Yujie Xu
yujiex@andrew.cmu.edu

Template adapted from CMU's 10725 Fall 2012 Optimization course taught by Geoff Gordon and Ryan Tibshirani.

Note of Computer Graphics

October 8, 2015

Lecture 1: Introduction, The Line Intersection Problem using Sweepline

Instructor: Gary Miller Notes taken: Yujie Xu
Date: Monday, Aug 31, 2015

1.1 Intro

1.1.1 Course description from the syllabus

"How do you sort points in space? What does it even mean? This course takes the ideas of a traditional algorithms course, sorting, searching, selecting, graphs, and optimization, and extends them to problems on geometric inputs. We will cover many classical geometric construc- tions and novel algorithmic methods. Some of the topics to be covered are convex hulls, Delaunay triangulations, graph drawing, point location, geometric medians, polytopes, configuration spaces, computational topology, approximation algorithms, and others. This course is a natural extension to 15-451, for those who want to learn about algorithmic problems in higher dimensions."

Textbook: Computational Geometry: Algorithms and Applications [1].

Traditional algorithm courses mainly discuss 1D problems, such as BST. The topics of this course contains the following main topics:

- Large dimensional problems
- The change of the nature of simple geometry problems when dimension increases

The applications of computational geometry include:

- 2D: graphics
- high dimension: machine learning
- GIS
- CAD, CAM
- simulation

Algorithm design approaches:

- Divide and Conquer: "Divide the problem on size n into k ¿ 1 independent subproblems on sizes $n_1, n_2, \ldots n_k$, solve the problem recursively on each, and combine the solutions to get the solution to the original problem." (15-210 lecture note)

- 2D sweep-line
- Random Incremental
-

The basic issues or standard computational problems that will be discussed in recent lectures include:

- Line segment intersection (sweepline algorithm, random incremental algorithm)
- Convex Hull: given a set of points, compute the convex hull of these points.
- 2D-LP

The standard geometry problems that will be discussed recently include:

- Line side test
- In circle test

First the abstract objects and their representation that will be used in this course are discussed and is listed in the Table 1.1

Table 1.1: Abstract Object and Their Representation

Absract Object	Representation	Issue
Real Number	Float	Rounding Err
	Bignum (with arbitrary precision, normally they use arbitrary length array of digits)	Memory Intense
	Computer Algebra (Symbolic Computation)	
Point	Pair of Real	
Line	Pair of Points	
Line Segment	Pair of Points	
Triangle	Tripple of Points	

1.2 How to use points to generate object

Suppose $P_1, P_2, \ldots P_k \in \omega^d$ where $P_k \in M$ is a d-dimensional vector space with each point being a M-dimensional point, the following combinations of points creates the linear subspace of the vector space and generates geometric objects:

- Linear Combination:

$$Subspace = \sum_i \alpha_i \cdot P_i, \alpha_i \in \mathbb{R}$$

For the $d = 2$ and $P_i \in \mathbb{R}^3$ case, the linear combination of P_1 and P_2 forms a plane with $\vec{OP_1}, \vec{OP_2}$ being the basis.

- Affine Combination:

$$Plane = \sum_i \alpha_i \cdot P_i, s.t.\ \alpha_i \in \mathbb{R} \wedge \sum_i \alpha_i = 1$$

For the $d = 2$ and $P_i \in \mathbb{R}^3$ case, the affine combination of P_1 and P_2 forms a line that passes P_1, P_2.

- Convex Combination:

$$Body = \sum_i \alpha_i \cdot P_i, s.t.\ \alpha_i \in \mathbb{R} \wedge \sum_i \alpha_i = 1 \wedge \alpha_i \geq 0$$

$S \subseteq \mathbb{R}^d$ is a convex set iff $\forall p, q \in S\ .\ , [p, q] \subseteq S$ (A set S in a vector space over R is called a convex set if the line segment joining any pair of points of S lies entirely in S [2])

For the $d = 2$ and $P_i \in \mathbb{R}^3$ case, the convex combination of P_1 and P_2 forms a line segment between P_1, P_2.

Convex Closure/Hull: The minimal convex set $S' \supseteq S$

A subset S of the plane is called convex if and only if for any pair of points $p, q \in S$ the line segment $\overline{pq}$ is completely contained in S. The convex hull CH(S) of a set S is the smallest convex set that contains S [1].

