CS 7301.003.20F Lecture 1—August 17, 2020

Main topics are #introduction, #administrivia, #planar_curves, and #Jordan_curve_theorem.

Introduction

- Hi, I'm Kyle!
- Welcome to CS/SE 7301.003.20F — Recent Advances in Computing — Computational Topology.
- Computational topology is an emerging area of study in computer science, and at least for the parts we'll discuss in this class, it's largely spun off from the computational geometry community.
- Computational topology can mean a lot of things! Any kind of algorithm design involving topological problems, or any use of topology to solve problems either in algorithm design itself or in other areas of computing.
- In fact, there's so many topics relevant to computational topology that I couldn't hope to cover them all! So this course is going to focus on a few things that I either have a lot of personal experience in or think will be useful to you all even if you don't work in the area directly.
- In particular, we're going to spend quite a bit of time in one of my main research areas: graph embeddings and algorithms for embedded graphs. Embedded graphs include the planar graphs, ones you can draw in the plane without edges crossing.
- We'll also discuss some stuff relevant to motion planning and something called simplicial homology and persistent homology. I don't work on this stuff directly, but I think it will be useful for you to learn about. A lot of data analysis techniques involve taking data points without obvious connections, and then looking for topological features in spaces based on how close or similar the data points are.
- These plans are still very loose, because I want to tailor this class to be a good match between my expertise and you alls' interests.
- So if there's any topics related to computation and topology that you would like me to discuss, please let me know. These topics could include things like robot motion planning with topological constraints, hexahedral meshing, tracing normal curves, more things I'm not familiar with but maybe I can help you make sense of them… don't be afraid to ask!

Administrivia

- But for now, let's discuss how this class will be run.
- Everything I'm about to say can be found on the course website: https://

personal.utdallas.edu/~kyle.fox/courses/cs7301.003.20f/

- This course has the "remote" modality, meaning we won't be meeting in person excepting special circumstances.
- All lectures and office hours will be held over MS Teams. If you haven't figured that out yet, I'm not sure how you're listening to me.
- I also encourage you to use MS Teams as a discussion board. There's probably tools in eLearning for this sort of thing too, but that system is such a pain to use that I'd rather avoid it as much as we can.
- I will use eLearning to email announcements, accept homework submissions, and post grades, though.
- The course has no official prerequisites. However, I highly recommend you take or have taken CS 6363—Design and Analysis of Computer Algorithms or an equivalent class at another university. I'll frequently refer to various graph algorithms or algorithm design techniques from 6363 as we go, especially when we're discussing algorithms for embedded graphs. I'll try to fill in necessary background material, but I can't promise I'll be able to completely fill you in during lecture. Please please please stop me to ask questions if I'm assuming too much background knowledge.
- One thing I won't assume is you having taken a topology class. Here, I'll be sure to fill in any definitions or basic results we might need as we go.

- Unlike some other 7301 sections, I will be teaching this course using regular lectures, likely on a whiteboard. Your responsibilities with the material are to watch the lectures, read any lecture notes I write myself, read any relevant portions of other lecture notes or books I link to, and at least skim any papers I link to.
- If you can, please attend the live lectures on MS Teams so you can ask me questions as we go. I'm wearing earphones and frequently checking my computer screen so I can stay in sync with you all.
- If you cannot attend live lectures, I completely understand, and I don't expect you to give me any notice or explanations. An MS Streams recording of each lecture will appear in the Lectures channel on Teams as soon as I can make it available.

- Your grade will be based on a weighted average of three things.
  - 40% will come from homework assignments.
    - I'll release homework once every few weeks and make it due about two weeks later. This semester is short and stressful so we'll probably do three, maybe four assignments.
    - You may work in formal groups of up to two people if you'd like, but it is not a requirement to group up. Each group should turn in **one** copy of the homework on eLearning. I'll give everybody in the group the same grade.

