Polygons and the convex hull

Prof. Noah Snavely
CS1114
http://cs1114.cs.cornell.edu

Cornell University
Computer Science

Administrivium

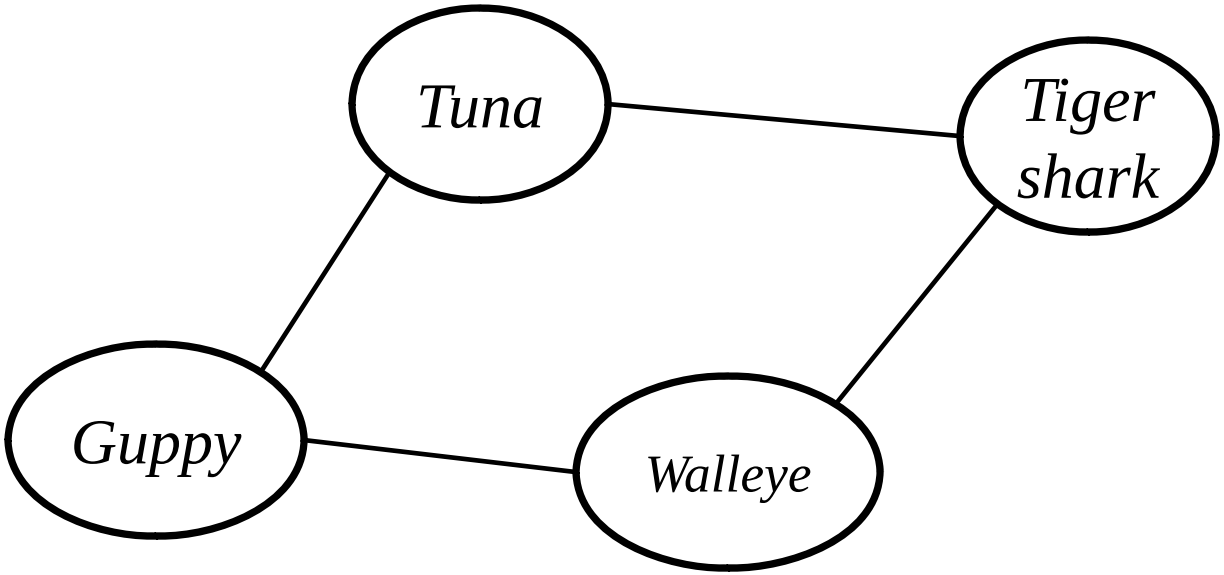
- Assignment 3 due this Friday by 5pm
  - Please sign up for slots on CMS
  - The last problem has you controlling the robots with the camera; you'll need to use the **`robotGetFrame`** function

Fish and graphs

- Have to find a home for *n* fish
- Some of the fish would each other fish
- How do we know how many tanks to order?

Fish and graphs: Solution 1

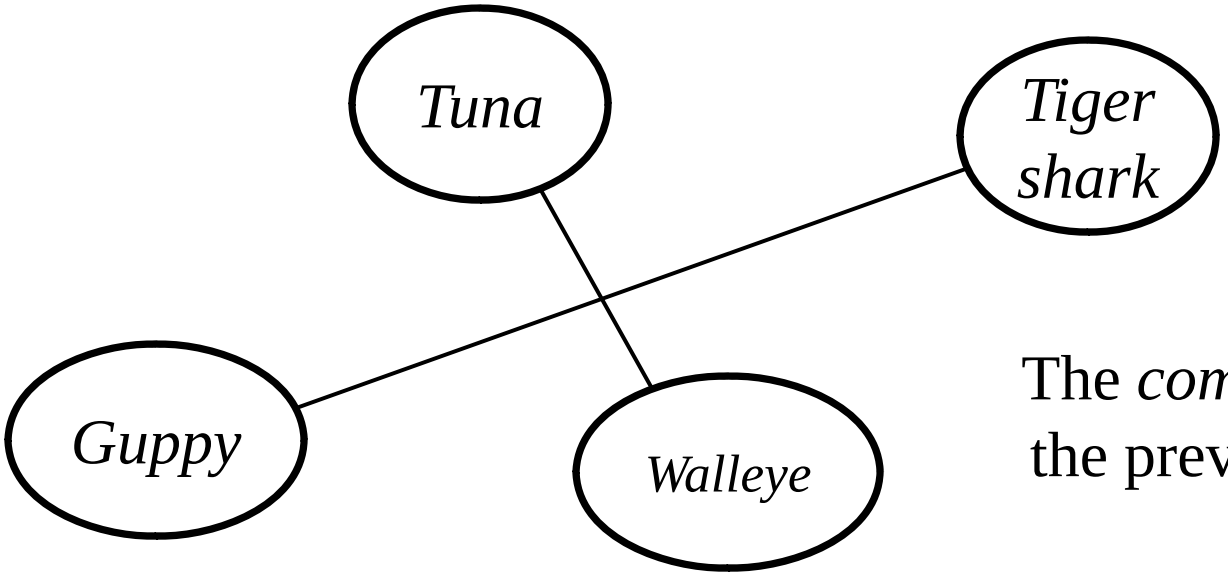
- Every fish is a vertex, each pair of conflicting fish is an edge



