CPS234 Computational Geometry **August 30, 2005**

Lecture 1: Introduction to Computational Geometry

Lecturer: Pankaj K. Agarwal *Scribe: Amit K. Patel*

1.1 Course Overview

Professor:	**Teaching Assistant**:
Pankaj K. Agarwal	Amit K Patel
LSRC, Room D315	LSRC, Room D103
(919) 660-6548	(919) 660-6501
pankaj@cs.duke.edu	akpatel@cs.duke.edu

Welcome to CPS 234, Computational Geometry! Computational geometry studies the design, analysis, and implementation of algorithms and data structures for geometric problems. These problems arise in a wide range of areas, including CAD/CAM, robotics, computer graphics, molecular biology, GIS, spatial databases, sensor networks, and machine learning. In addition to the tools developed in computer science, the study of geometric algorithms also requires ideas from various mathematical disciplines, e.g., combinatorics, topology, algebra, and differential geometry. This close interaction between various mathematical and practical areas has had a beneficial impact on both basic and applied research in computational geometry.

1.1.1 Course content and syllabus

The goal of this course is to provide an overview of the techniques developed in computational geometry as well as some of its application areas. The topics covered in the course will include:

1. Geometric Fundamentals: Models of computation, lower bound techniques, geometric primitives, geometric transforms
2. Convex hulls: Planar convex hulls, higher dimensional convex hulls, randomized, output-sensitive, and dynamic algorithms, applications of convex hull
3. Intersection detection: segment intersection, line sweep, map overlay, halfspace intersection, polyhedra intersection
4. Geometric searching: segment, interval, and priority-search trees, point location, persistent data structure, fractional cascading, range searching, nearest-neighbor searching
5. Proximity problems: closest pair, Voronoi diagram, Delaunay triangulation and their subgraphs, spanners, well separated pair decomposition

6. Arrangements: Arrangements of lines and hyperplanes, sweep-line and incremental algorithms, lower envelopes, levels, and zones, applications of arrangements
7. Triangulations: monotone and simple polygon triangulations, point-set triangulations, optimization criteria, Steiner triangulation, Delaunay refinement
8. Geometric sampling: random sampling and ϵ-nets, ϵ-approximation and discrepancy, cuttings, coresets
9. Geometric optimization: linear programming, LP-type problems, parametric searching, approximation techniques
10. Robust geometric computation: perturbation techniques, floating-point filters, rounding techniques

A tentative lecture plan is provided in Table 1.1. The most current and definitive syllabus will be maintained on the course web page *http://www.cs.duke.edu/education/courses/fall05/cps234/*) under the heading "Lectures" so students should consult the course web page for the most recent information. In addition to the lecture plan, the course web page will keep track of when assignments are issued and due, as well as who is in charge of lecture scribing a given lecture.

Prerequisites Students are expected to have previous exposure to basic algorithmic techniques and data structures. References to necessary background will be provided but otherwise, the course is self-contained.

1.1.2 Administrative details

Course web page and mailing list The course webpage will be maintained at *http://www.cs.duke.edu/education/courses/fall05/cps234/*). It will contain all the relevant material for the course, including lecture notes, assignments, handouts, and further reading material. There will also be a mailing list for the course called "cps234", which the instructor and TA will use to make announcements. Please make sure that your name is on the list.

Meeting times Class meets every Tuesday and Thursday afternoon from 2:50–4:05, except for October 11 (Fall Break) and November 24 (Thanksgiving). Any changes will be announced later in the class. The last day of lecture will be November 29. Students will present their final project reports during reading period.

Course materials There will be no mandatory textbook for the course. The recommended textbook is *Computational Geometry: Algorithms and Applications* but no single textbook seems ideal at this juncture. That being said, there are a number of valuable texts that serve different purposes. Every student is encouraged to purchase the text that they would find most useful. Here is a list of a few books containing material covered in class.

- M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications. Springer-Verlag, 2nd edition*, 2000. (course textbook)
- H. Edelsbrunner. *Geometry and Topology for Mesh Generation*. Cambridge Univ. Press, England, 2001.