        - If you need extra time past the deadline, you must ask for an extension. I will automatically approve all extensions of up to 48 hours, but you still need to ask. I might give you longer if there are extenuating circumstances. It's 2020. Please don't be afraid to ask.
    - The rest of the grade will come from a course project. This can be anything suitably project-like such as a survey, implementation for some application or performance testing, or even original research in computational topology. The course website has more details on what kinds of things you might do projects on.
    - Each *individual* student will turn in a one to two page project proposal saying what you'd like to do. These proposals are worth 10% of your grade. I'll put your proposals up on eLearning or maybe MS Teams so they'll be visible to the rest of the class but not the greater internet.
    - Finally, you'll get to form groups of up to two people again for the final project, which means you don't have to go through with your specific proposal if you like somebody else's better. I may approve a group of three if you give me a good reason. 50% of your grade will be split between short presentation at the end of the semester and a short paper of around 10 pages.
        - For research proposals, I expect almost all reports to be short surveys and descriptions of a few failed attempts. But if you partially or fully solve your problem, then all the better!
- I'll add up all of these things and **then** heavily curve the grades.
- Let me reemphasize, **there are no set percentage targets you need to meet to get certain grades**.
- This curving scheme means I can ask tough homework questions.
- I think fighting back against the homework is your best opportunity to learn.
- So don't expect high percentages.
- Also, algorithms is a theory topic, so you need to justify your answers or your algorithms with an argument, some might say a proof, that they are correct. Try to be at least as rigorous as I am in lecture.
- As a rule of thumb, if you can't explain why your answer is correct after appealing to things in class and your own logic, it probably isn't a correct algorithm.
- The website contains a whole page devoted to what you should and shouldn't do when answering homework questions.
- On the other hand, this is a 7xxx special topics course. I'll be pretty lenient with grades in the end. Please email me if you're concerned about your grade.

- I am tentatively planning to do office hours Wednesdays from 2:30pm to 3:30pm and Thursdays from 10:30am to 11:30am in the Office Hours channel on MS Teams. I can set additional private meetings by appointment if you'd like. If those regular times don't work

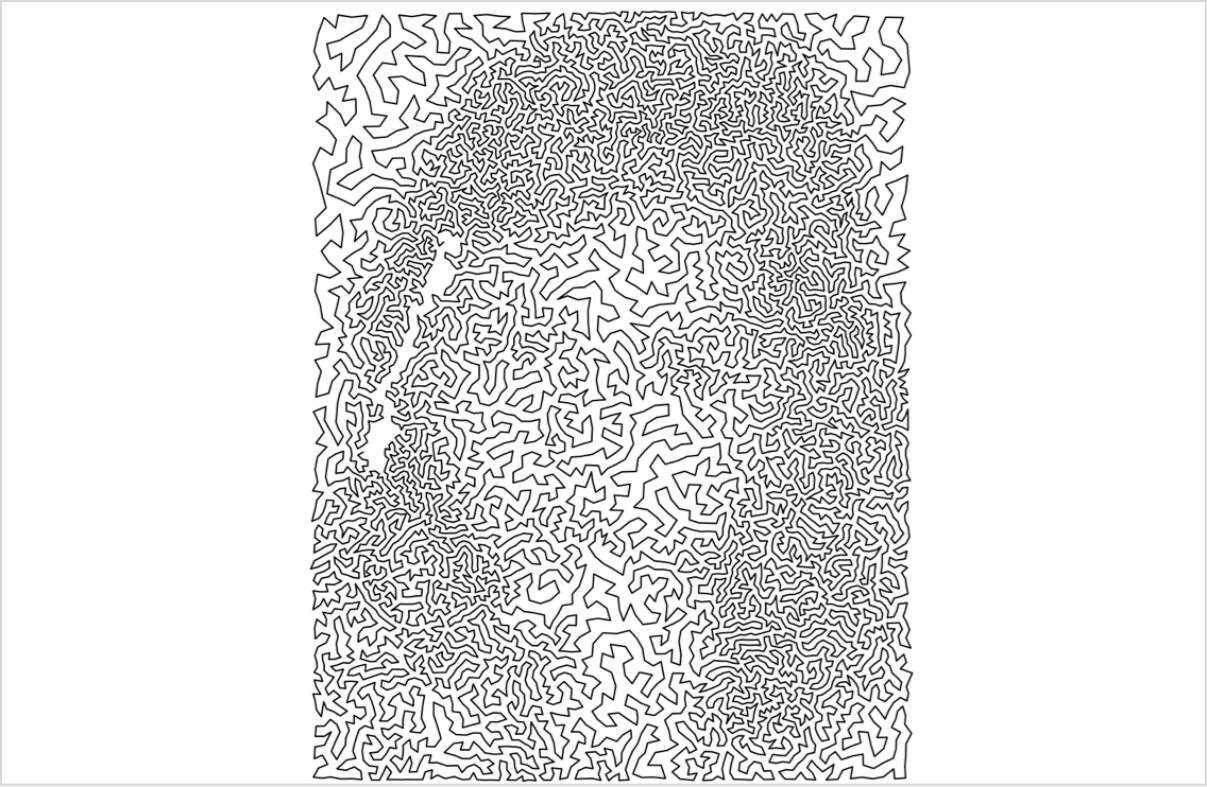
for too many of you, they can be changed.

