Menu Project. Fractal properties of self-avoiding walks and chains

random walks. Make an applet that draws 256-step random walks on the screen using horizontal and vertical steps one pixel in size (*i.e.,* one unit in the drawLine co-ordinates. Keep a list of the points (i, j) traversed. Your applet should use a thread so that it can be interrupted. Find the number of steps $M(3)$ within a square of length 3 centered on step number 128. For each of these walks, also determine the number of steps within a nested series of boxes: $M(5)$, $M(9)$, $M(17)$, $M(33)$, $M(65)$. Find the averages of these numbers using 100 independent random walks. Make the applet print the current averages whenever it is interrupted. Plot $M(L)/L^2$ versus L to judge whether the random walks show fractal scaling. If so, what is the apparent fractal dimension? The quantity $M(L)/L^2$ is the average *density* in the box. It gives a more revealing plot, because it makes it easier to see small differences from a uniform density.

kinetically self-avoiding walks. Modify the applet above by forbidding a new step which occupies a previous step of the walk. When such a step occurs, make another step in a different direction. It can happen that the walk gets trapped, so that no step is possible. If this happens before step number 256, throw away the whole walk and start over. Find the average $M(L)$ as above, and plot the average density versus L to determine any apparent fractal dimension. About what fraction of the time does the walk get trapped?

self-avoiding chain. Create an initial nonrandom walk in a zigzag configuration that starts from the origin, goes 10 steps to the right, goes down 3 steps, goes ten steps to the left, goes down 3 more steps, goes ten steps to the right, again etc, for a total of 256 steps. Pick one end at random, remove a step from that end, and add it to the other end at a randomly-chosen neighboring site. If this new step occupies another step of the walk, remove it and put it back where it came from. Continue in this way until the middle step of the walk has moved a distance 20 from its starting point. Note the number of attempted moves required. Compute the masses $M(L)$ for the middle step. Repeat the process for 100 walks in order to determine the averaged $M(L)$'s. Repeat the plot above and report on any differences in fractal scaling. Once the program is working, increase the number of steps in the walk and the number of walks averaged to obtain more accurate data. Compare the fractal properties of the kinetically self-avoiding walks and the self-avoiding chains.

tricks. The simplest way to see whether a given point was occupied is to check whether it was occupied by step 1,by step 2, by step 3, etc. But this is time consuming; it takes up to 256 queries. A faster method is to define an array $P(i, j)$ with the same dimensions as your applet window. When your random walk takes a step to point (i, j), add 1 to the element $P(i, j)$. If a step is removed from point (i, j), substract 1 from $P(i, j)$. Now you can determine whether (i, j) is occupied just by asking whether $P(i, j) = 0$. This is one query instead of 256.

For the self avoiding walk simulation, you have to modify your list of steps when you remove a step. Suppose you remove step 1. Then your list goes from 2 to 257. After many steps the list could grow very long. To avoid this, store your list as a ring structure, with a separate integer that marks the position of the first step. The previous entry in the list is the last step. After a few steps in your walk the first step will be *eg.* at position 3, of your list. The second step is at position 4, ... the 254th step is at position 256, the 255th step is at position 1, and the 256th step is at position 2. Then to remove the first step, you would just increment your marker from 3 to 4. To add another step to the end, you would put the position of the new last step as item 3 of your list.