Computer Science II — Homework 9 — Geometric Hashing

Preliminaries

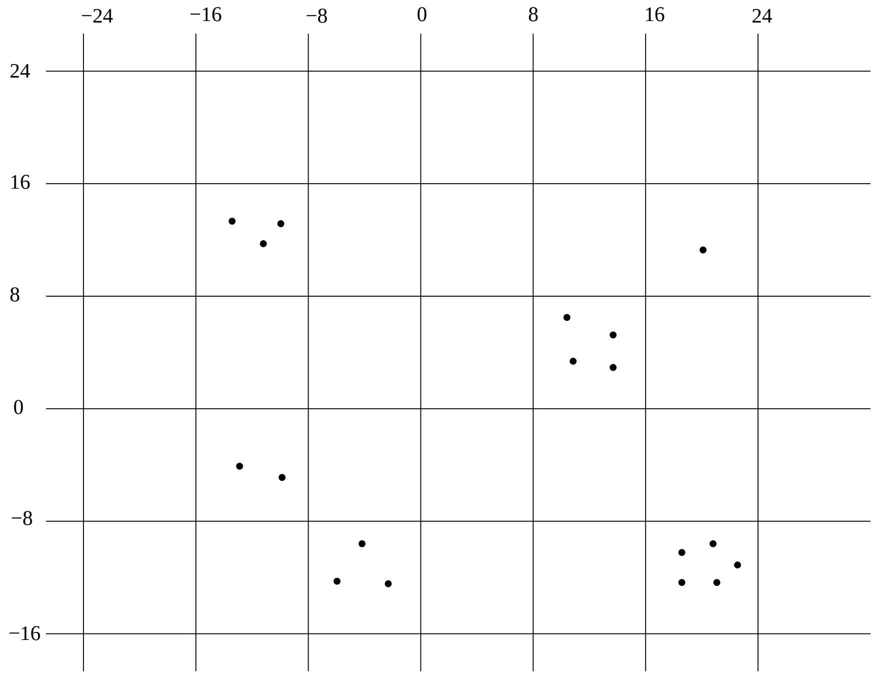
This assignment is due Friday April 25 at 11:59:59pm. It is worth 75 points toward your homework grade. See the end of the assignment for submission instructions.

Overview

Many applications in graphics, robotics, computer vision, machine learning and data mining require storing and querying a large number of data points having many dimensions. Since queries must be fast, $O(N)$ time search algorithms are not acceptable and even $O(\log N)$ is sometimes too slow. As a result, a substantial amount of research has been devoted to developing new spatial data structures and search algorithms for efficent point representation and querying. We are going to explore a simplified version of one such approach, called *geometric hashing*.

Geometric Hashing

To keep things simple, we will only consider two-dimensional space. Using a regular grid, the space is divided up into a series of squares of width w. Each square becomes a "bin", and the bins are indexed as though they are a two-dimensional array. When data points are read, they are assigned to bins. An example, with $w = 8$ is show here and explained further below.



The square containing four points and bounded on the left by 8, on the right by 16, on the bottom by 0 and on the top by 8 is bin $(1, 0)$. The other non-empty bins are indexed as $(-2, 1)$ (containing three points), $(-2, -1)$ (containing two points), $(-1, -2)$ (containing three points), $(2, -2)$ (containing five points), and $(2, 2)$ (containing one point).

The trick of geometric hashing is that these bins are not stored as a two dimensional array. Instead they are stored in a hash table. Only bins that have at least one point are stored — a bin is empty if and only if it is not stored in the hash table. Thus, in the example, six bins are non-empty and there are 18 points stored.

