Cluster growth model

- Apply update rule to unassigned cells that have activated neighbours

![Diagram showing the cluster growth model with transition rules for 'On' and 'Off' states. The diagram illustrates the probabilistic transitions between states, with 1 - p and p probabilities for transitioning from 'Off' to 'On' and vice versa, respectively.]
Percolation transition

- stationary final configurations on a 400x400 torus

\[ p = 0.59 \]
\[ p = 0.59274 \]
\[ p = 0.6 \]
\[ p = 0.7 \]
Simple data structure

- 3-state grid of cells stored as conventional 2D array

```c
const size_t halfL = 200;
const size_t L = 2*halfL;
enum cell_t { UNASSIGNED=2, ON=1, OFF=0 };
char grid[L][L];

void initialize_grid(void)
{
    for (size_t i = 0; i < L; ++i)
        for (size_t j = 0; j < L; ++j)
            grid[i][j] = UNASSIGNED;
    grid[halfL][halfL] = ON;
}
```
Simple updates

- Each sweep scales as $L^2$
- As many as $L$ sweeps for the On wavefront to propagate across the system

double prob;
void sweep_grid(void)
{
    for (size_t i = 0; i < L; ++i)
        for (size_t j = 0; j < L; ++j)
            if (grid[i][j] == UNASSIGNED and
                (grid[(i+1)%L][j] == ON or
                 grid[(L+i-1)%L][j] == ON or
                 grid[i][(j+1)%L] == ON or
                 grid[i][(L+j-1)%L] == ON))
                grid[i][j] = (Rand() < prob ? ON : OFF);
Redundant data structure

- Trade off memory for algorithmic efficiency

```cpp
const size_t halfL = 200;
const size_t L = 2*halfL;
enum cell_t { UNASSIGNED = 2, ON = 1, OFF = 0 }; char grid[L][L];

class coord
{ public:
    size_t x;
    size_t y;
    coord(size_t x_, size_t y_) : x(x_), y(y_) {}
};

#include <queue>
using std::queue;
queue<coord> perim;
```
void initialize_grid(void) {
    for (size_t i = 0; i < L; ++i)
        for (size_t j = 0; j < L; ++j)
            grid[i][j] = UNASSIGNED;
    grid[halfL][halfL] = ON;
    perim.push(coord(halfL+1,halfL));
    perim.push(coord(halfL-1,halfL));
    perim.push(coord(halfL,halfL+1));
    perim.push(coord(halfL,halfL-1));
}
Grow the perimeter

```cpp
double prob;
void sweep_grid(void)
{
    while (!perim.empty())
    {
        const coord c = perim.front(); perim.pop();
        const size_t i = c.x;
        const size_t j = c.y;
        if (grid[i][j] == UNASSIGNED)
            if (Rand() < prob)
                {
                    grid[i][j] = ON;
                    if (grid[(i+1)%L][j] == UNASSIGNED) perim.push(coord((i+1)%L,j));
                    if (grid[(L+i-1)%L][j] == UNASSIGNED) perim.push(coord((L+i-1)%L,j));
                    if (grid[i][(j+1)%L] == UNASSIGNED) perim.push(coord(i,(j+1)%L));
                    if (grid[i][(L+j-1)%L] == UNASSIGNED) perim.push(coord(i,(L+j-1)%L));
                }
            else
                grid[i][j] = OFF;
    }
}
```
Container classes

- sequence containers
  - O(1) element access
  - O(N) insertion
- associative containers
  - O(log N) lookup
  - O(1) insertion

```cpp
#include <vector>
using std::vector

#include <deque>
using std::deque

#include <set>
using std::set

#include <map>
using std::map
```

list, slist, multiset, multimap, stack, queue, ...
Floating point numbers

- Floating point numbers have the form
  \[ \pm f \times b^e \]

- The adjustable radix point allows for calculation over a wide range of magnitudes

- Floating point numbers are limited by the number of bits used to represent the fraction and exponent
C++ floating point types

- on the Intel architecture . . .

For a `float`, there is a 8-bit exponent field and a 23-bit fraction field.

For a `double`, there is a 11-bit exponent field and a 52-bit fraction field.
Accuracy of FP arithmetic

- FP arithmetic is by its nature inexact
- Important always to think about accuracy: should we believe the computer’s final answer?
- FP multiplication and division are relatively safe
- FP subtraction of nearly-equal quantities (or addition of equal magnitude, opposite sign quantities) can dramatically increase the relative error
Potential dangers

- FP operations can yield both “overflow” and “underflow”
- Additional notes on the class web site will explore the Infinity (Inf) and Not-a-Number (NaN) error states
Potential dangers

› Associativity breaks down: \((u + v) + w \neq u + (v + w)\)

› The following 8-digit decimal floating point operation has a 5% relative error depending on the order in which operations are performed:

\[
(11111113. + -11111111.) + 7.5111111 = 2.0000000 + 7.511111 = 9.511111
\]

\[
11111113. + (-11111111. + 7.511111) = 11111113. + -11111103. = 10.000000
\]
Potential dangers

