CS 3250: Computational Geometry

Spring 2018: T, Th 11:30 - 12:55 in Searles 126

Piazza link | Github link | Schedule | Code | Assignments

Computational geometry studies algorithms for problems that involve geometric data, such as finite sets of points, line segments and polygons. Some classical examples of geometric problems: Is a point inside a polygon? Do two polygons intersect? Given a set of points in 2D , find the closest pair of points; or, find the smallest convex polygons that contains them; or, find all points that fall in a given query range. What is visible from a point in a polygon? How many points are sufficient to guard any polygon? Given two sets of segments, find their intersections. Given a set of points, find the emallest enclosing circle. Given a set of polygonal obstacles, and a polygonal robot moving in 2D with tramnslation and rotation, plan its motion between a start and end location. And so on. Computational geometry is a very young and growing field, driven by applications in graphics, robotics, autonomous vehicles, vision, image processing,and GIS.

Tentative syllabus:

This class will cover basic topics in computational geometry:

- Introduction
  - finding collinear points
  - finding the closest pair of points
- Geometric primitives
  - area of triangle, orientation, segment intersection).

- Convex hulls in 2D
    - naive, gift wrapping, quickhull, Graham scan, incremental, divide-and-conquer algorithm
    - lower bound
- Segment intersection
    - Bentley-Ottman sweep
- Art gallery problem. Fisk sufficiency proof.
- Polygon triangulation
    - quadratic, based on ear removal algorithm.
    - triangulation of monotone polygons.
    - polygon triangulation in O(n lg n) via trapezoidalization.
- Orthogonal range searching
    - kd-trees and range trees.
- Voronoi diagrams.
- Delaunay triangulations.
- Motion planning in 2D
    - shortest path in a simple (non-convex) polygon with the funnel algorithm.
    - shortest paths among polygonal obstacles via visibility graph.
    - computation of the VG with plane sweep.
- Motion planning in 3D.

Prerequisites: Data Structures (cs2100) and Algorithms (cs2200). In other words, knowledge of:

- basic analysis: asymptotic notation, growth, solving recurrences.
- basic algorithm design techniques: divide-and-conquer, greedy.
- basic algorithms and data structures: searching, sorting, binary search trees, priority queues.

CS curriculum: This is a 3000-level class that fullfills the Algorithms/Theory requirement.

TAs:

- Duncan Gans
- Jake Adicoff

Office hours:

- Laura: Tuesdays 1:00-2:30; Thursdays 1:00-2:30
- Duncan: Mondays 7-9pm
- Jake: Wednesdays 7:30-9:30pm

Class webpage: *http://www.bowdoin.edu/~ltoma/teaching/cs3250-CompGeom/spring18/*. This is hosted on my personal website at Bowdoin and will contain all class-related materials along the semester. You can bookmark this page or you can follow the link from my website. The class does not have a Blackboard website (this way it's publicly available to everyone).

For all class-related discussion we'll be using Piazza (link at top).

Textbooks: The following textbooks are suggested, not required. You can find them in Searles 224.

- Computational geometry in C. J. O'Rourke.
- Computational geometry: algorithms and applications. Mark de Berg, Otfried Cheong, Mark van Kreveld, Mark Overmars.

Work and collaboration policy

The work for the class will come from programming assignments (approximately biweekly) and class engagement/participation.

Collaboration policy: You can work alone, or you can pair with a partner (pair-programming).

Collaboration across teams is at level-1: that is, verbal collaboration without solution sharing. You are allowed and encouraged to discuss ideas with other class members, but the communication should be verbal and additionally it can include diagrams on board. Noone is allowed to take notes during the discussion (being able to recreate the solution later frommemory is proof that you actually understood it). Communication cannot include sharing pseudocode for the problem. Please read the department's collaboration policy.

Pair programming policy: *If you chose to work with a partner, you must work together, physically in the same place, for the entire project that you do together. The overall project must be a true joint effort, equally owned, created and understood by both partners. Specifically splitting the problem into parts and working on them separately is not allowed and violates the honor code for the class. If you start together as a team and are unable to complete the project together, then you will each inherit teh shared code, and split up to work individually on the remainder of the project. You would then submit individually, with a note on what happened.*

There are many advantages to working with a partner (e.g. more fun, more learning, good skill for the "real world"), and I encourage you to try. You may also consider alternating between working alone and with a partner, and changing partners.

Remember that you are responsible for reading, understanding, and adhering to the policy. If you have any questions about any aspects of the policy, please do not hesitate to ask for clarification.