Theorem 1.1. $CC(P_1 \ldots P_k) = \{\alpha \in \mathbb{R} \mid \sum_i \alpha_i = 1 \wedge \alpha_i \geq 0\}$

In convex geometry Carathodory's theorem states that if a point x of $\mathbb{R}^d$ lies in the convex hull of a set P, there is a subset P' of P consisting of $d + 1$ or fewer points such that x lies in the convex hull of P'.

1.3 Primitives

Problems related to geometry primitives

1) Test equality $p = q$?

2) Line segment intersection in 2D

   Let $L_1 = [P_1, P_2]$, $L_2 = [P_3, P_4]$, let $P_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix}$.

$$L_1 \cap L_2 \neq 0 \iff (P_1 P_2) \begin{pmatrix} \alpha_1 \\ \alpha_2 \end{pmatrix} = (P_3 P_4) \begin{pmatrix} \alpha_3 \\ \alpha_4 \end{pmatrix} \text{ and } \alpha_1 + \alpha_2 = \alpha_3 + \alpha_4 = 1 \text{ and } \alpha_i \geq 0. \tag{1.1}$$

If written in matrix form, Equation 1.1 becomes:

$$\begin{pmatrix} x_1 & x_2 & -x_3 & -x_4 \\ y_1 & y_2 & -y_3 & -y_4 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} \tag{1.2}$$

So the general process of solving the line segment intersection problem is

Step I. Solveeq:lineSeg2 with some solver

Step II. Check if $\forall i, \alpha_i \geq 0$

There could be more than one solutions when L_1 and L_2 intersect in more than one point: the four point are collinear and there is an overlay. But in terms of the problem, the solution is still unique, since the solution is either True of False.

Remark: The good practice is make this line segment intersection test be a sum-routine and call an existing solver to solve the matrix. Don't try to inline the code

Some cases when the algorithm output False:

- L_1 and L_2 parallel but not collinear
- the intersection is on the extension of the two line segment
- the intersection is on one line segment and on the extension of the other.

3) Line side test

- input: three points in 2D: P_1, P_2, P_3
- output: if P_3 is to the left of ray P_1P_2

  One process of solving the problem: Subtract P_1 from both of the other vectors. Let $V_2 = P_2 - P_1$ and $V_3 = P_3 - P_1$. Now the cross product $V_2 \times V_3$ is the signed area of the parallogram formed by V_2 and V_3. This area is > 0 if and only if P_3 is to the left of ray $P_1 \mapsto P_2$.

  Alternatively, suppose $P_1 = O$ (the origin), then the signed area of the parallogram formed by $P_1\vec{P}_2, P_1,\vec{P}_3$ is

$$\det \begin{pmatrix} x_2 & x_3 \\ y_2 & y_3 \end{pmatrix} \tag{1.3}$$

$$LHS = \det \begin{bmatrix} x_1 & x_2 & x_2 \\ y_1 & y_2 & y_2 \\ 1 & 1 & 1 \end{bmatrix} = det \begin{bmatrix} x_1 & x_2 - x_1 & x_3 - x_1 \\ y_1 & y_2 - y_1 & y_3 - y_1 \\ 1 & 0 & 0 \end{bmatrix} \tag{1.4}$$

  Just report the sign of determinant of LHS of Equation 1.4

Remark: The good property of method 2 is that it can be generalized to higher dimension easily: for a 4D space, the test checks the determinant of a 4 by 4 matrix.

4) In circle test

- input: four points in 2D: P_1, P_2, P_3, P_4
- output: if P_4 is in the circle of (P_1, P_2, P_3)

1.4 Problems

1.4.1 Line Segment Intersection Problem

- Input: n line segments
- Output: All I intersections
- Naive Approach: $\binom{n}{2}$ line segment intersection tests. $O(n^2)$
- There is $O(n \log n + |I|)$ today

1.4.1.1 Sweep Line Algorithm

Define the following:

- Input: $S = \{S_1, \ldots S_n\}$ line segments.
- $P \equiv$ Line segment end points
- $I \equiv$ Line segment intersections
- $Event \equiv P \cup I$

Assumptions:

- Horizontal line segments are not considered
- The case where three lines intersect at the same point is not handled

L is a horizontal line disjoint from $P \cup I$ that sweeps from top to bottom

Linear ordering (transitive, irreflexive and total) $(A, \prec)$:

- set $A = \{s \in S \mid s \cap l \neq \emptyset\}$
- relation: the position of the intersection from left to right between line L and the line segments: $p \prec q p_y > q_y \vee (p_y = q_y \wedge p_x < q_x)$

Remark: Order of set A changes at events

The order is stored in a Balanced BST B. The basic idea is to sweep the line L from top to bottom and stop at events.