- The distributive law

\[ u \times (v + w) \neq (u \times v) + (u \times w) \]

can also fail badly:

\[
20000.000 \times (-6.0000000 + 6.0000003) = 20000.000 \times 0.000000300000000
= 0.0060000000
\]

\[
(20000.000 \times -6.0000000) + (20000.000 \times 6.0000003) = -120000.00 + 120000.01
= .01000000
\]
Potential dangers

- It can easily occur that \( 2(u^2 + v^2) < (u + v)^2 \)
- Hence, variance is not guaranteed to be positive
- Naively calculating the standard deviation can lead to your taking the square root of a negative number

\[
\sigma = \frac{1}{n} \sqrt{n \sum_{k=1}^{n} x_k^2 - \left( \sum_{k=1}^{n} x_k \right)^2}
\]
Potential dangers

- Many common mathematical relations no longer hold ...

\[(x + y)(x - y) = x^2 - y^2\]

\[\sin^2 \theta + \cos^2 \theta = 1\]
“Carefully written programs”

- technical meaning: programs that are numerically correct
- this is very difficult to guarantee!
“Carefully written programs”

1. \( (x + y)/2 \)
2. \( x/2 + y/2 \)
3. \( x + ((y - x)/2) \)
4. \( y + ((x - y)/2) \)

Which formula should we use to compute the average of x and y?
“Carefully written programs”

1. \( \frac{x + y}{2} \)

2. \( \frac{x}{2} + \frac{y}{2} \)

3. \( x + \frac{(y - x)}{2} \)

4. \( y + \frac{(x - y)}{2} \)

May raise an overflow if \( x \) and \( y \) have the same sign
“Carefully written programs”

1. \( \frac{x + y}{2} \)

2. \( \frac{x}{2} + \frac{y}{2} \)  
   
   May degrade accuracy but is safe from overflows

3. \( x + \frac{(y - x)}{2} \)

4. \( y + \frac{(x - y)}{2} \)
“Carefully written programs”

1. \((x + y)/2\)
2. \(x/2 + y/2\)
3. \(x + ((y - x)/2)\)
4. \(y + ((x - y)/2)\)

May raise an overflow if \(x\) and \(y\) have opposite signs.
“Carefully written programs”

- you want functions that are robust
- give some thought to the rare or extreme cases that may cause your function to misbehave
- avoid overflows and underflows
- avoid undefined operations, e.g., \( \sqrt{-1}, \frac{0}{0} \)
"Carefully written programs"

- Example: roots of the quadratic equation, \( ax^2 + bx + c \)
- According to the usual formula, \( x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \)
- Problem can arise if \( b^2 \gg |4ac| \) so that \( \sqrt{b^2 - 4ac} \approx |b| \)
- Cancellation can lead to catastrophic loss of significant digits:
  \[
  \frac{b \pm |b|}{2a} \approx \frac{0}{0}
  \]
“Carefully written programs”

› One possible workaround: use exact algebraic manipulations on a per case basis

\[
x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} = \frac{-2c}{b + \sqrt{b^2 - 4ac}}
\]

\[
x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a} = \frac{2c}{-b + \sqrt{b^2 - 4ac}}
\]

cancellation

\[
cancellation
\]

\[
no\ cancellation
\]


```
#include <cassert>
#include <cmath>
using std::sqrt; // square root
using std::fabs; // absolute value

void quadratic_roots(double a, double b, double c,
                      double &x1, double &x2)
{
    const double X2 = b*b-4*a*c;
    assert(X2 >= 0.0);
    const double X = sqrt(X2);
    const double Ym = -b-X;
    const double Yp = -b+X;
    const double Y = (fabs(Ym) > fabs(Yp) ? Ym : Yp);

    x1 = 2*c/Y;
    x2 = Y/(2*a);
}
```
“Carefully written programs”

› Example: norm of a complex number

\[ z = x + iy, \quad |z| = \sqrt{x^2 + y^2} \]

› Avoid possible overflow when squaring terms:

\[
|z| = x\sqrt{1 + r^2}, \quad r = \frac{y}{x}, \quad \text{if } |y| < |x|
\]

\[
|z| = y\sqrt{1 + r^2}, \quad r = \frac{x}{y}, \quad \text{if } |x| < |y|
\]
"Carefully written programs"

- Example: the sinc function $\frac{\sin(x)}{x}$
- possible problems as $x \to 0$
- workaround: explicit power series expansion

\[
\frac{\sin(x)}{x} = x - \frac{x^3}{3!} + \frac{x^5}{5!} + \cdots = 1 - \frac{x^2}{3!} + \frac{x^4}{5!} + \cdots
\]
Arbitrary precision arithmetic

› scheme for performing operations on integers and rational numbers with no rounding, e.g.,

\[
\frac{2153}{9932} + \frac{871}{7362} = \frac{12250579}{36559692}
\]

› available in symbolic manipulation environments (Maple, Mathematica) and “bignum” libraries

› implemented in software; limited by system memory