- A coloring of the graph gives you a valid assignment of fish to tanks

Fish and graphs: Solution 2

- Every fish is a vertex, each pair of *compatible* fish is an edge

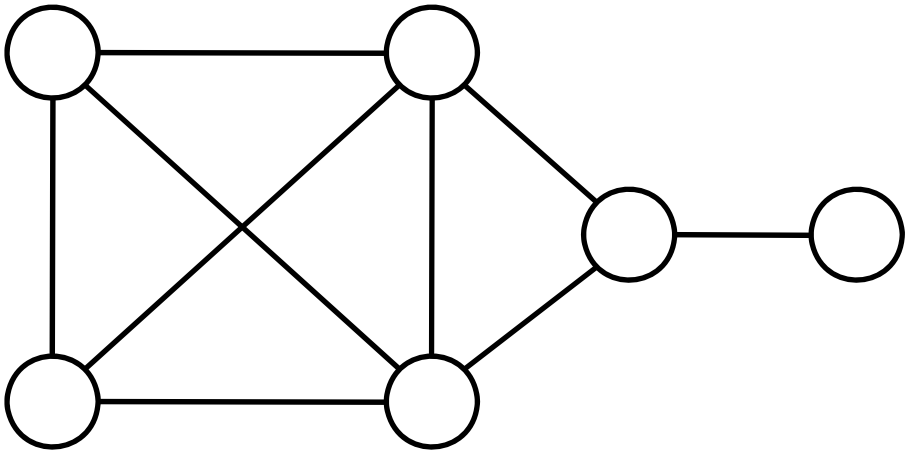


The *complement* of the previous graph

- A *clique* in the graph can be assigned to the same tank

Cliques

- A set of nodes in a graph that are all connected



Does it make sense?

- … to find the median at each stage of quicksort?

- Yes and no:
- For each subarray (of size n), we already do O(n) work to partition that subarray.
  - We can find the median in O(n) time (expected)
  - Finding the median doesn't increase the big-O running time from O(n log n)

- Usually slows things down in practice

Does it make sense?

- From Wikipedia:

Selection-based pivoting [edit]

A selection algorithm chooses the *k*th smallest of a list of numbers; this is an easier problem in general than sorting. One simple but effective selection algorithm works nearly in the same manner as quicksort, except that instead of making recursive calls on both sublists, it only makes a single tail-recursive call on the sublist which contains the desired element. This small change lowers the average complexity to linear or $\Theta(n)$ time, and makes it an in-place algorithm. A variation on this algorithm brings the worst-case time down to $\Theta(n)$ (see selection algorithm for more information).

Conversely, once we know a worst-case $\Theta(n)$ selection algorithm is available, we can use it to find the ideal pivot (the median) at every step of quicksort, producing a variant with worst-case $\Theta(n \log n)$ running time. In practical implementations, however, this variant is considerably slower on average.