- There's no required textbook, but I'll frequently link to material I may find useful on the schedule on the course webpage. I'll likely link to a textbook draft by Jeff Erickson for example since he is how I learned much of what I know and research in computational topology. I'll start pulling from other sources more often as the semester progresses.
- I'll also post my own lecture notes. They're essentially a script I'm writing for myself. They should be fairly thorough, but I probably wrote them the day before class in Markdown. They aren't pretty, and they may have bugs.

- Alright, final bit of administrivia. I've either already posted an "assignment" to eLearning or will shortly to fill out these prerequisite forms. There aren't any official prereqs, but you should still fill out the form. Please fill out the form and submit it to eLearning.
- And with that, let's spend what time we have left actually talking about some computational topology!

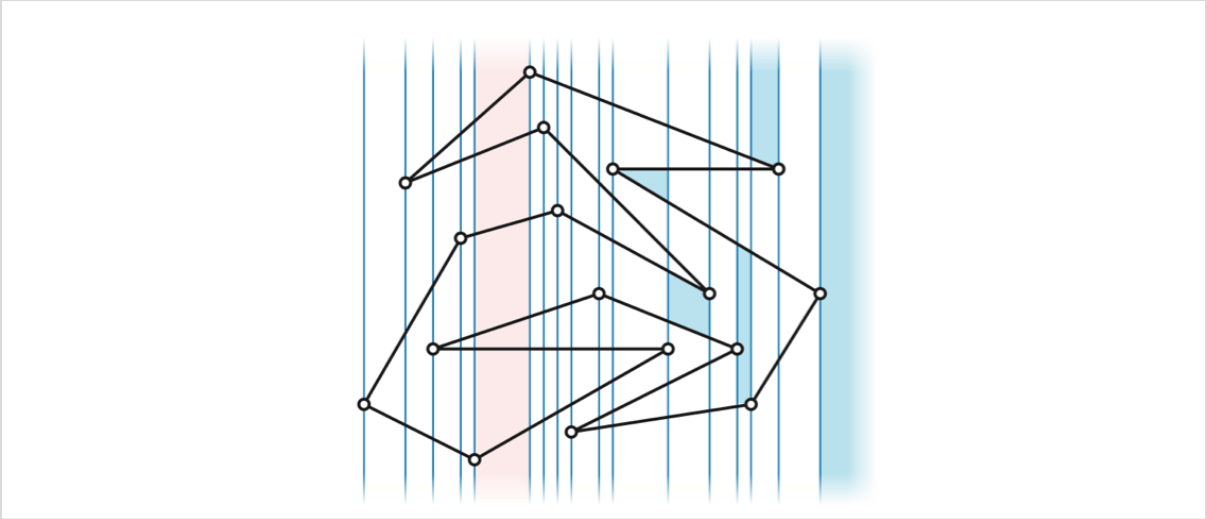
Planar Curve Definitions

- One of the simplest problem domains we can discuss is the problems over curves in the plane. If we can't understand these and the definitions involved, we're going to have a difficult time when we move on to problems in embedded graphs.
- So today we're going to go over some basic definitions and fundamental results for curves in the planes. There may not be too much computation today.
- A *path* in the plane is a continuous function pi : [0, 1] ➜ R^2 where [0, 1] is the unit interval on the real line. Points pi(0) and pi(1) are the *endpoints* of the path, and we might say pi is a path from pi(0) to pi(1). As we increase the domain value, the image from pi follows the path through the plane.
- Path pi is *simple* if it is injective (one-to-one) (has no repeated points).
- A *closed curve* or *cycle* in the plane is a continuous function from the unit circle S^1 = {(x, y) \in R^2 : x^2 + y^2 = 1} to R^2. Like before, a cycle is *simple* if it is injective.
- A subset X of the plane is (path-)connected if there is a path from any point in X to any other point in X. A (path-)connected component of X is a maximal path-connected subset of X.