In order to make this concrete, we need to specify a way to map (x, y) point locations to bin indices, **and** we need a way to hash bin indices, mapping them to locations in a hash table. This depends on the size of each bin. Given point location (x, y), the associated 2d bin index (i, j) is

$$i = \lfloor x/w \rfloor \qquad \text{and} \qquad j = \lfloor y/w \rfloor .$$

To implement these in C++ you will need the function `floor`, which is declared in the header file `cmath`. Unfortunately, this function returns a double (even though it is an integer value), so you must convert it to an int, as in

```
  i = int( ceil(x/w) ).
```

To hash the (i, j) values (and therefore the associated bin), we will adapt the hash function used in Lecture 20 and Lab 11, so that the hash value is

$$|i * 378551 + j * 63689|.$$

Use `std::abs` to implement the absolute value function.

Assignment

Your assignment is to implement a class that performs geometric hashing. A starter header file, `geo_hash.h`, and a test main program, `test_geo_hash.cpp`, are provided on the course web site. Your specific job is to implement the functions in the class `geo_hash`. You may add variables and functions to `geo_hash` and you may add other classes as well, but you will be tested on the correctness of the twelve public member functions already declared in in `geo_hash`. You may not change these function declarations. Place the member function implementations in a file called `geo_hash.cpp`.

The twelve functions are as follows:

- A constructor that only takes a `bin_width` parameter (the same as w described above), which defaults to 10.
- `add_point`: add a single point to the geometric hash.
- `add_points`: add a vector of points to the geometric hash.
- `points_in_circle`: find and return a vector that contains all points in the geometric hash that fall within a given circle (distance to the center is less than or equal to the radius). The points in this vector should be ordered by increasing x value of the points and, for ties, by increasing y.
- `points_in_rectangle`: find and return a vector that contains all points in the geometric hash that fall within a given rectangle. The points should be ordered in the same way as points found by `points_in_circle`. Together this function and the previous function are the main query mechanisms of the geometric hash.

- `erase_points`: erase from the geometric hash all points that fall within a specified circle. You will notice that the radius defaults to a very small positive number, so that a single point location (if it is in the hash) may be removed — the small number is provided to accommodate minor differences in point values due to round-off effects. The function returns the number of points erased. **Important note:** when the erase function removes all remaining points from a particular bin, that bin should be removed from the hash table itself.
- `point_to_bin`: convert a point to a bin location
- `hash_value`: convert a point to a bin location and then compute its hash value. This hash value is an unsigned int and does not depend on the size of the table
- `points_in_bin`: return a vector containing all points in a bin associated with a given point location. These points should be ordered as above.
- `num_non_empty`: return the number of bins that currently have points stored in them (and are therefore explicitly represented in the table)
- `num_points`: return the number of points stored in the table. Note that this must be at least as large as the number of non-empty bins.
- `table_size`: number of locations in the hash table vector.

You will notice that several of these functions — such `erase`, `points_in_circle`, and `points_in_rectangle` — may require finding the points in a larger area than is covered by one bin. In this case, your functions will need to find and test all necessary bins. This is part of the challenge of this assignment.

Use of Hashing

As the main underlying data structure of the `geo_hash` class you must implement hashing. You should feel free to adapt code from the implementations we studied in Lecture 20 and Lab 11. (You can start with either version!) In doing so, make your hash tables as specific as you can to the particular problem you are solving. This means you should not use templating and you should not use hash table iterators.

It will be necessary to resize the hash table as needed. Start the table at size 4. Resize the table, when the number of non-empty bins (not necessarily the number of points) is more than 1.5 times the table size. When resizing, if n is the current size, the new size should be $2n + 1$. (For example the first resize should occur when adding the 7th bin.) Do not ever shrink the size of the table, even if the number of bins goes to 0. Note that resizing does not depend on the number of points — all points could be in a single bin.

Submission Instructions

Please adhere to the following submission instructions. Place all of your code in **two files** called `geo_hash.h` and `geo_hash.cpp`, and place them in a folder named hw9 (no spaces, no uppercase, not `hw9 submit`, not `H9`). Do not include any extra files. (In particular, do not include the `test_geo_hash.cpp` file — the submission script will use our `test_geo_hash.cpp` file to compile your code.) Zip the folder (not just the files in the folder), so that when it unzips a new folder is created. Submit it through the course web page.