O(n) vs. O(n^2)

- Algorithm X is O(n), algorithm Y is O(n^2)
- X will eventually catch up to Y
- … but we don't know when
- *n* = 1,000,000,000 might not be large enough
  - (see bubblesort)

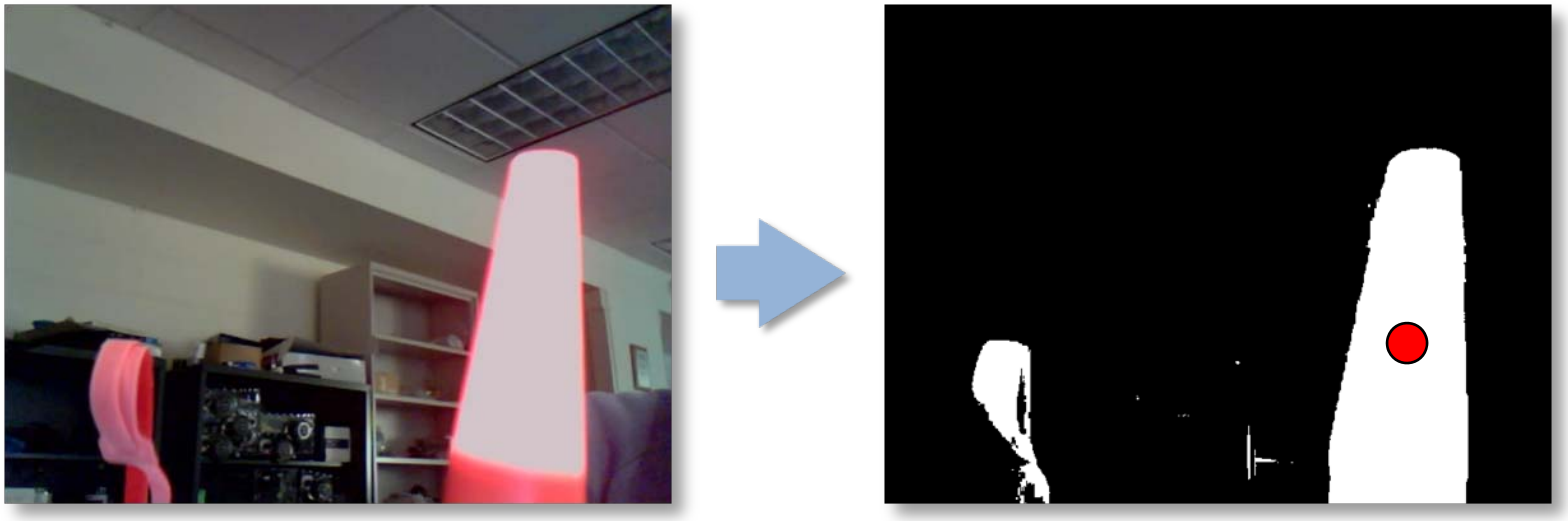
- … (but usually will be)

Introspection sort

```
function S = introsort(A)
n = length(A);
if n < 2000
    S = bubblesort(A);
else
    S= quicksort(A);
end
```

