

1 Overview

Welcome to CS633 Computational Geometry. In this lecture I will talk about: What is computational geometry? To give an answer to this question, I will try to cover many applications in the computational geometry and provide you a set of resources available in the computational geometry community including software, webpages, and conferences/journals. We will go over the syllabus and focus on the possible applications and projects that we will have for this course. Finally, we will study our first computational geometry problem: computing convex hull.

This shall be a very exciting course and I hope you will learn a lot.

2 What is computational geometry about?

Computational Geometry is a study of **algorithms** and **data structures** for geometric objects. Geometric objects/problems are everywhere, thus computational geometry has many applications in the real world. For example, imagine you are given a box and 10 objects and you are asked to pack these 10 objects into the box. How do you know if these 10 objects will fit into the box? Even if you are told that you can pack these objects into the box, how do you know how to arrange these objects so that they can fit into the box? You may think this is a trivial problem. However, this problem, called the packing problem, is extremely difficult and occurs in domains such as manufacturing, transportation and robotics. In this section, we will look at some more applications and also resources that can help you succeed in this class.

2.1 Applications

See the power-point slides (will be posted online) for detail.

- Computer graphics: rendering, animation, video games
- Solid modeling ([CGS](#): Constructive Solid Geometry)
- Geometric processing: model simplification, compression, deformation
- Robotics: motion planning, manipulating
- Sensor network
- [GIS](#): Geographic information system
- [Origami design](#): folding and unfolding
- CAD/CAM prototyping
- Bioinformatics
- ...

2.2 Scopes

1. Line segment intersection
2. Triangulations (Art gallery problem)
3. Robot Motion planning and Visibility Graph
4. Range searching (K-d tree, Range tree, Binary space partitions and Quadtree/Octree)
5. Point location
6. Voronoi Diagrams
7. Delaunay Triangulations
8. Convex hull
9. Arrangements and Duality

2.3 Resources

There are lots of resources that you can find today on the Internet via Google. Here are just a few more important links to the software, webpages and publications.

2.3.1 Software

Here are just a few well known libraries: [CGAL](#) (computational geometry algorithm library), [LEDA](#) (combinatorial and geometric computing), [Qhull](#) (convex hull, voronoi, halfspace intersection), [FIST](#) (triangulation), [OOPSMP](#) (motion planning) . . .

Note: Major CG libraries (such as the ones listed above) are coded in C/C++. It is absolutely important that you have *working knowledge* of C/C++ before you decide to stay in this class. We will be using some of these libraries for the assignments.

2.3.2 Webpages

- [Geometry in Action](#) and [Geometry Junkyard](#) (maintained by [David Eppstein](#))
- [Computational Geometry Page](#) (maintained by [Jeff Erickson](#))
- [Computational Geometry on Web](#) (maintained by [Godfried Toussaint](#))
- [The Open Problems Project](#) (maintained by [Erik Demaine](#), [Joseph Mitchell](#) and [Joseph O'Rourke](#))

These are also good sources for your project topic.

2.3.3 Conferences and Journals

Journals:

- Computational Geometry: Theory and Applications
- Computational Geometry & Applications
- Discrete & Computational Geometry

Conferences:

- Proceedings of the ACM Symposium on Computational Geometry
- Proceedings of the Canadian Conference on Computational Geometry
- ...

These will be good sources for your paper review assignment.

3 Syllabus

This lecture is organized in to several lectures and student presentations. There will be quizzes, assignments and a final project. Please check your GMU email and our course webpage frequently for announcements and updates. Our course webpage is located at:

http://cs.gmu.edu/~jmlie/teaching/09-fall_cs633/

3.1 Prerequisite

CS583 or instructor's approval, which means you should know:

- sorting algorithms — quick sort, merge sort, ...
- graph algorithms — shortest paths, connected components, graph coloring, ...
- basic data structures — lists, trees, graphs, ...
- algorithm design techniques — divide-n-conquer, incremental improvement, ...
- algorithm analysis techniques — asymptotic notations, recursion, complexity classes ...
- Working knowledge of C/C++

3.2 Textbook

- *Computational Geometry: Algorithms and Applications* by Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Cheong, third revised edition, Springer-Verlag, 2008. ISBN # 3-540-65620-0. (You can also use the second edition; the differences are pretty minimum.)