Claim if the next event is $S \cap S'$ then S, S' are neighbors.

Priority queue Q_L is kept to hold events. By induction, it contains:

- P below L
- Neighboring line segments that intersect below L

Algorithm:

```
Insert P into Q
While Q not empty
P = ExtractMax(Q)
HandleEvent(P)

Handle Event(P):
    if P is upper end of S
        insert (S, B)
        add-intersection(left(S), S, Q)
        add-intersection(S, right(S), Q)
    if P is lower end of S then
        add-intersection(left(S), right(S), Q)
        delete(S, B)
    if P is an intersection of S' and S
        swap (S, S', B)
        add-intersection(left(S'), S', Q)
        add-intersection(S, right(S), Q)
        report P
```

Each step of the alg is $\log(n)$, there are n of upper end and lower end. There are I of "P is an intersection". The total cost is $O(n + |I|) \log n$

1.4.1.2 Map Overlay Problem

"given a set S of n closed segments in the plane, report all intersection points among the segments in S"

- Input: segments $S = \{S_1, \ldots, S_n\}$
- Output: Break all segments into sub segments such that two sub segments intersect only at endpoints
- Algorithm: SweepLine

- Events: All segment intersections (I), and All end points (P)

Use link lists to store the following

- $U(P)$ = Subsegments with the upper end point P
- $L(P)$ = Subsegments with the lower end point P
- $C(P)$ = intersection in P

First initialize U and L with S, initialize $C(P) = \emptyset$ The procedure goes as:
`HandleEvent(P point, T tree, Q queue)`

1) $\forall x \in C(P)$, form new sub segments, break S into S_1, S_2, add S_1, S_2 to $U(P)$ and $L(P)$
2) $\forall s \in L(P)$, delete (S, T)
3) $\forall s \in U(P)$, insert (S, T)
4) for all new neighbor pairs, add intersection to Q

The run time of this algorithm is $O(m \log n)$ with $m = \#subsegments$, because there's at most m inserts and deletes into T and Q.

Lecture 2: Convex Hull-1

Instructor: Gary Miller   Notes taken: Yujie Xu
Date: Wed, Aug 31, 2015

Missed the lecture as a result of going to another lecture. The following notes are tidied from the online lecture notes.

2.1 Definitions

Definition 2.1. $A \subseteq \mathbb{R}^d$ *is convex, if it is closed under convex combination.*

Definition 2.2. *Convex Closure(A)* $\equiv$ *smallest convex set* $\supseteq A$

Definition 2.3. *Convex Hull:*

- $CH(A) = JCC(A)$ *(Boundary), we'll use this definition*
- $CH(A) = CC(A)$

A is a finite set, thus in 2D, CH(A) is just a closed polygon with vertices in counter-clockwise order.

2.2 Lower Bound

Sorting $\leq_M$ CH: sorting problem reduces to convex hull problem, meaning CH is at least as hard as sorting problem.

- Input: $(x_1, x_1^2), (x_2, x_2^2), \dots, (x_1, x_1^2)$
- $CH(x_1, x_1^2), (x_2, x_2^2), \dots, (x_1, x_1^2)$ output a sequence of points with x_i in sorted order.

2.3 An important use of CH in triangulation

Let $P_1, P_2, \dots P_n \in \mathbb{R}^2$, $\bar{P}_i = (P_x, P_y, P_x^2 + P_y^2)$, then $CH(\bar{P}_1, \bar{P}_2, \dots, \bar{P}_n) \equiv$ Triangulated surface, the Delaunay Triangulation

"Triangulation is the division of a surface or plane polygon into a set of triangles, usually with the restriction that each triangle side is entirely shared by two adjacent triangles."`http://mathworld.wolfram.com/Triangulation.html`

To test whether a line segment is on convex hull or not, we'll use the following characterization:

Claim [a, b] is on CH(A) iff $a \neq b$, and

- $a, b \in A$
- $\forall a' \in A$, either a' left of [a, b] or $a' \in$ [a, b]

2.4 Quick Sort and Backward Analysis

The process of quick sort is as follows:

```
QS(M):
    pick random a \in M
    split M with L < a < R
    return QS(L) * a * QS(R)
```