- If you look at a simple cycle in the plane, it "obviously" separates the plane into two components. But sometimes, it's not too obvious.

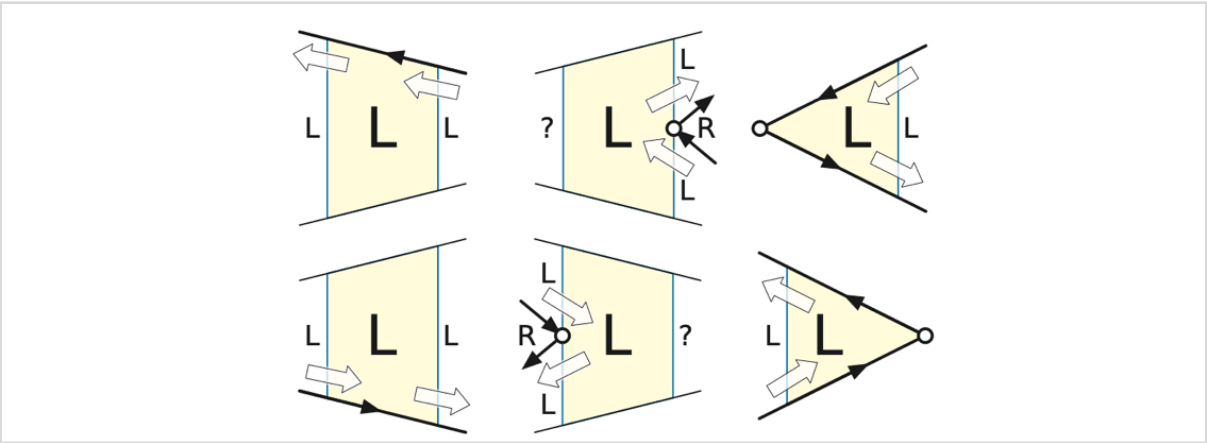


- The Jordan Curve Theorem: The complement R^2 \ C of any simple closed curve C in the plane has exactly two components.
- I'm not going to prove this theorem formally, but we'll see a proof (sketch) of a special case today or Wednesday.
- If you took computational geometry, the following definitions might sound familiar: A *polygonal chain* is a finite sequence of line segments p_0 p_1, p_1 p_2, …, p_n joining adjacent pairs in a finite sequence of points p_0, p_1, …, p_n in the plane. The points p_i are the *vertices* of the polygonal chain, and the segments p_{i - 1} p_i are its *edges*.
- Polygonal chains describe piecewise-linear paths in the plane. Specifically, we can define pi_P : [0 , 1] ➜ R^2 where for any i, the restriction of pi_P to [i / n, (i + 1) / n] is a linear map of line segment p_i p_{i + 1}.
- Polygonal chain P is *closed* if it has at least one edge and p_0 = p_n; otherwise it is *open*. Closed polygonal chains are also called *polygons*; no, the definition does not require simplicity. It's an n-gon if it has n edges and n vertices.
- A polygonal chain is *simple* if its vertices are distinct (except maybe p_0 = p_n) and its edges intersect only at endpoints. We can finally prove a case of the Jordan Curve Theorem:
- The Jordan Polygon Theorem: The complement R^2 \ P of any simple polygonal chain P in the plane has exactly two components.
- I'll sketch a proof that follows arguments by Schönflies (1896). We'll assume no two points are vertically aligned.
- Let ell_i be the vertical line through p_i. These lines divide the plane into n + 1 vertical *slabs* (two of which are halfplanes). The edges of P further divide the slabs into *trapezoids*,

each of which has at most four line segments on its boundary; the *floor*, the *ceiling*, and the left and right *walls*. We'll formally define the trapezoids as including its walls, but not its floor, ceiling, or any vertices.



