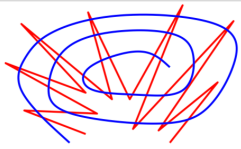
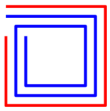
CS 6301.008.18S Lecture—March 20, 2017

Main topics are #curve_similarity and #Fréchet_distance.

Curve Matching and Hausdorff Distance

- From this point in the semester on, I'm going to focus less on what it's in the textbook and more on newer results and areas of computational geometry.
- Today, we'll look at problems inspired by shape matching. We'll eventually get to something similar to motion planning, so hopefully it will be a nice transition point.
- Say we're given two curves P and Q in R^d. Yes, what I'm going to tell you actually works in any constant dimension today.
- Our goal is to figure out how similar P and Q are by defining some kind of distance between them. One application is in handwriting recognition. Maybe I "know" what an A looks like, and I "know" what a B looks like. Is this handwritten curve an A or a B? The lower distance to the reference curve should give the answer. Now, how should we define the distance?
- There's one popular distance measure between point sets, including curves, called the Hausdorff distance.
- Let h(P, Q) = max p in P min q in Q ||pq||, and let h(Q, P) = max q in Q min p in P ||pq||. So we want the point p farthest from Q and the point q farthest from P.
- The Hausdorff distance is H(P, Q) = max{h(P, Q), h(Q, P)}.
- So this seems like a pretty good distance measure at first. If you have a point p very far from everything on Q, then yeah those curves probably aren't similar.
- But it doesn't take into account the order of points along the curves. The curves below have small Hausdorff distance even though they look very different.





Fréchet Distance (1906)

- So today, we'll look at something a bit more complicated, but also much better for computing curve similarity.
- First, we need to add some formalism.
- The *curve* P is formally a function [0, 1] ➔ R^d. As you walk along the domain from 0 to 1, you walk along (say) the white board following the *image* of P. You only ever walk *forward* along the curve.
- One popular way to define Fréchet distance is to imagine walking along the image of the

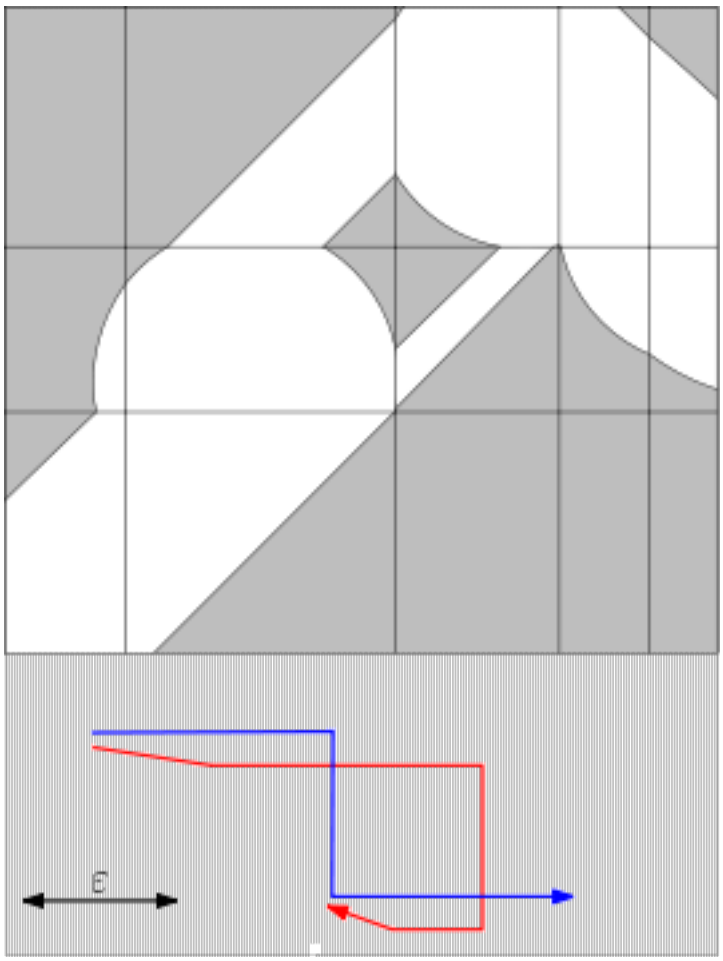
two curves P and Q simultaneously. Except, we have specify there is a *person* walking along P's image and a dog walking along Q's image.

- The person has the dog on a leash. If the dog gets too far ahead or behind, the leash will get taught, so they'd better following similar routes at roughly the same pace. What is the shortest leash length possible to complete the walks? It's kind of like a motion planning the problem: plan the motion of the person and dog simultaneously to keep that leash short.
- Formally, the walks taken by the person and dog are reparameterizations of the curves. A reparameterization is a continuous and bijection function alpha : [0, 1] ➜ [0, 1] where alpha(0) = 0 and alpha(1) = 1.
- The person speeding up or slowing down during the walk along P is represented as a reparameterization.
- So we say the Fréchet distance Fr(P, Q) = inf_{reparameterizations alpha, beta} max_{t in [0, 1]} ||P(alpha(t)) Q(beta(t))||.
- Our goal is to compute Fr(P, Q).
- To do so, we'll focus on the decision version of the problem: Given P, Q, and r, is Fr(P, Q) ≤ r?
- We'll reduce computing the actual Fréchet distance to solving the decision problem at the end of the lecture.