Lecture	Date	Topic
		Geometric Fundamentals
1	08/29	Models of computation, geometric primitives, lower bounds
		Convex Hull
2	09/01	2D convex hull
3	09/06	Convex hull in high dimensions
4	09/08	Randomized incremental construction (RIC) for convex hull
		Intersection Detection
5	09/13	Segment intersection: sweep-line algorithm, RIC
6	09/15	RIC analysis, polyhedra intersection
		Geometric Data Structures
7	09/20	Segment, interval, priority-search trees, quad trees
8	09/22	Orthogonal range searching
9	09/27	Planar point location
		Proximity Problems
10	09/29	Voronoi diagrams (VD), Delaunay triangulation (DT)
11	10/04	RIC for DT, subgraphs of DT
12	10/06	Spanners, well seperated pair decomposition
		Arrangements
13	10/13	Arrangements of lines and curves, lower envelope, zone theorem
14	10/18	Levels and incidences in arrangements
15	10/20	Arrangements of hyperplanes
		Triangulation
16	10/25	Polygon triangulation
17	10/27	Point-set triangulation, Delaunay refinement
		Geometric Sampling
18	11/01	Random samplings, ϵ-nets and ϵ-approximations
19	11/03	Cuttings and their applications
20	11/08	Simplex range searching
21	11/10	Coresets
		Geometric Optimization
22	11/15	Linear Programming
23	11/17	LP-type problems
24	11/22	Parametric searching
		Robustness Issues
25	11/29	Perturbation methods, rounding, filters

Table 1.1: Tentative lecture plan

- J. E. Goodman and J. O'Rourke (eds.), *Handbook of Discrete and Computational Geometry, 2nd edition*, CRC Press LLC, Boca Raton, FL, 2004.
- K. Mulmuley, *Computational Geometry: An Introduction Through Randomized Algorithms*, Prentice Hall, Englewood Cliffs, NJ, 1994.
- J. O'Rourke, *Computational Geometry in C, 2nd edition*, Cambridge University Press, New York, 1998.
- J.-R. Sack and J. Urrutia (eds.), *Handbook of Computational Geometry*. Amsterdam, Netherlands: North-Holland, 2000.

In addition, a number of web sites, papers, and other reading resources will be provided during the semester. These will appear on the course webpage; all course materials will be available through the course web site *http://www.cs.duke.edu/education/courses/fall05/cps234/* under the heading "Reading;" other web links may be also be collected under the heading "Links".

Finally, lecture notes for every lecture will be made available on the course webpage, under the heading "Lectures" within ten days of the given lecture.

Scribing lecture notes Because there is no one textbook covering all class material, course lecture notes will be a very important resource for everyone. For this reason, they should be produced in a detailed and comprehensive manner.

Each student taking the course will act as a scribe for at least two lectures. The scribe for a given lecture will be responsible for taking detailed notes during the lecture, typing up those notes in LaTeX, and submitting the notes *one week* after the lecture (or sooner). Notes should be written lucidly and be extremely comprehensive—they will serve as an authoritative summary and reference of class material for use by current and future students (as well as the instructor and TA).

The scribing template file `scribing.template.tex` can be downloaded from the course webpage under the heading "Links." The `scribing.cls` file must also be available in order for the template file to compile properly in LaTeX. Both files can be found at the course web page. If you click on the `scribing.cls` file link and are not prompted to download the file, but the contents are instead shown in your browser, you can simply copy and paste them into a text editor and save the file as `scribing.cls` .

With these files in place, you can begin typing up your notes in LaTeX with your favorite text editor by adding your notes to `scribing.template.tex` in the appropriate section. You are encouraged to use Xfig to draw pictures. See the TA for help on using LaTeX or Xfig.

1.1.3 Submitting scribed lecture notes

Please rename your .tex file to `lectureNN.tex` , where `NN` is the lecture number (please use two digits: use `lecture03.tex` for Lecture 3, for example). Importantly, your .tex file must compile before you submit it! To ensure this, try compiling your .tex file with the following commands:

```
latex  lectureNN.tex
latex  lectureNN.tex
dvips  -o lectureNN.ps    lectureNN.dvi
```

or alternatively,

```
pdflatex   lectureNN.tex
pdflatex   lectureNN.tex
```

Running latex/pdflatex twice is necessary, not a typo. If all this works and your postscript (.ps) or PDF (.pdf) file looks good when you view it, then submit the .tex and .ps or .pdf files, along with any necessary image files in both .jpg and .eps formats, as discussed in `scribing.template.tex`. Completed scribed lecture notes must be submitted to the TA within one week after the lecture (or sooner). The instructor and TA will go through them and will ask you to revise the notes if necessary. The revised version will be due within two-three days. Since students will rely on these notes for the course material, it is essential that the notes be submitted on time.

1.1.4 Getting help with LaTeX

How many of you are familiar with LaTeX?