Dart Game:

```
Initial state: an empty array of squares
while there exists an non-empty square
    pick a random non-occupied square to throw a dart on
```

The cost of each dart throw is # empty squares to the left of the dart and the # of empty squares to the right of the dart

Claim The expected cost of the dart game = the expected cost of quick sort

Backward Dart Game:

```
Initial state: an array of squares each with a dart in it
while there exists a dart
    pick a random dart and remove it
```

Claim The expected cost of the dart game = the expected cost of the backward dart game

Analyzing the backward game:

Let T_i be the expected cost of removing a random dart Each chunk of empty squares can be

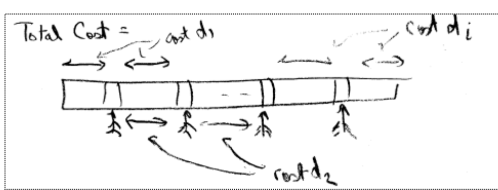

Figure 2.1: Backward Dart Game

consumed by the dart on its left or right, except for the left most and the right most chunk, so the total cost of removing each of the i dart is

$$\leq 2(n - i)$$

Averaging the total cost of the total i possible event yields the expected cost for removing a random dart when there are i dart in the board is:

$$T_i = \frac{2(n-i)}{i}$$

The total expected cost of $i \in [1, n]$ is

$$\sum_{i=1}^{n} \frac{2(n-i)}{i}$$

2.5 Algorithms for CH

2.5.1 2D CH with Divide and Conquer

2.5.1.1 Procedure

Let $A = \{P_1, P_2, \ldots, P_n\}$, with $P_i = (x_i, y_i)$ Preprocess: sort points in A with x-coordinate.
2D-CH(A) =
if $|A| = 1$, return P_1
else $CH_L = 2D - CH(P_1, P_2, \ldots, P_{n/2})$
$CH_R = 2D - CH(P_{n/2+1}, \ldots, P_n)$
Stitch(CH_L, CH_R)

```
a = rightmost(L)
b = leftmost(R)
LowerBridge(L, R)
    repeat the following:
    *) if lower_a is not left of (a, b) vector, set a <- lower_a
    **) if lower_b is not left of (a, b) vector, set b <- lower_b
UpperBridge(L, R)
    repeat the following:
    *) if upper_a is not right of (a, b) vector, set a <- upper_a
    **) if upper_b is not right of (a, b) vector, set b <- upper_b
```

2.5.1.2 Correctness (Termination)

Take `Lowerbridge` for example. Each step of *) or **) creates a triangle, the triangles are ordered with the intersection of their lowest point of intersection with the line L.

Each round of *) or **) moves one of a or b downwards and leaving the intersection of the new edge, `lower_a, b` or `a, lower_b` with L strictly decreasing. There is a lower bound for the edges $\{(x, y) \mid x \in L \land y \in R\}$.

Lower bridge is in $CC(A)$ by definition.

The proof for the upper bridge is symmetric to that of the LowerBridge.

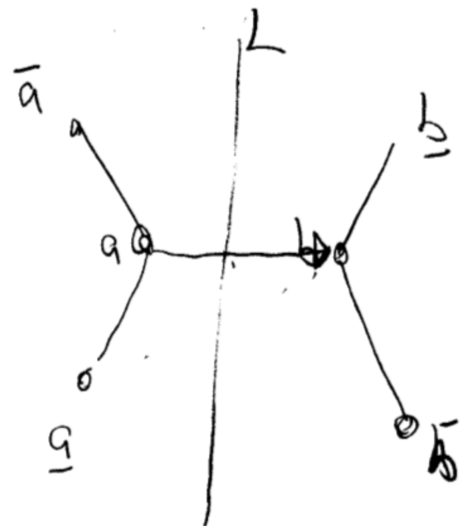

Figure 2.2: stitch graph

2.5.1.3 Cost

Preprocess by sorting the point: $O(n \log n)$
Stitch is $O(n)$
$T(n) = 2T(n/2) + c \cdot n$, so the total cost is $O(n \log n)$

2.5.1.4 Random Incremental CH

The procedure of the Random Incremental goes as follows:

```
Make a triangle T = (P1, P2, P3) from the set of points,
pick a point C in the interior of T
Construct a ray from C to each of the Pi
Partition Pi by the edge of T they cross
Randomly permute P4 to Pn
    For i = 4 to n
    let e be edge crossed by ray c -> Pi
    BuildTent(P, e)

BuildTent(P, e):
    Find edges "visible" to P by searching out from e
    Replace visible edges with 2 new edges that are "just visible", i.e. one of their ne
    Assign rays to the new edges
```

Runtime:

- Worst case: quadratic

- Best Case: linear

Imaging all points are in counterclockwize order.

