

Forest fire model

The forest fire model of P. Bak, K. Chen, and C. Tang, *Phys. Lett. A* **147**, 297 (1990), is a simple *probabilistic* cellular automaton with complex behavior that mimics the spreading of fires in a forest.

The forest is modeled as a square region of side L with a regular lattice of L^2 cells or sites which represent trees. A cell can have three states:

1. A living tree, which is colored green.
2. A tree on fire, which is colored yellow.
3. A dead tree, which is colored black.

Initializing the forest

At time $t = 0$, the forest is populated using two parameters:

- Each cell has a tree with probability p_t , i.e., generate a random number r in the unit interval $0 < r < 1$, and color the site green if $r < p_t$, or black otherwise.
- If the cell has a tree (if $r < p_t$ in the previous step) then set the tree on fire with probability p_f , i.e., generate a uniform deviate r' and color the tree yellow if $r' < p_f$.

Local update rule

At time t , each cell in the forest will be in one of three states, green, yellow, or black. The forest at time $t + 1$ is determined by the following rules:

1. If a tree is green, check its four nearest neighbors. If any neighbor is yellow (on fire), then the tree catches fire and is colored yellow.
2. If a tree is yellow (on fire), then it dies and is colored black.
3. If a tree is dead (black), then a new tree is grown with probability p , i.e., generate a uniform deviate r and color the cell green if $r < p$.

Note that rules 1 and 2 are *deterministic*, but rule 3 is *probabilistic*: this model is therefore a *probabilistic* cellular automaton.

OpenGL Forest File Program

```
// OpenGL Forest File Program

#include <cstdlib>
#include <iostream>
using namespace std;
#include <GL/glut.h>

int L = 200;           // number of trees in x and y
double pt;             // initial probability to have a tree
double pf;             // probability this tree is on fire
double p;              // probability to grow a tree
enum {LIVE, FIRE, DEAD}; // 3 states of cells
int **tree;            // the trees
int **oldTree;         // previous for synchronous updating
GLubyte *image;        // for drawing forest image

void allocate() {
    static int oldL = 0;
    if (L != oldL) {
        if (tree != 0) {
            for (int i = 0; i < oldL; i++) {
                delete [] tree[i];
                delete [] oldTree[i];
            }
        }
    }
}
```

```
        delete [] tree;
        delete [] oldTree;
        delete [] image;
    }
    tree = new int* [L];
    oldTree = new int* [L];
    for (int i = 0; i < L; i++) {
        tree[i] = new int [L];
        oldTree[i] = new int [L];
    }
    image = new GLubyte [9 * L * L];
}

void initialize() {
    allocate();
    for (int i = 0; i < L; i++)
        for (int j = 0; j < L; j++) {
            if (rand() / double(RAND_MAX) < pt) {
                tree[i][j] = oldTree[i][j] = LIVE;
                if (rand() / double(RAND_MAX) < pf) {
                    tree[i][j] = oldTree[i][j] = FIRE;
                }
            } else {
                tree[i][j] = oldTree[i][j] = DEAD;
            }
        }
}
```

```
void takeStep() {  
  
    // save state of the forest  
    for (int i = 0; i < L; i++)  
        for (int j = 0; j < L; j++)  
            oldTree[i][j] = tree[i][j];  
  
    // probabilistic automaton update rule  
    for (int i = 0; i < L; i++)  
        for (int j = 0; j < L; j++) {  
            switch (oldTree[i][j]) {  
                case LIVE:  
                    // use periodic boundary conditions  
                    if ( oldTree[i > 0 ? i - 1 : L - 1][j] == FIRE ||  
                        oldTree[i < L - 1 ? i + 1 : 0][j] == FIRE ||  
                        oldTree[i][j > 0 ? j - 1 : L - 1] == FIRE ||  
                        oldTree[i][j < L - 1 ? j + 1 : 0] == FIRE )  
                        tree[i][j] = FIRE;  
                    break;  
                case FIRE:  
                    tree[i][j] = DEAD;  
                    break;  
                case DEAD:  
                    if (rand() / double(RAND_MAX) < p)  
                        tree[i][j] = LIVE;  
                    break;  
                default:  
                    break;  
            }  
        }  
}
```

```
    }  
    glutPostRedisplay();  
}  
  
void reshape(int w, int h) {  
    glViewport(0, 0, w, h);  
    glMatrixMode(GL_PROJECTION);  
    glLoadIdentity();  
    gluOrtho2D(0, w, 0, h);  
    glMatrixMode(GL_MODELVIEW);  
    glLoadIdentity();  
}  
  
GLubyte colors[3][3] = { {0, 255, 0}, {255, 255, 0}, {0, 0, 0} };  
  
void display() {  
    glClear(GL_COLOR_BUFFER_BIT);  
    for (int i = 0; i < L; i++)  
        for (int j = 0; j < L; j++)  
            for (int k = 0; k < 3; k++)  
                image[3 * (L * i + j) + k] = colors[tree[i][j]][k];  
    glRasterPos2i(0, 0);  
    glDrawPixels(L, L, GL_RGB, GL_UNSIGNED_BYTE, image);  
    glutSwapBuffers();  
}  
  
int main(int argc, char *argv[]) {  
    glutInit(&argc, argv);  
    if (argc > 1)
```

```

    L = atoi(argv[1]);
    cout << " Enter probabiltty p_t:  ";
    cin >> pt;
    cout << " Enter probabiltty p_f:  ";
    cin >> pf;
    cout << " Enter probabiltty p:  ";
    cin >> p;
    initialize();
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(L, L);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Two-dimensional forest fire automaton");
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glShadeModel(GL_FLAT);
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutIdleFunc(takeStep);
    glutMainLoop();
}

```

Results found by Bak et al.

Recall that the dynamics of the model depends on one parameter, namely the probability p for growing new trees from dead stumps. The system is more or less independent of the starting configuration. Bak et al. found the following results for this model:

- The system is characterized by a *correlation length*

$$\xi(p) \sim \frac{1}{p^\nu} ,$$

where $\nu \approx 1.0$ in this 2-dimensional model. (Bak et al. also studied the system in 3 dimensions.)

- If the correlation length is larger than the linear size of the system

$$\xi(p) > L ,$$

then the fires die out in a time $\sim L$. The growth of new trees from dead stumps is not large enough to sustain the fires.

- If the correlation length is smaller than the linear size of the system

$$\xi(p) < L ,$$

then the fires are sustained: the system reaches a critical steady state in which the growth of new trees feeds the fires with sufficient fuel to keep them burning for ever.

- The distribution of fires in the critical state is a *fractal*, i.e., an object with fractional dimension. In the 2-dimensional model, they found that the fractal dimension of the fire fronts $D \simeq 1.0$, i.e., the fronts are approximately linear; but in 3 dimensions, they found $D \simeq 2.5$, i.e., intermediate between a surface ($D = 2$) and a volume ($D = 3$) distribution.