If you are not familiar with LaTeX, the document entitled *The Not So Short Introduction to* LaTeX 2_ε should be a wonderful start. Despite its name, it is very compact, but remains fairly comprehensive. It is available on Blackboard. There are other books and web sites available if you need further assistance. Email the instructor or the TA and we can provide more information.

All of this information on LaTeX and instructions for submitting scribed lecture notes is available at the course web site under the heading “Links.”

1.1.5 Grading

The final grade will be based on your performance in home work problems, research project, and scribing lectures.

Problem sets: 30% of grade There will be three to four problem sets that each student has to complete individually. You will have roughly two weeks for each problem set.

No collaboration is permitted on problem sets. However, if you have worked for a while on a particular problem and have encountered a mental wall, and if you have banged your head against it for a while, it is permissible to consult others to make progress—that is better than stopping. But it should remain understood that this interaction must be one of consultation and not collaboration: hints rather than answers. After consultation, it is expected that you should still have some thinking to do. If you do not understand the material on which the assignments are based, contact the TA or me. In addition, if you do consult with another student, you must cite this in your solution.

Research project: 40% of grade There will be one research project to be completed individually. The intention is for the project work to be of publishable or nearly publishable quality—you might as well get something substantive out of the experience. Each student is encouraged to develop their own projects, but a number of topics will be discussed in the class. You should start thinking about the project as soon as possible.

The project will consist of two parts. The first part, which will be due immediately after the fall break, will require each student to submit a proposal, literature survey, and initial work to support the proposed research. The final deliverables, due in the first week of December, will be:

- A well-written paper for submission to the instructor (the primary measure of assessment will be the quality of the methods, analysis, and results in the paper; however, the paper should also be well-written and express your results lucidly).
- A short presentation to the class highlighting the major contributions of the paper (the primary measure of assessment will be the clarity with which you present your ideas and your ability to effectively communicate your results to your peers).

Lecture scribing: 30% of grade All students are required to serve at least twice as a lecture scribe. Your scribed lecture notes will be given a score of 0-20, based on the degree of comprehensiveness and care with which they were produced. Students may need to consult additional resources and create text or figures to supplement the lecture content itself in order to produce a comprehensive set of notes that merit full credit. In other words, the scribed notes should be more than was covered in class in order to receive full credit. There will be opportunities later in the semester for students to scribe a third time if they wish to earn additional points in this category.

Scribed lecture notes should be completed individually, but you can certainly ask your peers to clarify any point that your notes leave unclear: it is better to have a more complete and correct set of notes than to muddle something. In addition, you are of course welcome to seek LaTeX typesetting assistance from any source if you need it.

1.2 Problem Primitives and Computational Models

Geometry is a broad area requiring many lifetimes to study in full. We concentrate on the foundations of computational aspects of geometry a field that emerged in the late 1970's. We are concerned about developing and analyzing geometric algorithms. These are algorithms answering decision problems or computing geometric structures given as input a set of geometric objects. In this class, we concentrate on two broad categories of problems – *one-shot* algorithms and *query* algorithms.

One-shot algorithms look like the following.

Input: a set S of geometric objects
Output: some geometric/topological/combinatorial structure or the answer to a decision problem

There are several ways of measuring the performance of an algorithm. The two such measures are running time and space. For one-shot problems, we will mostly be concerned with worst-case time performance but occasionally, expected running time as well.

Sometimes, by allowing the use of more space, the running time of certain problems is reduced. This is seen in *query* problems. Query problems look like the following.

Input: a set S of geometric objects and a query q
Output: answer to the query q

Given S we do a series of similar queries. The goal is to answer each query as fast as possible. Redoing work done for previous queries is therefore undesirable. To remedy this, S may be preprocessed to make each query fast. Time and, to a lesser extent, space required to reprocess is secondary to query time. Often times, the query time is measured with respect to space required by the preprocessing step.

Computational models How does one measure running time? The most common way is to measure CPU time. However, one can imagine an algorithm requiring more time doing memory operations or I/O operations than arithmetic operations. For these cases, counting memory operations or I/O operations is a better measure of performace than counting CPU cycles.

The asymptotic running time of an algorithm depends on the machine used. In order to compare the efficiency of two different algorithm, we are forced to define a standard model of a computer or a computational model. Here is a list of some computational models you may have seen.

- Turing machine
- RAM
- Decision tree
- Quatum Turing machine
- Pointer machine model

RAM is the most common model for measuring running time. In a RAM model, each read and write operation to storage takes constant time. Each arithmetic operations also takes constant time. The pointer machine model is motivated by graph algorithms. The "memory" in this model is a linked list of nodes where each node is capable of doing certain basic operations.