The worst case of this setting is the first triangle is (P_1, P_2, P_3) and new points come in sub-script increasing order, for round 1, all the remaining $n - 3)$ points are partitioned to edge P_1, P_3, then $BuildTent(P4, [P_1, P_3]$, nothing is visible besides, P_1, P_3, do not add new edge, then go on, all edges are partitioned to belong to $[P_1, P_4]$. For a stage with i points in the existing convex hull, $(n - i)$ partitioning operations are needed, hence the total cost is $\sum_{i=3}^{n}(n - i)$, which is $O(n^2)$

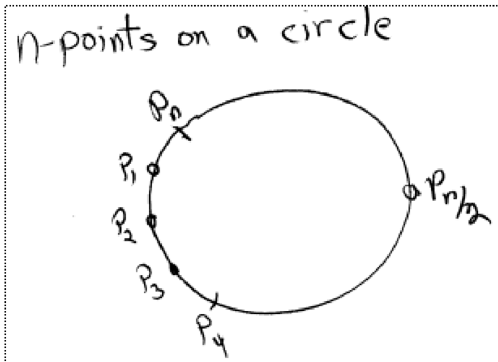

Figure 2.3: Worst Case

The best case is, the first triangle is $(P_1, P_{n/2}, P_n)$. Half of the points are assigned to $[P_3, P_{n/2}]$, the other half is assigned to $[P_1, P_{n/2}]$. The work at the first stage is $n - 3$. Then BuildTent happens on the $P_{n/2}, P_{n/4}, P_{3n/4}, \ldots$

Timing

Claim Work other than BuildTent cost $O(n)$

For BuildTent:

1) For each new point in $\{P_4, P_n\}$, there will be at most 2 edges added, thus adding visible edges takes time at most 2n

2) For searching for "visible" edges: use amortized analysis with the following "charging rule":

   (a) If an edge e is not visible, charge P_i, there are 2 for each i

   (b) If an edge e is visible, charge the edge. There are $4n$ of them, because the number of new edges added is $2n$, the number of removal of edges is also $2n$ (because an edge should be first added in order to be removed)

3) For step 3) of assigning rays to the two new edges in each stage.

   Here we use the backward analysis. Consider on stage i when there are i points "processed", we randomly pick one "peg" (point) and remove it, if the peg is inside the rubber band boundary of the CH on stage i, we cost nothing, if the rubber band changes, we spend the number of rays crossing the left and right edge incident to the peg that is just removed. i.e.

The cost of removing a random peg when there are i pegs processed is

$$P_i = \begin{cases} 0 & P_i \text{ in interior} \\ \#\text{rays crossing left or right} & \text{otherwise} \end{cases}$$

The expected cost of removing a random peg on stage when there are i pegs not removed is:

$$E_i \leq \frac{2(n-i)}{i-3}$$

The total cost is:

$$T = \sum_{i=4}^{n} E_i \leq \sum_{i=4}^{n} \frac{2(n-i)}{i-3} \leq 2n \sum_{i=1}^{n-3} \frac{1}{i} \leq 2n \sum_{i=1}^{n} \frac{1}{i} = 2nH_n \in O(n \log n)$$

Lecture 3: 2D LP

Instructor: Gary Miller *Notes taken: Yujie Xu*
Date: 2015

3.6 Introduction

The problem that given Half space (Hspace) $H_1, H_2, \ldots, H_n$, to compute the intersection is equivalent to sorting and it takes $O(n \log n)$:

$$\bigcap H_i \equiv Sorting$$

Although sorting takes $O(n \log n)$, the "selection" problem takes only $O(n)$. The same goes with the "selection" version of the Half space intersection problem, which is to find the farthest point in some direction.