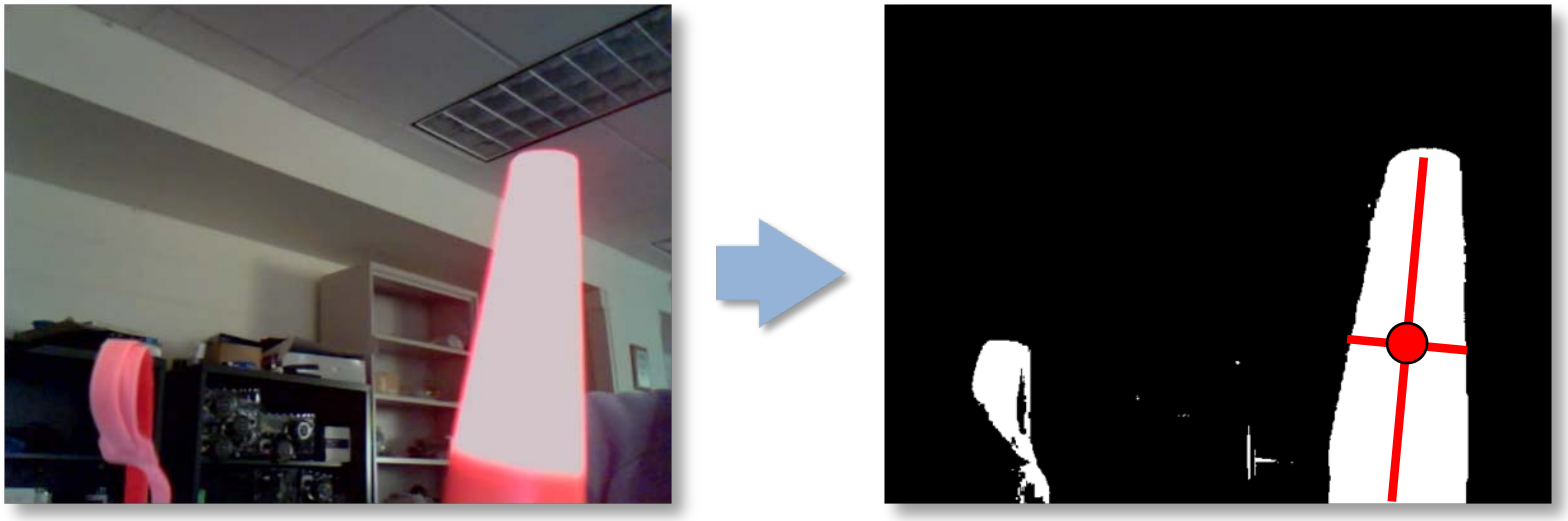
Finding the lightstick center

1. Threshold the image
2. Find blobs (connected components)
3. Find the largest blob **B**
4. Compute the median vector of **B**



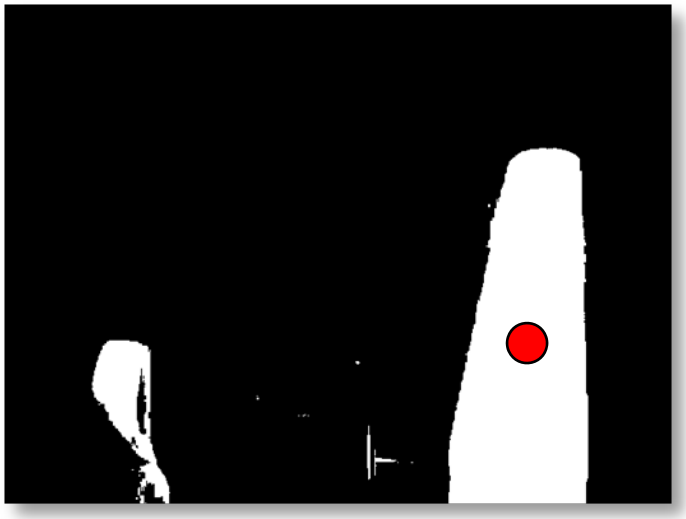
Finding the lightstick center

- But we also want to control the robot based on the orientation of the lightstick



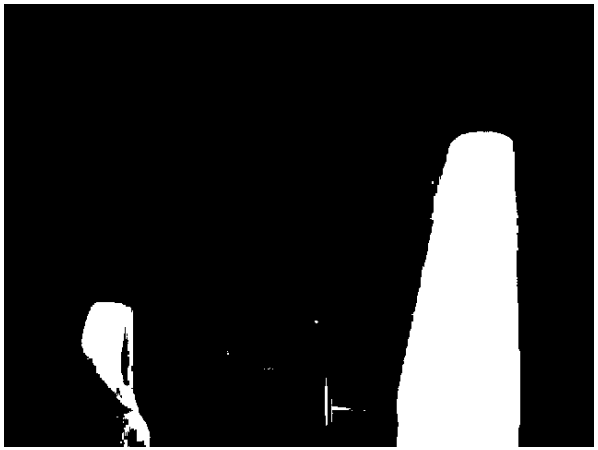
Finding the lightstick center

- So far we've only built functions that take a set of points and return another point
  - With one exception…
- How can we express the *shape* of the lightstick?

