Chemical sculpting

- **Topic:** kinetic Monte Carlo, surfaces, anisotropy
- **Task:**
  1. Implement a method for choosing an event from a list of event probabilities.
  2. Test your simulation and make sure you etch the surface of the material.
  3. Change removal probabilities and see how that affects the results.
  4. You can also try etching a cluster or adding sites that may not be etched.
- **Template:** etching.py
- **Further reading:**
  - https://en.wikipedia.org/wiki/Etching_(microfabrication)
  - https://en.wikipedia.org/wiki/Kinetic_Monte_Carlo
  - https://en.wikipedia.org/wiki/Surface_growth

etching.py

`etching.animate(history)` [source]

Animate the simulation.

Parameters:: **history** (*list*) – list of systems at different times

`etching.count_neighbors(i, j, grid)` [source]

Count the number of occupied neighbors for the lattice site (i,j).

The system is represented as a square lattice so the first neighbors of a given site are the 4 sites directly next to it left, right, up and down.

The second neighbors of a site are the 4 sites diagonally next to it: up-left, up-right, down-left, down-right.

This function counts how many of those sites are occupied (have the value 1).

The edges of the system wrap around. It means that for a site that is, say, on the left edge of the system, the left-side neighbor is a site on the right edge of the system.

Parameters::
- **i** (*int*) – row index
- **j** (*int*) – column index
- **grid** (*array*) – the system as an integer array

Returns:: number of first neighbors, number of second neighbors

Return type:: int, int

`etching.count_probabilities(grid)` [source]

Calculates, for each site, the probability that this particular site is the next to be removed.

The function loops over all occupied sites (i,j) and calculates the removal rate $r_{i,j}$ for each site.

The total removal rate is defined as

$$R = \sum_{i,j} r_{i,j}$$

and the probability that site (i,j) is the next to be removed, is

$$p(i,j) = \frac{1}{R} r_{i,j}.$$

Finally, the function lists the coordinates of all occupied sites in a single site vector S and constructs the corresponding vectors containing the site specific probabilities p and accumulated probabilities P.

The elements of vector S are pairs of integers, so you may have, e.g., $S_0 = (0,0), S_1 = (1,0), S_2 = (2,0)$ etc.

The elements of vector p are probabilities, so you may have, e.g., $p_0 = p(0,0), p_1 = p(1,0), p_2 = p(2,0)$ etc.

The elements of vector P are sums of probabilities,

$$P_n = \sum_{k=0}^{n} p_k.$$

So $P_0 = p_0, P_1 = p_0 + p_1, P_2 = p_0 + p_1 + p_2$ etc. Especially the probabilities must sum up to 1, so the last element of vector P is always 1.

Parameters:: **grid** (*array*) – the system as an integer array

Returns:: p, P, S

Return type:: array, array, array

`etching.draw(frame, history)` [source]

Draws the system for animation.

Parameters::
- **frame** (*int*) – index of the frame to draw
- **history** (*list*) – list of systems at different times

`etching.generate_system(lattice_size, cluster=False)` [source]

Creates the system.

The system is represented by a 2D $N \times N$ lattice where each lattice site may be either occupied (1) or unoccupied (0).

The system may be a surface, in which case one of its sides has a row of empty sites, or a cluster, in which case all sides have empty sites.

Parameters::
- **lattice_size** (*int*) – lattice size N
- **cluster** (*bool*) – if True, the system is a cluster

Returns:: the system as an integer array

Return type:: array

`etching.get_removal_rate(i, j, grid, unremovables=[])` [source]

Calculates the removal rate for a site.

For normal sites, the removal rate is evaluated based on how many 1st and 2nd neighbors the site has. These are calculated with `count_neighbors()`.

Certain sites may be listed as protected sites that may never be removed. Such sites mimic the use of protective masks.

> **Note**
>
> Edit to change the physics!

Parameters::
- **i** (*int*) – row index
- **j** (*int*) – column index
- **grid** (*array*) – the system as an integer array
- **unremovables** (*list*) – indices of sites that are never removed

Returns:: the removal rate

Return type:: float

`etching.main(lattice_size, n_steps, n_plots, show_as_you_go=False, cluster=False)` [source]

Main program.

Creates a system as a $N \times N$ square lattice, then removes atoms from it.

Parameters::
- **lattice_size** (*int*) – lattice size N
- **n_steps** (*int*) – the total number of atoms to remove
- **n_plots** (*int*) – the number of images to create
- **show_as_you_go** (*bool*) – If True, will show images of the system during simulation. If False, an animation is created at the end. Only set to True for short tests.
- **cluster** (*bool*) – if True, the system is a cluster

`etching.pick_random_event(grid)` [source]

Randomly choose the next event (removal of a site) to happen.

The random event is chosen as follows:

- The probability p for choosing each site is calculated with `count_probabilities()`.
- Also the accumulated probabilities P are calculated.
- A random number $R \in [0,1]$ is drawn.
- The event n for which $P_{n-1} < R \leq P_n$ is chosen. (For $n = 0$, it is enough that $R \leq P_0$.)
- The function returns the coordinates of the site for this event, S_n.

> **Note**
>
> This function is incomplete!

Parameters:: **grid** (*array*) – the system as an integer array

Returns:: coordinates (i,j) of the chosen site

Return type:: array

`etching.print_progress(step, total)` [source]

Prints a progress bar.

Parameters::
- **step** (*int*) – progress counter
- **total** (*int*) – counter at completion

`etching.randomly_remove_atom(grid)` [source]

Randomly choose one site for removal and make it unoccupied.

The site is chosen using `pick_random_event()`. The chosen site is marked as unoccupied by setting the grid value to 0.

Parameters:: **grid** (*array*) – the system as an integer array

`etching.show_system(grid)` [source]

Draws the system.

Parameters:: **grid** (*array*) – the system as an integer array