- *Computational Geometry in C* by Joseph O'Rourke (Cambridge University Press; 2000 edition, ISBN # 0-521-64976-5) (**not required**)
- In addition, we will study papers from various journals and conferences; these will be made available electronically.

3.3 Grading

- Assignments — 50%: There will be homework assignments, programming assignments, and paper review assignments; see Section 3.4 for detail.
- Quizzes (and possibly a midterm exam) — 20%
- Course project presentations (proposal and final) — 30%

3.4 Assignments and Projects

All required assignments and projects should be completed by the stated due date and time. The total score of your assignment score will be 10 points less every extra day after the due date (i.e., the 100 total points will become zero after 10 days pass the due date).

3.4.1 Homework assignments

There will be several short homework assignments. They will typically be assigned during one class and due at the beginning of the next class. Generally, they will relate to material covered in class that day/week.

3.4.2 Programming assignments

There will two programming assignments. Most likely they will be the *art gallery problem* and *motion planning*. Part of the source code (in C/C++) will be given. The results of these assignments will be shown during the class.

3.4.3 Paper review assignments

There will be one assignment which involves writing a review of a research paper (with more than 6 pages) from a recent conference or journal. The particular paper will be selected by the student (in consultation with the instructor). Guidelines for the paper review will be available on the course webpage.

3.4.4 Final project

The goal of the project is to study in depth some issue related to computational geometry. Projects may range from an investigation of an open problem in computational geometry (solution not

required for a good grade...) to experimental studies of known algorithms. Project topics will be selected by the student (in consultation with the instructor) by the end of the first month of the course (date will be set). Projects may be done with partners (of course, more will be expected than if the project is done individually).

A list of suggested topics will be posted online soon.

3.5 Policies

All required assignments should be completed by the stated due date and time. The total score of your assignment score will be 20 points less every extra day after the due date (i.e., the 100 total points will become zero after 5 days past the due date).

Please note that all coursework is to be done independently (except the collaborative final project). Plagiarizing the homework will be penalized by maximum negative credit and cheating on the exam will earn you an F in the course. See the GMU Honor Code System and Policies at <http://www.gmu.edu/catalog/acadpol.html> and <http://cs.gmu.edu/wiki/pmwiki.php/HonorCode/HomePage>. You are encouraged to discuss the material BEFORE you do the assignment. As a part of the interaction you can discuss a meaning of the question or possible ways of approaching the solution. The homework should be written strictly by yourself. In case your solution is based on the important idea of someone else please acknowledge that in your solution, to avoid any accusations.

The quiz and exam will be a closed book exam - no notes will be allowed.

4 2D Convex Hull algorithms

We begin by describing the problem of computing the convex hull of (unorganized) points in two dimensions.

4.1 Convexity and Convex hull of 2D points

Definition 1. A shape S is convex if the line connecting any two points of S is in S .

What is a convex hull? Well, there are several definitions of convex hull.

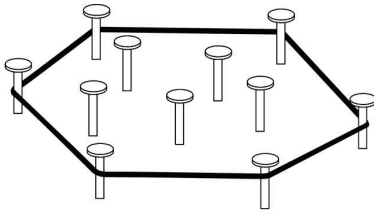
Definition 2. A convex hull of a set of 2D points P is the smallest **convex** polygon that encloses all the points of P .

Definition 3. A convex hull of a set of 2D points P is the intersection of all convex sets that contain P .

Definition 4. A convex hull of a set of 2D points P is the intersection of all half-planes that contain P .

Example: Convex hull of a set of 2D points

Example: Convex hull of a polygon



Like the sorting problem you learned in the algorithms course, computing convex hull is a fundamental problem in computational geometry and has applications in pattern recognition and collision detection. Computing convex hull is also used as basic operations in many other problems in computational geometry.

