Spatial binning and hashing

In the smoothed particle hydrodynamics simulation, particles interact only with those particles that are within a circle of radius h of them. In the naive reference code, a substantial fraction of the total time is spend finding which pairs of particles interact, and the cost of finding interacting pairs scales quadratically with the number of particles in the simulation. But the particles in the simulation never get too close together, and so any given particle will typically only interact with a bounded number of neighbors. We can therefore make the simulation much faster by checking for interactions only between particles that are close enough that they could conceivably interact.

One simple way to limit the number of interactions we check is to partition space into fixed-size bins. In particular, if we partition space into bins with side length $l \geq 2h$, then a particle in a particular bin can interact with at most the other particles in that bin and particles in three neighboring bins; see Figure 1

I do something very simple: I represent each particle as a structure[1], and use a linked list structure for each bin. The particle structure has the form

```
typedef struct particle_t {
  // Position, velocity, acceleration, etc.
  struct particle_t* next; // Next in bin
} particle_t;
```

and I have an array of head pointers `particle_t* bins[]`, one for each bin. Before computing interactions at each time step, I clear the `bins` pointers, and then re-build the bins with a loop of the form

```
for (int i = 0; i < n; ++i) {

    // Figure out bin for particle i
    int b = ...;

    // Add particle to the start of the list for bin b
    part[i].next = bins[b];
    bins[b] = part[i];
}
```

[1]I use parallel arrays in the reference code, but refactoring into an array of structures gives better performance. I hope you can explain why at this point in the class!

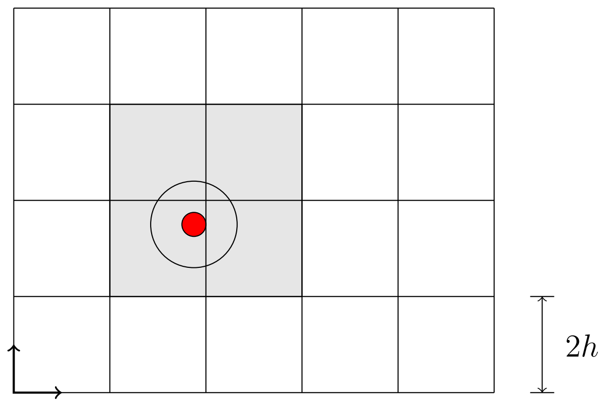

Figure 1: The red particle in the bin at zero-indexed coordinates $(1, 1)$ can only interact with particles in the grey bins at $(1, 1)$, $(1, 2)$, $(2, 1)$, and $(2, 2)$.

Then when I want to compute interactions for particle i, I only need to look up those particles in the bin where particle i resides, along with a few neighboring bins.

Of course, giving you this much code still leaves you with a number of practical issues to address, such as how to index the bins and how to keep using symmetry in the computations of the interactions between particles! And there's nothing that says that you have to structure your computation in the same way that I do. For example, you could eschew pointers entirely and just use a bucket sort algorithm[2] to re-order your particles at each step so that particles in the same bin are in contiguous array locations[3].

Partitioning space into bins that are $2h$ on a side is reasonable when a large fraction of the total volume is filled with particles, as is the case for the dam break simulation that we run by default. But this is not such a reasonable approach if only a small fraction of the volume in the computational domain is filled with particles. In such a case, we would not have a separate bucket for every bin in space. Instead, we would do *spatial hashing*, which is a fancy way of saying that we would index the buckets by a hash of the bin identifier. As long as the hash table is around the same size as the

[2] I'm happy to tell you about bucket sort if you come ask me, but you might be better of just doing a Google search (or looking in a favorite algorithms textbook).

[3] I actually do a sort in my code in order to improve locality, but only once every hundred steps or so

number of particles, we expect not to have too many collisions; and for any given particle, it is still easy to look up the hash buckets that would contain any particles with which it might interact. Note that I do not expect you to do spatial hashing for this assignment — simply having one bucket for each spatial bin is fine!