- I first claim R^2 \ P has at most two components.
  - Imagine each edge p_i p_{i + 1} as directed from p_i to p_{i + 1}. We'll label a trapezoid as *left* if at least one of the following is true:
    - The floor is directed left to right.
    - The ceiling is directed right to left.
    - The right wall contains a vertex p_i, and p_{i - 1} p_i is below p_i p_{i + 1}.
    - The left wall contains a vertex p_i, and p_{i - 1} p_i is above p_i p_{i + 1}.



  - *Right* trapezoids have a symmetric definition. Every trapezoid has at least one of the four things mentioned in the bullets, so every trapezoid must be left, right, or (as far as we can tell so far) both.
  - Now, imagine creating a sequence of left trapezoids as we walk along the polygon from p_0 to p_n = p_0. As we traverse an edge p_i p_{i + 1} directed right (resp. left), we add trapezoids just above (resp. below) the edge in order from left to right (resp. right to left). When we traverse p_i whose neighbors are both right of p_i with p_{i - 1} p_i above p_i p_{i + 1}, we add the trapezoid just left of p_i to the sequence. When we traverse p_i whose neighbors are both left of p_i with p_{i - 1} p_i is below p_i p_{i +

1}, we add the trapezoid just to its right to the sequence.
    - Every left trapezoid appears at least once in the sequence, and adjacent members of the sequence share a wall, so the union of the left trapezoids is connected.
    - Similarly, the union of the right trapezoids are connected.
- Now to show R^2 \ P has at least two components.
    - Label each trapezoid *even* or *odd* to match the parity of the number of polygon edges above the trapezoid. So each slab has an even trapezoid on top and then they alternate odd and even as you go down.
    - Now, suppose trapezoids A and B share a wall on ell_i with A on left and B on right. If p_{i - 1} and p_{i + 1} are on opposite sides of ell_i, the same number of polygon edges lie above A and B. If both p_{i -1} and p_{i + 1} are left of ell_i and they lie below A and B's wall, then the same number of edges lie above both A and B. If both vertices are to the left but above the common wall, then A has two more edges above it than B. Similar cases exist when the vertices are to the right of ell_i. In all cases, A and B have the same parity.
    - And then by induction, all trapezoids in the same component of R^2 \ P have the same parity.
    - There's at least one edge in P, so there's at least one odd trapezoid which lies just below the highest vertex, so there are at least two components, one containing even trapezoids and another containing odd.
- Notice how trapezoids containing infinitely high points "outside" P are even. We can test if a given point lies inside, outside, or on P by asking if it lies in an odd trapezoid, in an even trapezoid, or on an edge. To do so, we just loop through in O(n) time, checking which edges lie above the point.

```
POINTINPOLYGON(P[0..n-1], q):
  sign ← -1
  P[n] ← P[0]
  for i ← 0 to n - 1
      sign ← sign · ONORBELOW(q, P[i], P[i+1])
  return sign
```

```
ONORBELOW(q, r, s):
  if r.x < s.x
      swap r ↔ s
  if (q.x < s.x) or (q.x ≥ r.x)
      return +1
  return sgn(Δ(q, r, s))
```

- The algorithm PointInPolygon returns +1, -1, or 0 to indicate point q is inside, outside, or on P.
- OnOrBelow(q, r, s) returns -1 if q lies below segment rs, 0 if q lies on rs, or +1 otherwise.
- It uses a routine sgn which tells you if the triple (q, r, s) of points is oriented counterclockwise. You may recognize such a test if you took computational geometry.