Last Lecture: Quick Review & What I Do

Lecture 27
CS211 – Fall 2005

Announcements

- Course Evaluations
  - Worth one assignment point
    - Will count as part of your course grade
    - Not a bonus point
  - Must be done by Dec 4
  - http://www.engineering.cornell.edu/courseeval
- Game Design Open House
  - www.cs.cornell.edu/projects/game
  - Tuesday, December 6, 2005 in Upson 319 from 3-6pm
- Check the course website for additional announcements as the Final Exam approaches
  - Consulting ends this week
  - Office hours continue until Final Exam
    - Any changes will be announced on the course website

Quick Overview

- Programming concepts
  - We use Java, but the goal is to understand the ideas rather than to become a Java expert
  - Recursion
  - Object-Oriented Programming
  - Interfaces
  - Graphical User Interfaces (GUIs)
- Data structure concepts
  - The goal here is to develop skill with a set of tools that are widely useful
  - Induction
  - Asymptotic analysis (big-O)
  - Arrays, Trees, and Lists
  - Searching & Sorting
  - Stacks & Queues
  - Priority Queues
  - Sets & Dictionaries
  - Graphs
  - Union/Find

Programming Concepts

- Recursion
  - Stack frames
  - Exceptions
- Object-oriented programming
  - Classes and Objects
    - Primitive vs. reference types
  - Inheritance
    - Overriding vs. overloading
- Interfaces
  - Type hierarchy vs. class hierarchy
  - Upcasting vs. downcasting
  - The Comparable interface
  - Iterators
- GUIs
  - Components, Containers, & Layout Managers
  - Events & listeners

Data Structure Concepts

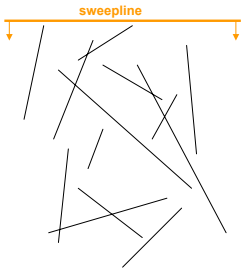
- Induction
- Grammars & parsing
- Asymptotic analysis (big-O)
  - Solving recurrences
  - Lower bounds on sorting
- Basic building blocks
  - Arrays
  - Lists
    - Singly- and doubly-linked
  - Trees
    - Binary Search Trees (BSTs)
- Searching
  - Linear- vs. binary-search
- Sorting
  - Insertion-, Selection-, Merge-, Quick-, and Heap-sort
- Useful ADTs (& implementations)
  - Stacks & Queues
    - Arrays & lists
  - Priority Queues
    - Heaps
    - Array of queues
  - Sets & Dictionaries
    - Bit vectors (for Sets)
    - Arrays & lists
    - Hashtables
    - BSTs & balanced BSTs
  - Union/Find
    - "Reverse" trees
    - Weighted union
    - Path compression
  - Graphs…

Overview of Graphs

- Implementations
  - Adjacency matrix
  - Adjacency list
- Topological sort
- Coloring & planarity
- Searching (BFS & DFS)
- Dijkstra's shortest path algorithm
- Minimum Spanning Trees (MSTs)
  - Prim's algorithm (growing a single tree)
  - Kruskal's algorithm (build a forest by adding edges in order)

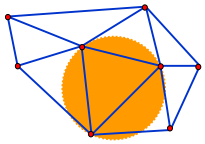
What I Do: Computational Geometry

- Using a computer to solve geometric problems
  - Get to use lots of data structure ideas
  - Example
    - Given n line segments in the plane, report all intersections
    - Uses both a PQ and a Balanced Tree
- Areas I work in
  - Motion Planning
  - Meshing
  - Shape Matching
    - computer vision
    - protein matching
  - More theoretical questions



The Delaunay Triangulation

- Has the "Empty Circle Property" (each Delaunay triangle's circumcircle is empty)
- Is commonly used for *meshing*



initial sites

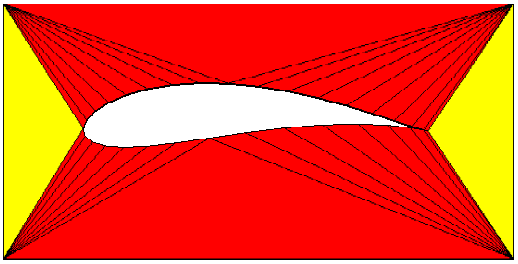
a Delaunay triangulation

an empty circle

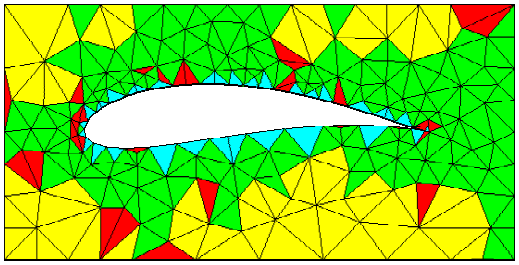
Meshing Requirements

- Control of element density
  - Small elements (in "interesting" regions) for accuracy
  - Large elements (elsewhere) for efficiency
- Allow internal boundaries
  - Needed to represent, for example, a crack
- Ideally: guarantee of element quality
  - Nice, but unnecessary for a single mesh
    - You can always fix it up "by hand"
  - But often need many meshes as geometry changes over time

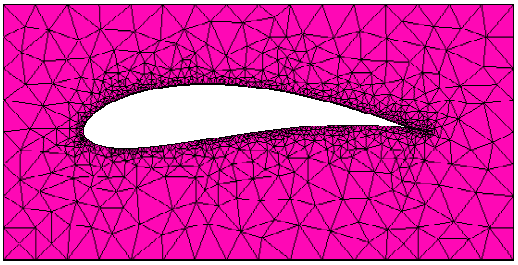
Initial Crude Mesh



During Improvement

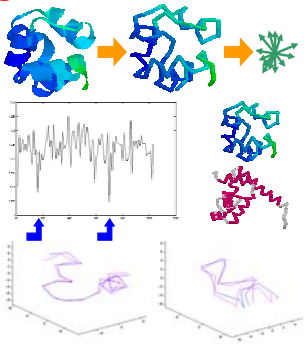


Final Mesh



Protein Shape and URMS

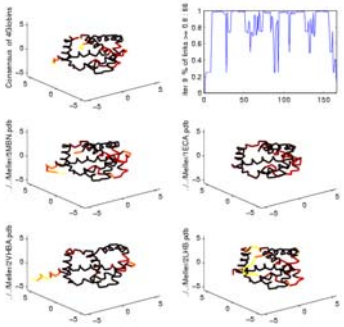
- Protein function is largely based on the protein's geometric shape
- How do we analyze protein shape?
- Our technique: URMS (Unit-vector Root Mean Square distance)
- Advantages
  - Insensitive to outliers
  - Efficient to compute
  - Equal weight for all portions of protein



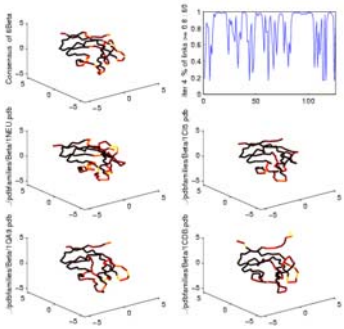
Protein Families & Consensus Shape

- Evolution ⇒ a protein ancient ancestor evolved into a family of proteins
- Membership in a protein family is expressed by sequence similarity, but is more strongly expressed by structure similarity
  - 25-30% sequence resemblance (almost always) ensures shape resemblance
- Goal: Create a *Consensus Shape Algorithm* that produces
  - A multiple alignment of structures, and
  - A single (core) structure that summarizes the structural information for a protein family

An Alpha Protein Family (Globins)



A Beta Protein Family



Unrelated Proteins