When looking at running times, it is important to consider both the upper bound and lower bound. If your algorithm has an $O(n^2)$ running time, can you do better? For algorithms that output a geometric structure, the complexity of this structure immediately implies a lower bound.

Partial sums Given an array $S = \{a_1, \ldots, a_n\}$ of n real numbers, compute $Q(l, r) = \sum_{i=l}^{r} a_i$. Without any preprocessing, each query requires $O(n)$ addition operations.

Can we preprocess so each query takes only constant time? Construct an array $\widehat{S} = \{\sigma_1, \ldots, \sigma_n\}$ such that $\sigma_i = \sum_{j=1}^{i} a_j$. For a query $Q(l, r)$, return $\sigma_r - \sigma_l$. Each query requires $O(1)$ operations – two reads and one subtraction. The preprocessing can be done with $O(n)$ additions.

What if we are not allowed to use the subtraction operation. How fast can we do a query? We may create an $n \times n$ array $\widehat{S}$ such that $\widehat{S}_{ij} = \sum_{k=i}^{j} a_k$. Queries take $O(1)$ time but the prepocessing takes $O(n^2)$ time and space.

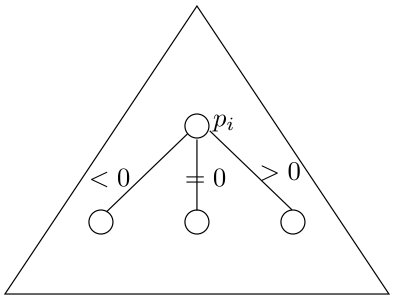

Figure 1.1: An algebraic decision tree: Node i evaluates a polynomial $p_i(a_1, \ldots, a_n)$ of bounded degree. The result of p_i determines the branching of each computation.

How fast can we do a query if we restrict ourselves to $O(n)$ space? We can do queries in $O(\lg n)$ time. The idea is the following. Construct a binary tree T with $a_1, \ldots, a_n$ as its leaves. In each internal node, store the sum of the leaves of both its substrees. Andrew Yao [Yao82] shows an $O(\alpha(n))$ parital sum query algorithm when the values are from a semigroup (substraction not allowed). Seven years later Chazelle and Rosenberg [CR89] prove the matching lower bound.

Lower bounds and algebraic decision tree models Many times we may choose a particular set of operations ignoring others and count the required number of those operations to solve a problem. This gives a lower bound on the problem.

Consider a decision tree T computing a decision problem on input $S = \{a_1, \ldots, a_n\}$. Node i in the tree computes $p_i(a_1, \ldots, a_n)$. The computation takes one of three branches depending on whether p_i is 0, less than 0, or greater than 0. See Figure 1.1. If $p_i(a_1, \ldots, a_n)$ is a polynomial of constant degree, then T is called an *algebraic decision tree*. The comparison tree used to show the lower bound on comparison sorting is a linear decision tree.

For a decision problem $f(a_1, \ldots, a_n)$ with $a_i \in \mathcal{R}$, the language accepted by f is

$$W = \{(a_1, \ldots, a_n) \mid f(a_1, \ldots, a_n) = 1\}.$$

Let $\#w$ be the number of connected components in W and β_i the ith Betti number of W.

Theorem 1 *(Ben'Or [BO83]) Let f be a decision problem. The worst-case running time to compute f under the algebraic decision tree model is $\Omega(\lg_d \#w - n)$.*

Theorem 2 *(Yao [Yao94]) Let f be a decision problem. The algebraic decision tree computing f takes time greater than $\Omega(\log \beta_i - n)$ for each i.*

References

[BO83] *Michael Ben-Or. Lower bounds for algebraic computation trees. In* STOC '83: Proceedings of the fifteenth annual ACM symposium on Theory of computing, *pages 80–86, New York, NY, USA,*

1983. ACM Press.

[CR89] *B. Chazelle and B. Rosenberg. Computing partial sums in multidimensional arrays. In* SCG '89: Proceedings of the fifth annual symposium on Computational geometry*, pages 131–139, New York, NY, USA, 1989. ACM Press.*

[Yao82] *Andrew C. Yao. Space-time tradeoff for answering range queries (extended abstract). In* STOC '82: Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 128–136, New York, NY, USA, 1982. ACM Press.*

[Yao94] *Andrew C. Yao. Decision tree complexity and betti numbers. In* STOC '94: Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 615–624, New York, NY, USA, 1994. ACM Press.*