4.2 Data structure

Note that the convex hull of P is defined for the “area” that contains all the points of P . However, we normally use the boundary of the convex hull to represent the hull itself (called boundary representation or Brep).

Question: What kind of data structure should we use to represent the convex hull in 2D?

Well, we can use a list to store the vertices of the convex hull in the counter clockwise order. So, $CH = \{p_1, p_2, p_3, \dots, p_n\}$, where $p_i = (x_i, y_i)$.

However, edge of the convex hull is also an important property. (For example, when you try to check if a given point is inside a polygon or not.) An edge can be represented by a directed line and its two end points.

Definition 5. A directed line \vec{pq} is a line with direction from p to q .

Finally we define the convex hull as: $CH = \{V, E\}$, where $V = \{p_1, p_2, p_3, \dots, p_n\}$ and $E = \{e_1, e_2, \dots, e_n\}$, where $e_i = (p_i, p_{i+1})$.

Before going into the details of our first algorithm, lets define our problem more carefully.

Definition 6. 2D Convex Hull Problem: Given a set of n points P in two dimensions, compute the convex hull CH of P and represent CH using the data structure described in this section.

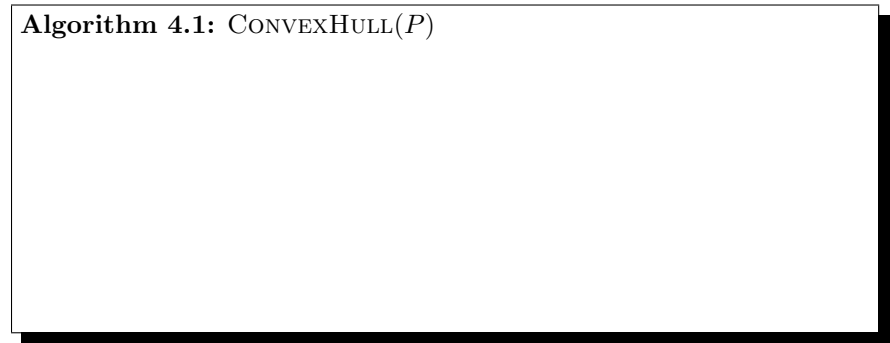
4.3 A brute force algorithm

Almost all algorithms, even brute force algorithms, start with some insights into the properties of the problem.

Observation 7. If we walk around the boundary of the convex hull in the counter clockwise direction, all points will always be on our left hand side.

Based on this observation, we can check if a pair of points and see if they will form an edge of the convex hull and if we can check all possible pairs of points from a given point set P then we will be able to compute the convex hull boundary of P . Algorithm 4.1 shows an outline of this approach.

Algorithm 4.1: CONVEXHULL(P)



Note that after we identify the edges of the convex hull using Algorithm 4.1, we still have to re-organize the edges so the convex hull can have the structure described in Section 4.2.

4.3.1 Time Complexity

The time complexity of Algorithm 4.1 is $O(n^3)$, where n is the size (number of points) of P . There will be $O(n^2)$ pairs and for each pair we will have to check $O(n)$ points before we find a point or no point on the right hand side of the directed line.

4.3.2 Degeneracies

Even though we know that Algorithm 4.1 is correct, it may not handle all possible inputs correctly.

Example: When points are collinear.

In many cases, in order to avoid considering the degeneracies in the input data, we will usually say “given the data in general position” (general-position assumptions). This allows us to consider important geometric properties first. This does not mean degenerate cases are not important. On the contrary, they are usually nontrivial to handle and may lead to many serious implementation issues. In fact, several researches have been focused on this problem (of how to handle degenerate cases properly).

4.3.3 Robustness

Another problem of Algorithm 4.1 is that we assume that we can always check if a point is on the right correctly. Unfortunately, this is not true because of the possible rounding errors during computation. This makes Algorithm 4.1 not robust.

How do we compute if a point is on the right?

To solve this problem, we can use packages that provide “exact arithmetic”, which, however, will slow down the computation furthermore.