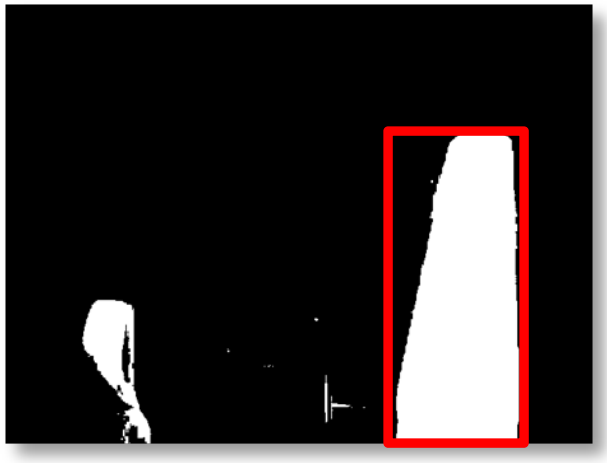


Finding the lightstick center

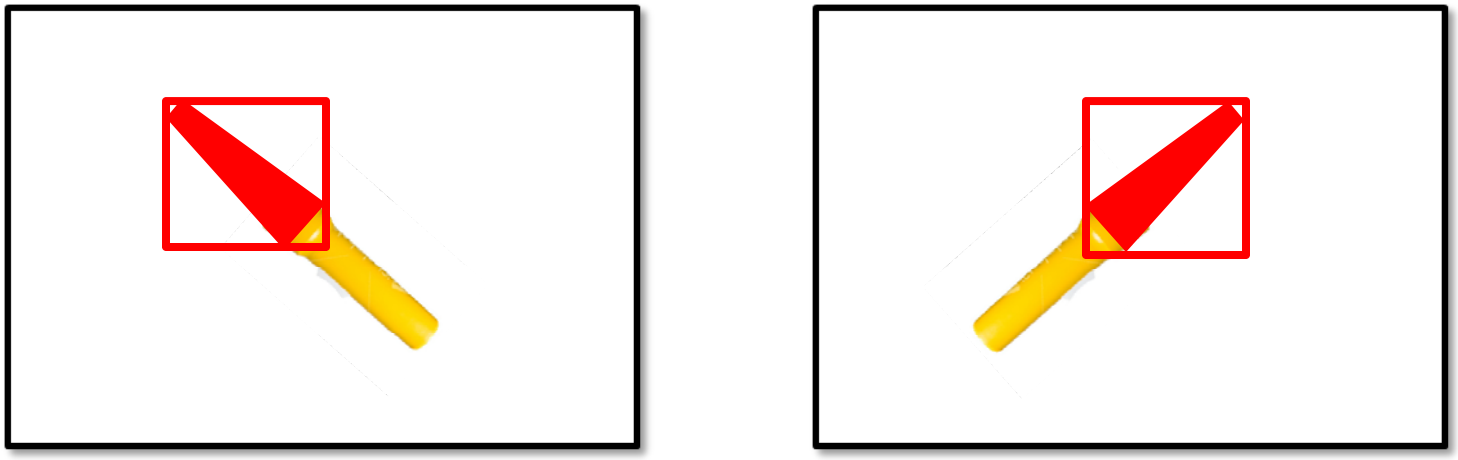
- We'll try to come up with a simple *polygon* to describe the lightstick



- Simplest polygon: the bounding box



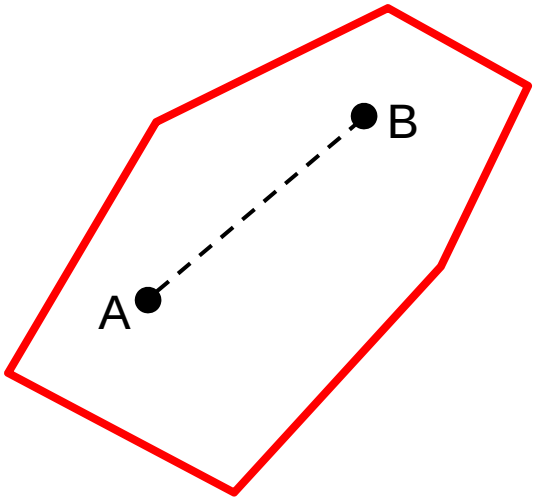
Bounding box



- Not as informative as we might like
- Let's come up with a polygon that fits better…

