac{1}{2^n}, \quad N(r) = 3^n, \quad d_H = \lim_{n \rightarrow \infty} \frac{\log 3^n}{\log 2^n} = \frac{\log 3}{\log 2} = 1.5849625 \dots \quad (7)$$

These simple fractals exhibit the property of self similarity in an obvious way.

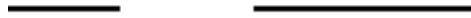
Mass Fractal Dimension

How does the mass of an object of uniform density scale with its linear size? Consider simple geometric objects with linear size r , as in Fig. 2

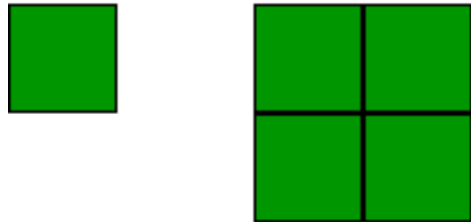
The *mass fractal dimension* of an object is defined by the equation

$$m(r) \sim r^{d_M}. \quad (8)$$

- Doubling the length of a line segment doubles its mass
 $m \sim r^1$



- Doubling the side of a square quadruples its mass
 $m \sim r^2$



- Doubling the side of a cube octuples its mass
 $m \sim r^3$

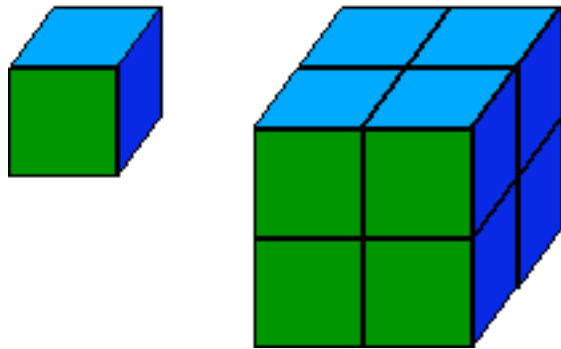


Figure 2: The mass dimension of simple geometric objects.

Homework Problem

Use <http://www.physics.buffalo.edu/phy410-505/topic2/dla.cpp> to measure the fractal dimension of DLA clusters. Generate a large sample of clusters with some fixed number of occupied sites. Compute the average mass (number of particles) inside a circle of radius r centered on the seed particle. Plot mass inside $m(r)$ versus radius and determine the fractal dimension by fitting your data. Compare your result with that of Witten and Sander[1].

References

- [1] T.A. Witten and L.M. Sander, “Diffusion-Limited Aggregation, a Kinetic Critical Phenomenon”, Phys. Rev. Lett. **47**, 1400 (1981), <http://link.aps.org/doi/10.1103/PhysRevLett.47.1400>
- [2] L. Niemeyer, L. Pietronero, and H.J. Wiesmann, “Fractal Dimension of Dielectric Breakdown”, Phys. Rev. Lett. **52**, 1033 (1984), <http://link.aps.org/doi/10.1103/PhysRevLett.52.1033>.