4.3.4 Other brute force algorithms

This approach also depends on a simple observation.

Observation 8. *Given a point set P , points in the triangle formed by three arbitrary points of P are not points on the convex hull of P .*

This brute force algorithm is similar to Algorithm 4.1. We enumerate all possible triangles (there can be $O(n^3)$ such triangles) and discard points that are inside one or multiple triangles.

Another important convex hull algorithm is called “gift wrapping” [CK70]. In this algorithm, a new convex hull edge is appended to the currently known convex hull edge (just like wrapping a gift). The time complexity of the “gift wrapping” is $O(nh)$, where h is the number of edges of the convex hull. This algorithm is called **output sensitive** algorithm and is efficient when h is small.

4.4 An incremental algorithm

The next algorithm is an incremental algorithm, which means we will consider one point at a time and construct the convex hull for the points examined so far. This algorithm is based on another property of the convex hull.

Observation 9. *If we walk around the boundary of the convex hull (or any convex shape) in the counter clockwise direction, we will always make left turns.*

Based on this observation, we can check the points from left to the right one by one and see if the last three consecutive points make a left turn. If so, these three points are likely to be on the convex hull. If they do not make a left turn, then we know we have found something “outside” of the current convex hull and we have to fix it.

Well, we have to fix the “convex hull” so that the property in Observation 9 of the convex hull is kept. That is the so called “invariance”, which is very useful in developing an algorithm. To make the convex hull become valid, remove the *second last* point and check if the last three points is a left turn and repeat the process above.

Therefore, an important property of this (and most) incremental algorithm is that it construct the convex hull for the points that we have considered so far. We will also encounter several other incremental algorithms later.

Example:

Question: What *data structure* should we use for storing convex hull vertices in this algorithm?

Algorithm 4.2: CONVEXHULL(P)

Algorithm 4.3: HALFHULL(s, e, S, P)

4.4.1 Time Complexity

The time complexity of Algorithm 4.3 is $O(n)$ because the number of “turn checking” depends on the number of points added to and deleted from the stack. Since each point can only be added once and delete once, the number of “turn checking” is $O(n)$.

The time complexity of Algorithm 4.2 is therefore dominated by the sorting of points, which takes $O(n \log n)$ time.

4.4.2 Degeneracies

Does it handle collinear points correctly? Mostly, yes, however, we should still be aware of that we will need to sort the points in the lexicographic order and the left turn should be a strict left turn.

Example:

4.4.3 Robustness

What happened if rounding errors occurred when we check if points make a left turn. Well, it's possible that a right turn is classified as a left turn. In this case, you will see a small dent. If a right turn is classified as a right turn, then you will see (however hardly noticeable) a point left on the outside of the convex hull.

In any case, the “convex hull” constructed by Algorithm 4.2 is still a (closed) polygon and is therefore more robust than Algorithm 4.1.

Example:

4.4.4 Other incremental algorithms

Algorithm 4.2 is similar to the algorithm proposed by [Ronald Graham](#) in 1972 (called Graham Scan) [Gra72]. In Graham scan, the points are sorted using the *angles* to a pre-selected point instead of using the coordinates. The problem of Algorithm 4.2 and Graham Scan is their difficulty of extending to higher dimensions.

Example:

Another way of building the convex hull incrementally is to compute the tangent lines (planes) from a point to the convex hull and remove the edges that are facing the new point. Initially, this algorithm seems to have $O(n^2)$ time complexity. However, it can be done in $O(n \log n)$ time if the points are pre-sorted from left to right.

Example:

4.5 A divide-n-conquer algorithm

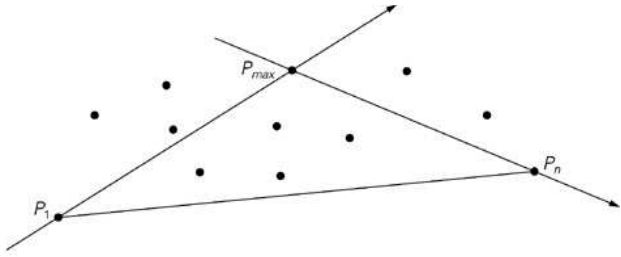
Here we consider a divide-and-conquer algorithm called **quickhull** [Edd77, Byk78]. The idea of quickhull is similar to quicksort. (Do you remember what quicksort is?) The algorithm is based on the following observations.