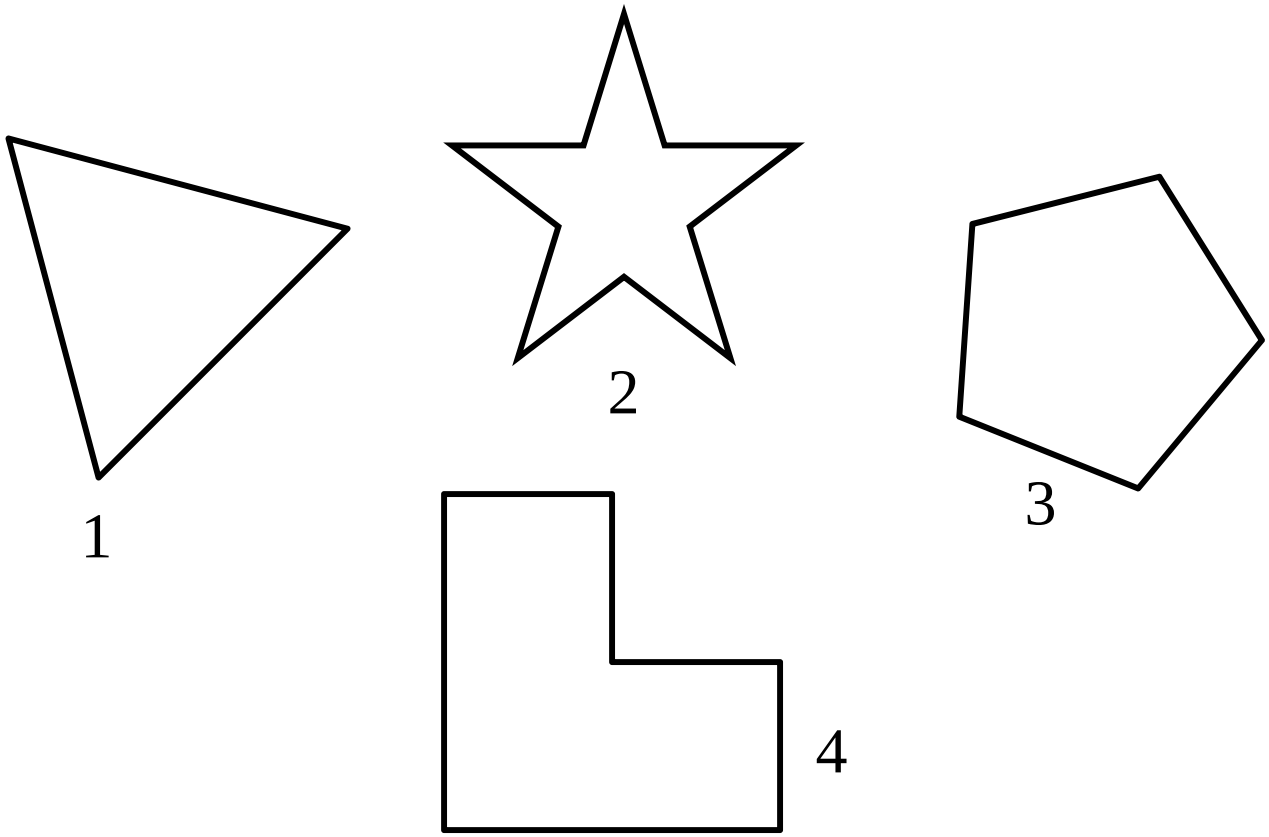
Detour: convex polygons

- A polygon *P* is **convex** if, for any two points *A*, *B* inside *P*, all points on a line connecting *A* and *B* are also inside *P*



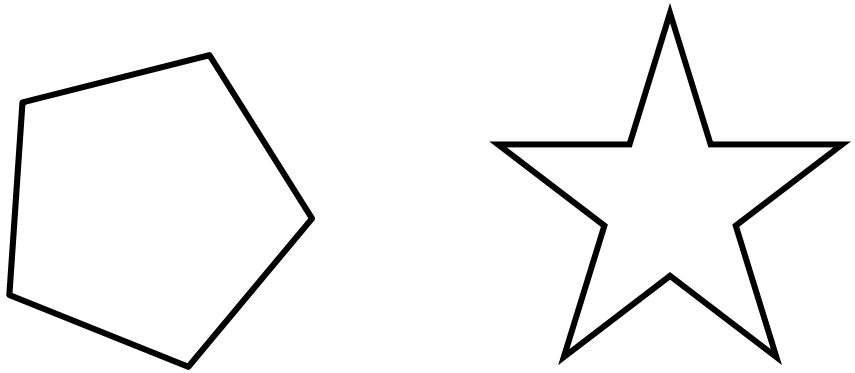
Convex polygons

- Which polygons are convex?



