Garbage collection; lightstick orientation

Prof. Ramin Zabih

http://cs100r.cs.cornell.edu

Cornell University
Computer Science

Administrivia

- Assignment 3 is due Friday
- Prelim on 10/11
  - Review on Wed 10/10 in section (by Gurmeet)
  - Coverage through Wed 10/10

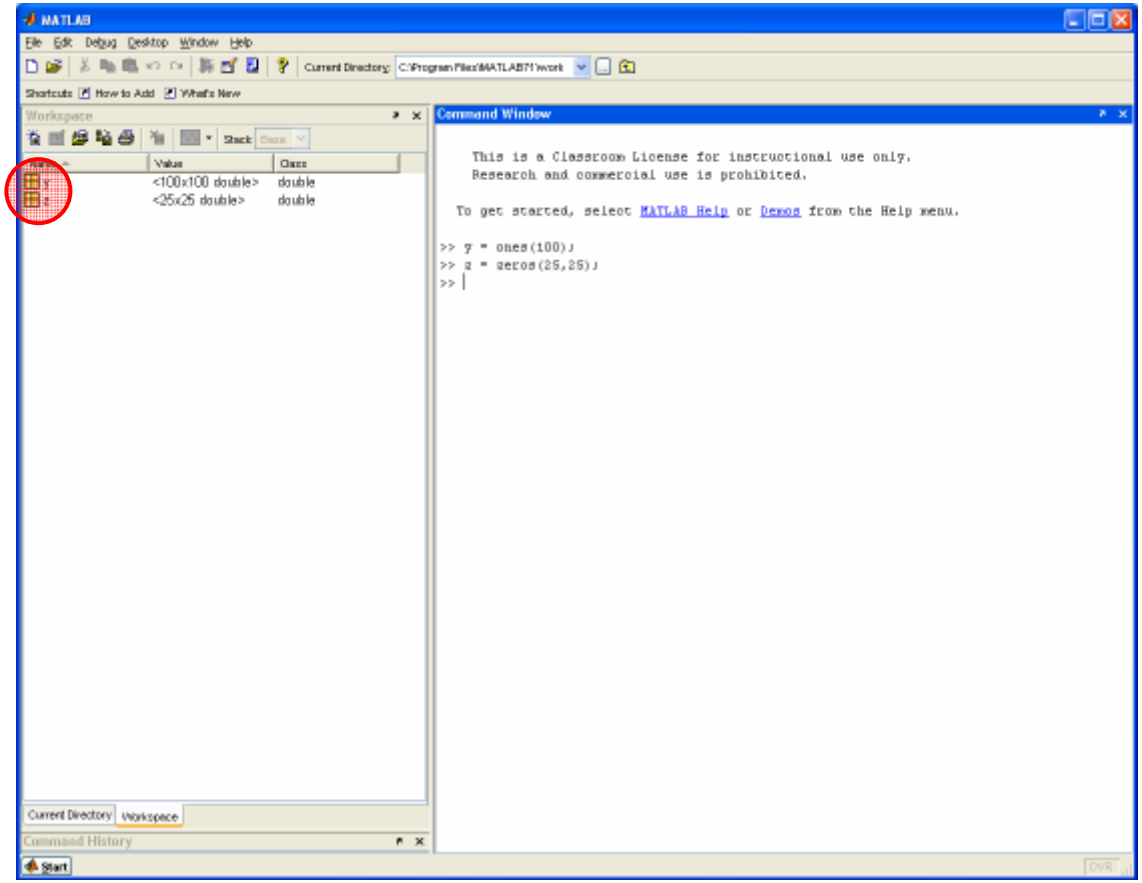
Manual storage reclamation

- Programmers always ask for a block of memory of a certain size
  - In C, explicitly declare when it is free
- Desirable but complex invariants:
  - Everything should be freed when it is no longer going to be used
  - And, it should be freed exactly once!
  - Also want to minimize fragmentation
- Any serious C programmer writes their own memory allocation system!

Automatic storage reclamation

- First challenge: figure out what memory locations are in use by the programmer
  - They are the ones that the programmer can get to (i.e., in a linked list currently in use)
  - Anything that has a name the programmer can get to (this is explicit in Matlab)
  - Start with something that has a name, then chase pointers as far as you can go
  - Some programs can access everything…
    - These languages don't allow such programs!

Root set in Matlab



How to garbage collect?

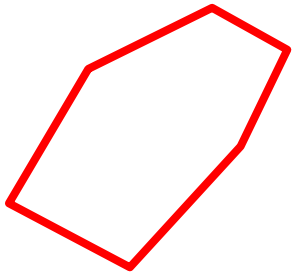
- Need to distinguish a value from a pointer
  - Garbage collector doesn't know anything about the programmer's data representation
- Usual solution: use highest-order bit
  - The pointer to location 3 is represented as $2^n + 3$, where n is usually 31 (word size – 1)
  - Some computers actually had hardware support to tell pointers from values
    - Nice idea that died out in the 90's

Simple algorithm: mark-sweep

- Chase the pointers from the root set until you are done
  - Modifying everything as you go ("mark")
    - Typically, set a bit
- Everything not marked is garbage, and can go back on the free list
- Problem: typically a lot more garbage than non-garbage, and need to examine all the garbage with mark-sweep
  - Solutions exist, but are somewhat complex

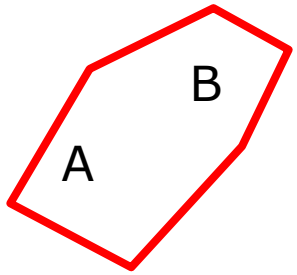
Back to lightstick shape

- We have a blob of red pixels
  - Now what?
- We can easily check which pixels are at the edge of the region
  - How does this help us figure out the shape?
- Easy solution: build a polygon
  - With a small number of vertices



Convex polygons

- A polygon is **convex** if, for any two points A,B in the polygon, all the points between A and B are also in the polygon



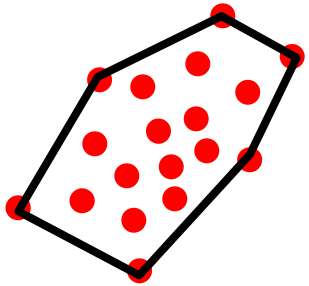
  - Corresponds to your intuitive notion of convex
    - More or less…
  - Convex things turn out to matter a lot!

Testing convexity

- How can we test if a polygon P is convex?
  - The obvious way is pretty dumb
    - As is so often the case (in 100R, and in reality)
- Consider the smallest convex polygon containing P
  - Called the **CONVEX HULL**
  - What is the convex hull if P is convex?

Convex hull of point sets

- What is the smallest convex shape containing a bunch of points?
  - I.e., our red pixels?



- State of the art in lightstick orientation!
  - We still need to compute polygon orientation

Computing convex hull

- How do we actually compute the convex hull of a set of points?
  - Note that it is easy to identify at least a few points that are part of the convex hull
  - Which ones are they?
- More precisely, the bounding box is related to the convex hull
  - It touches at 4 points
  - Can it touch at more?

Gift-wrapping algorithm

- Convex hull: smallest convex polygon containing the points
- Gift-wrapping:
  - Start at lowest point
  - Find new point such that all the other points lie to the left of the line to it
    - I.e., the largest angle
  - Repeat