Observation 10. *The leftmost and rightmost points in P must be part of the convex hull.*

Observation 11. *The furthest point away from any line must be part of the convex hull.*

Observation 12. *Points in the triangle formed by any three points in P will **not** be part of the convex hull.*

Algorithm 4.4 first computes a line that connects the left- and right-most points of P . This line will separate the points into two groups. For each group we will compute a hull.



Algorithm 4.4 takes a line and a set of point. The points are all on the same side of the line. Then a triangle is formed by the line and the furthest point from the line. All points inside the triangle are removed. The remaining points are divided into two groups. (why is it two groups? why not three?) Algorithm 4.4 is called recursively until no points remain.

Example:

Algorithm 4.4: QHULL($P[1 \dots n]$)

comment: P is a set n points

There is an animation of quickhull available at:

http://www.cs.princeton.edu/~ah/alg_anim/version1/QuickHull.html

4.5.1 Time Complexity

Similar to quicksort, in the worst case (when the points are all on the convex hull boundary), quickhull takes $\Theta(n^2)$ time. However, quickhull is also an output sensitive algorithm, i.e., more accurate time complexity should be $O(nh)$, where h is the size of the convex hull.

4.5.2 Degeneracies

What happens if the points are collinear?

4.5.3 Robustness

What happens if the points are classified as inside (resp., outside) the triangle when they are in fact outside (resp., inside)?

4.5.4 Other divide-n-conquer algorithms

The quickhull algorithm focused on the “dividing” process in this divide-n-conquer paradigm. On the contrary, Preparata and Hong [PH77] proposed an divide-n-conquer algorithm that focused on the “merging” process. That is they split the points into two (almost) equally sized points and compute the convex hulls separately for them, and then obtain the final answer by merging the two convex hulls. Their method takes only $O(n \log n)$ time and can be extended to three dimensions fairly easily. However, their method is pretty tricky, especially when merging the convex hulls, which need to be done in linear time in order to obtain the final $O(n \log n)$ time complexity.

Example:

5 Conclusion

In this lecture we went over some applications in computational geometry. We also learned about several algorithm to compute the convex hull from points. This lecture essentially covers the first chapter of the textbook.

Note that even though the problem of computing the convex hull from points has time complexity lower bound $O(n \log n)$ (this can be shown by reducing the sorting problem to the convex hull problem), convex hull can be computed more efficiently (in linear time) when the input is a polygon [MA79, Mel87, GY83].

References

- [Byk78] A. Bykat. Convex hull of a finite set of points in two dimensions. *Inform. Process. Lett.*, 7:296–298, 1978. 4.5
- [CK70] D. R. Chand and S. S. Kapur. An algorithm for convex polytopes. *J. ACM*, 17(1):78–86, January 1970. 4.3.4
- [Edd77] W. F. Eddy. A new convex hull algorithm for planar sets. *ACM Trans. Math. Softw.*, 3:398–403 and 411–412, 1977. 4.5
- [Gra72] R. L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Inform. Process. Lett.*, 1:132–133, 1972. 4.4.4
- [GY83] R. L. Graham and F. F. Yao. Finding the convex hull of a simple polygon. *J. Algorithms*, 4:324–331, 1983. 5

- [MA79] D. McCallum and D. Avis. A linear algorithm for finding the convex hull of a simple polygon. *Inform. Process. Lett.*, 9:201–206, 1979. 5
- [Mel87] A. Melkman. On-line construction of the convex hull of a simple polyline. *Inform. Process. Lett.*, 25:11–12, 1987. 5
- [PH77] F. P. Preparata and S. J. Hong. Convex hulls of finite sets of points in two and three dimensions. *Commun. ACM*, 20:87–93, 1977. 4.5.4