Definition 3.1. *LP: maximize $C^T x$ subject to $Ax \leq d$ where A is a $n \times m$ matrix in $\mathbb{R}^{n \times m}$, $x, C \in \mathbb{R}^{m \times 1}$, $d \in n \times 1$. Define $x < y$ if $\forall i \,.\, x_i < y_i$*

Definition 3.2. *The LP is feasible if $\exists x \,.\, Ax \leq d$*

Note: *the region $\{x \mid Ax \leq d\}$ is convex.*

For the 2D case,

$$\begin{pmatrix} a_1 & b_1 \\ \vdots & \\ a_n & b_n \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} d_1 \\ \vdots \\ d_n \end{pmatrix}$$

Geometric View of half plane
$a_i x + b_i y \leq d$ with $d \geq 0$ is a half plane normal to the vector (a_i, b_i)

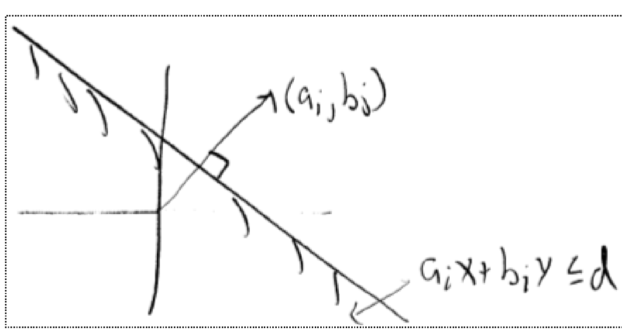

Figure 3.4: The half plane geometric view

Input: Half Space, $\{H_1, \ldots, H_n\}$ and a vector $C \in \mathbb{R}^2$.

Goal: to find the $x \in \bigcap_{i=1}^{n} H_i$ farthest in C direction.
Simplifications:

1) No H_i normal to C, (unique OPT solution)/
2) Bounded feasible solutions (not saying the $\bigcap_{i=1}^{n} H_i$ is bounded, but it is bounded in the C direction).
3) Given a "Bounding Box": m_1, m_2.

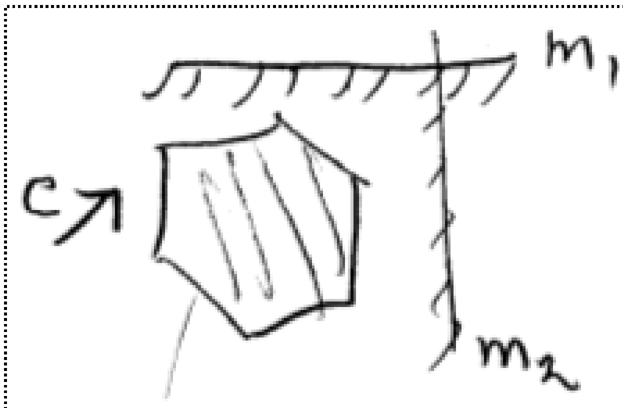

Figure 3.5: Bounding Box

3.7 1D LP

- Input: Constraints in the form $a_i x \leq b_i, a_i \neq 0$
- WLOG $a_i = 1$
- Constraints can be classified into two categories:
  - $C^+ = \{i \mid x \leq b_i\}$ (UB)
  - $C^- = \{i \mid -x \leq b_i\}$ so $x \geq -b_i$ (LB)

  Define $\alpha = \max\{-b_i \mid i \in C^-\}$ and $\beta = \{b_i \mid i \in C^+\}$

  Note: Feasible if $\alpha \leq \beta$
- If feasible, return β if $sign(C) = 1$, return α otherwise.

Theorem 3.3. *1D-LP is $O(n)$*

Because find max or min is $O(n)$

3.8 2D LP

The idea is to bring inrandom constraint one at a time and update the solution.

3.8.1 Procedure

```
v0 <- 2D-LP(m1, m2, c) = \partial(m1) \land \partial(m2) --- (*)
randomly order h1, ... , hn
for i = 1 to n do
    if v(i-1) \in hi then vi <- v(i - 1)                 --- (*)
    else (make and solve 1D-LP)
        L = \partial(hi)
        L1' = L \cap h1, ..., L_{i - 1}' = L \cap h_{i - 1}
        C' = proj(C, L) , note C' \neq 0
        vi = 1D-LP(h1', ..., h{i - 1}', C')              --- (*)
    if vi is undefined, report "no-solution" and halt.
```

3.8.2 Correctness

Claim At the time marked with (*), $v_i = LP(m_1, m_2, h_1, \ldots, h_i, C)$

Proof. by induction

- Base Case: OK by definition of bounding box.
- Inductive Case: assume v_{i-1} is correct.