Discrete Fréchet Distance and Free Space Diagrams

- Let's start by discussing an easier problem called *discrete Fréchet distance*. We'll replace the person and dog by two frogs connected with a leash. At each time step, a single frog hops forward one point or both frogs hop at the same time. Still only forward. We only care about leash length when the frogs are sitting on points of P and Q.
- We can represent the possible positions of the frogs as vertices in a directed graph. The graph has m X n vertices, arranged in a grid. Vertices in the ith column are for points on P. Vertices in the jth row are for points on Q. A single frog jumping is represented by a horizontal or vertical edge between neighboring vertices. Both frogs jumping is representing as a diagonal edge.
- A vertex (p_i, q_j) is *free* if ||p_i q_j|| ≤ r.
- So now we ask, can we go from the bottom left vertex for (p_1, q_1) to the upper left for (p_m, q_n) using only free vertices? The graph has O(mn) vertices and edges. We can solve the problem in O(mn) time by building the graph and doing a search from (p_1, q_1) in O(mn) time.
- Now let's go back to original problem on curves. Things are a bit trickier now, because the person and dog could lie not only on pairs of vertices but also anywhere on a pair of edges.

- But since P and Q are curves, we can specify their position by a pair of values from [0, 1] X [0,1]. So instead of a discrete grid, we'll use the unit square. Position (alpha, beta) represents the pair (P(alpha), Q(alpha)).
- If the leash has length r, then the person and dog can stand at points P(alpha), Q(alpha) where ||P(alpha) Q(alpha)|| ≤ r. The set of such (alpha, beta) is called the *free-space diagram*. Just like the free space in motion planning except we add the word diagram because that's what they did in the paper.
- Below, P is the lower red curve. eps should say r. The hole in the middle represents person and dog sitting at opposite corners of that square.



- Say a path in the diagram is *monotone* if it is both x-monotone and y-monotone.
- Lemma: Fr(P, Q) ≤ r if and only if there is an x and y-monotone path from (0,0) to (1,1) in the free-space diagram for P, Q, and r.
- So how do we decide if a monotone path exists?

Reachable Free Space

- We'll say (alpha, beta) is *reachable* if there is a monotone path from (0,0) to (alpha, beta). Our goal is to decide what pairs are reachable and in particular if (1, 1) is reachable.
- Unfortunately, the free-space diagram can get fairly complicated, especially for arbitrary curves.

- However, we're assuming P and Q are polygonal chains.
- Each position (alpha, beta) in the unit square corresponds to a pair of points (P(alpha), Q(beta)) that lie on pair of edges. We can partition the unit square into rectangular cells based on these edges.
- In particular, let (alpha, beta) be in cell box_{i,j} if P(alpha) is on the ith edge of P (between p_i and p_{i+1}) and the jth edge of Q.
- A monotone path within a cell corresponds to person and dog walking forward along a single pair of edges.
- I won't prove it here, but it turns out the free space within any one cell is the intersection of an ellipse and the cell, and you can compute the intersection points with the boundary of the cell in constant time.
- In particular, the free space within the cell is *convex*.
- So each cell edge intersects the free space along at most one interval we call the *free space intervals*.
- A subset of the points in each free space interval are reachable. We call these the *reachability intervals*.
- And here's the main observation: if we know the reachability intervals on the bottom and left side of a cell, we can compute the reachability intervals on the right and top sides as well.
    - If the bottom is reachable at all, the entire right free space interval is as well. Also, every free point right of the left side of the bottom reachability interval.
    - If the left is reachable at all, the entire top free space interval is as well. Also, every free point above the bottom boundary of the left reachability interval.
- So, order the cells from left-to-right, bottom-to-top. You can reach the entirety of the free space intervals on the bottom left cell if and only if (0, 0) is free.
- Now for each cell in this order, use the reachability intervals you already computed for its left and bottom sides to learn its reachability intervals on the right and top sides.
- There are O(mn) cells, so the algorithm takes O(mn) time.

Computing Fr(P, Q)

- So we have an O(mn) time algorithm for the decision problem. How do we actually compute the Fréchet distance?
- Remember how earlier in the semester we'd look for line segment intersections by sweeping a vertical line? It's like checking all x-coordinates from -infinity to infinity.
- Imagine doing a similar thing here. We continuously increase r from negative infinity to infinity.
- Generally, increasing r just means the free space gets a little bigger and the reachability intervals get a little longer.

- But at certain critical values of r, noticeable changes happen to the free space intervals and reachability intervals.
- There are a few kinds of events we care about:
  - Vertex-edge: Let p be a point on P and u be a segment on Q. When r reaches the distance between p and u, you get a single point free space interval on the left side of the cell for the segment after p and u. There are O(mn) of these events.
  - Vertex-vertex: When r reaches the distance between vertices p on P and q on Q, a free space interval reaches the corner of their incident segments' pairs' cells. We really only care when this happens to p_1 and q_1 or p_m and q_n since vertex-edge events already cover the creation of these free space intervals. There are O(mn) of these.
  - Monotonicity: We also need to know when reachability intervals overlap horizontally or vertically so we can actually route monotone paths through them. Let p and p' be points on P in order along P and let u be a segment of Q. Just before the event, u gets within distance r of p' then far again then gets within distance r of p. At the event, there's a point Q(beta) on u that is within distance r of both p and p', meaning there is suddenly a horizontal line at beta in the free-space diagram between p and p''s vertical lines. There are O(m^2 n + m n^2) such events.
- Given the relevant objects for an event, you can figure out for which value of r the event will occur for those objects in constant time.
- We need to figure out which of these events creates the first monotone path between (0, 0) and (1, 1) in the free space diagram.
- So we can compute all such events and sort them by when they occur in O((m^2 n + m n^2) log (mn)) time. Then, binary search over the event's values of r, running the decision procedure for each guess of r.
- The whole thing takes O((m^2 n + m n^2) log (mn)) time total.
- There's a technique called *parametric search* which helps us avoid computing all the monotonicity the events. It results in a running time of O(mn log(mn)) time instead.