Case i $v_{i-1} \in h_i$, then $v_{i-1} \in$ feasible region, so v_{i-1} is OPT for $h1, \ldots, h_i$

Case ii $v_i \notin h_i$

Claim $v_i \in \partial h_i = L$

Proof. Assume for the sake of contradiction, v_i is inside L, say the line segment $[v_i, v_{i-1}$ intersect L at A, then according to the previous assumption, there is always unique solutions to the LP, so from v_{i-1} to A to v_i, the objective value strictly decreases. thus the objective value at A is greater than that at v_i and A is feasible. Hence the assumption that v_i is inside L is false. □

Still need to show why solving 1D LP will give the right solution.

□

3.8.3 Timing

Claim 2D LP is $O(n)$ expected time We use backwards analysis by randomly throw away some constraints.

For a stage with i constraints: $h_1, \ldots, h_i$, we remove some random constraints h_j.

Definition 3.4. *h_j is critical if removing it changes the OPT solution. There are at most 2 critical constraints at each point v_i (the OPT for the stage where there are i constraints)*

Cost of removing $h_j = k$ if h_j is not critical, $k \cdot i$ otherwise. k is some constants. (why??)

The worst case is when there are exactly 2 critical constraints.

$$E_i = \frac{2 \cdot ki + (i-2)k}{i} \leq \frac{3ki}{i} = 3k$$

The total expected cost is $\sum_{i=1}^{n} 3k = O(n)$

3.9 Finding the Bounding Box

3.9.1 Definitions and theorems

Theorem 3.5. *The LP is unbounded or we can find the bounding box.*

Lemma 3.6. *$Ax \leq b$ maximize $C^T x$ is unbounded iff*

1) The LP is a feasible

2) $\exists d \,.\, C^T d > 0 \wedge Ad \leq 0$ (this is to slide the lines to the origin)

Proof. ($\Longleftarrow$)
By 1) we can pick $x \in \mathbb{R}^{m \times 1}$ s.t. $Ax \leq b$

By 2) we can find $d \in \mathbb{R}^{m \times 1}$ s.t. $C^T d \geq 0$ and $Ad \leq 0$.

Claim $\alpha d + x$ is feasible and can be arbitrarily large

Proof. $A(\alpha d + x) = \alpha Ad + Ax \leq Ax + b \leq b$ for all $\alpha \geq 0$, so for all $\alpha \geq 0$, $\alpha d + x$ is feasible.
$C^T(\alpha d + x) = \alpha C^T d + C^T x$ can be arbitrarily large. □

□

Proof. ($\Longrightarrow$)
(Hint: using the compactness argument)

First, since the LP is unbounded, by definition, it is feasible.

There exists $x_1, x_2, \ldots$ such that $Ax_i \leq b$ and $\lim_{i \to \infty} C^T x_i = \infty$. Let $S = \{d \mid C^T d > 0\} \neq \emptyset$ (??) □

3.9.2 Procedure to find d

WLOG, $\exists d$ s.t. $C^T d = 1$ and $Ad \leq 0$ is projected rows of A onto line $C^T x = 1$. (slide all constraints so that they pass the origin) and solve the 1D LP.

Note: $\exists d \iff \beta \geq \alpha$

Case i $\beta < \alpha$ then $Ax \leq b$ is feasible, thus $C^T x$ subject to $Ax \leq b$ is unbounded

Case ii $\beta = \alpha$ then either the LP is not feasible, or it is unbounded (because there are two parallel constraints

Case iii $\alpha < \beta$, then $h_\alpha \equiv$ half Space giving α, and $h_\beta \equiv$ half space giving β. h_α, h_β are the bounding box.

Lecture 4: Geometric Transformation

Instructor: Gary Miller   Notes taken: Yujie Xu
Date: Monday, Sep 09, 11, 2015

4.10 General Intro

The geometric transformation is related to the following problems:

- Linear Programming
- Convex Hull
- Delaunay Triangulation
- Voronoi Diagram
- Stereographic Map

4.11 Definitions

Definition 4.1. $\mathbb{R}^d$ *is a d-dimensional real space. A point p in a d-dimensional space is written as* $p = \begin{bmatrix} P_1 \\ \vdots \\ P_d \end{bmatrix}$

$||p||^2 = P_1^2 + \cdots + P_d^2 = P^T P = \begin{bmatrix} P_1 & \dots & P_d \end{bmatrix} \begin{bmatrix} P_1 \\ \vdots \\ P_d \end{bmatrix}$

Definition 4.2. *Hyperplane:* $\{x \in \mathbb{R}^d \mid P^T X = \alpha\}$
Halfplane: $\{x \in \mathbb{R}^d \mid P^T X \leq \alpha\}$
The P vector is the normal vector to the plane.

Definition 4.3. *(Reflection about unit sphere)*

$$Reflect(P) = \frac{P}{P^T P} = \frac{P}{|P|^2}$$

Remark: $P \in$ unit sphere (on the boundary) are fixed points, i.e. $P^T P = 1$

Remark: points inside the unit sphere is sent to the outside

Remark: points outside the unit sphere is sent to the inside

Lecture 5: Guarding a Polygon

Instructor: Gary Miller   Notes taken: Yujie Xu
Date: Friday, 2015

5.12 Guarding a Polygon

- Input: Polygon P.
- Output: k "guards" $p_1, \ldots, p_k$ inside P so that for all points of P, there exist a guard that can see it. i.e. we want
  - k guards that cover P
  - k is small

Theorem 5.1. *For a polygon P with n vertices, $\frac{n}{3}$ guards are necessary (Figure 5.6) and sufficient to cover P.*

5.13 $n/3$ - guard Alg

```
P' = Triangulate P
3-color P
    construct the geometric dual of T
    3-color P by traversing in a BF manner, color new vertices with the missing color.
Pick the least used color
//note: any color will do, but we want the least number of guards.
```

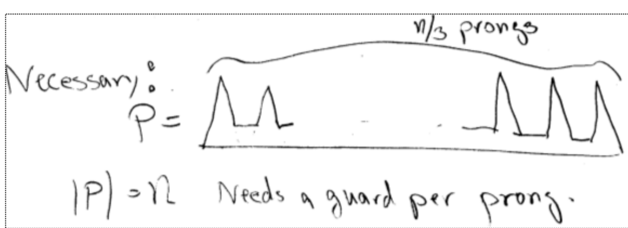

Figure 5.6: The case when $3/n$ guards are needed

Lecture 6: Fundamental Theorems for Convex Set

Instructor: Gary Miller   Notes taken: Yujie Xu

Date: Monday, Sep 28, 2015

Lecture 7: Triangulating a Polygon

Instructor: Gary Miller Notes taken: Yujie Xu
Date: Monday, Sep 28, 2015

Definition 7.1. *PSLG: Planar Straightline Graph, an embedding of a planar graph in the plane such that its edges are mapped into straight line segments (Wikipedia).*

Definition 7.2. *Polygonal chain: PSLG consisting of a simple cycle P.*

Claim A polygon chain has a unique interior

Definition 7.3. *Polygon: polygonal chain + interior*

Definition 7.4. *Triangulation: addition of line segment so that*

1. *It is stil PSLG*
2. *Interior is decomposed into triangles*

Theorem 7.5. *Every simple polygon can be triangulated*

Proof. Input: let m be the number of segments on the n points.

- Base case: $n = 3$

  Assuming there are no 180 degree, then we are done.

- Inductive case (a little tricky) $n > 3$,

  assume it holds for $m < n$, then

  Let v be the left most point with its neighbors w and u.

  – If $[u, w]$ interior to the polygon: we get two polygon: $P_1 = Triangle(u, v, w), P_2 = rest$, with $|P_1| = 3$ and $|P_2| = n - 1$, using IH to triangulate P_2

  – Otherwise, $S = \{v' \mid v'\text{interior to } Triangle(u, v, w)\} \neq \emptyset$, let v' be the left most of S. $[v, v']$ separates the polygon into P_1, P_2, with $|P_1| < n \wedge |P_2| < n$, one can triangulate the two.

□

Theorem 7.6. *Not every simple polygonal surface (polytope: flat sides) in 3D can be decomposed into tetrahedron.*

Here is a counter example.

For a prism with twisted top, let $B = face(a, b, c)$, consider the tetrahedreon with face B, one need to find the missing point. It could be x or y. It is not x, because edge $[a, x]$ is out of the prism. It is not y, because $[b, y]$ is out of the prism (?? what does this ultimately show ??)

Remark: In general: test if a polygonal surface is decomposable is NP-hard

References

[1] Mark de Berg, Otgried Cheong, Marc van Kreveld, and Mark Overmars. Computational Geometry : Algorithms and Applications. Springer Berlin Heidelberg, 2008.

[2] Wolfram Research, Inc. Convex set. web, August 2015. `http://mathworld.wolfram.com/ConvexSet.html`.