Testing convexity

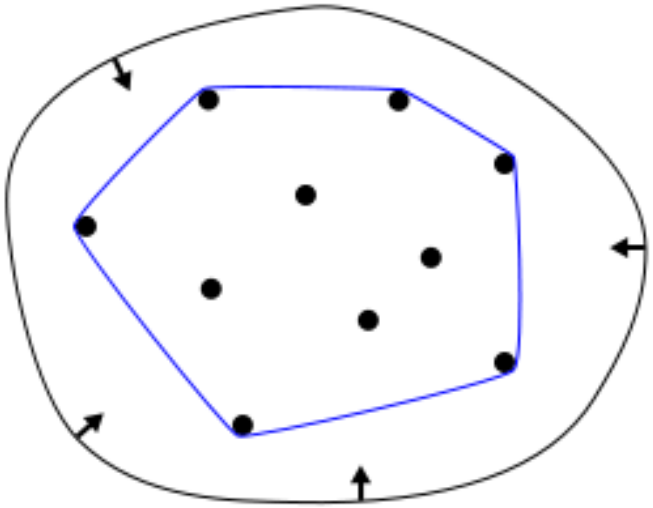
- How can we test if a polygon *P* is convex?



- Consider the smallest convex polygon containing *P*
  - Called the **CONVEX HULL**
  - What is the convex hull if *P* is convex?

Convex hull

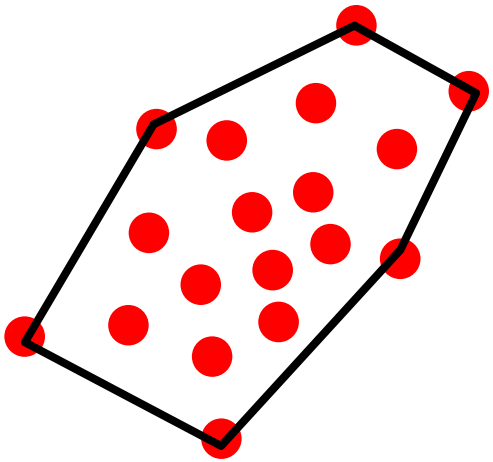
- Can also define for sets of 2D points: the smallest convex shape containing a set of 2D points



from http://en.wikipedia.org/wiki/Convex_hull

Convex hull of point sets

- We can use this to find a simple description of the lightstick's shape

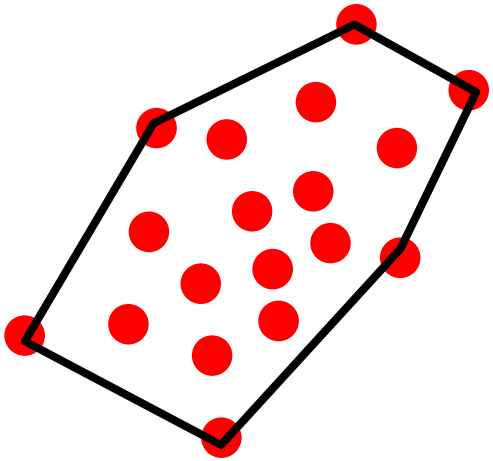


http://www.cs.princeton.edu/~ah/alg_anim/version1/ConvexHull.html

- How can we compute the convex hull?

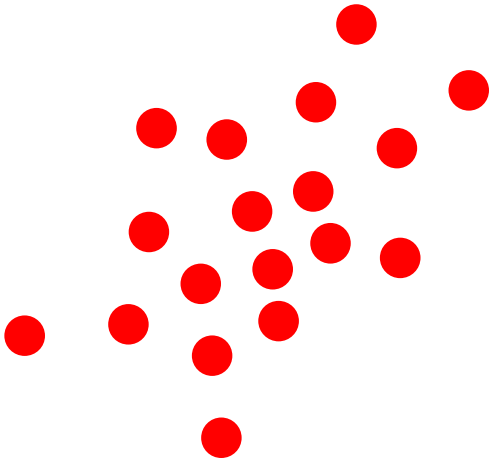
Computing convex hulls

- Idea: two points are an edge in the convex hull if _____



Computing convex hull

- Which two horizontal lines touch points on the convex hull?



- Which two vertical lines?
- → It is easy to identify at least four points that are part of the convex hull

Gift-wrapping algorithm

1. Start at lowest point
2. Rotate the line until we hit another point
   - All other points will lie on one side of this line
   - Look for the point that gives you the largest angle with the current line
3. Repeat
4. You're done when you get back to the starting point

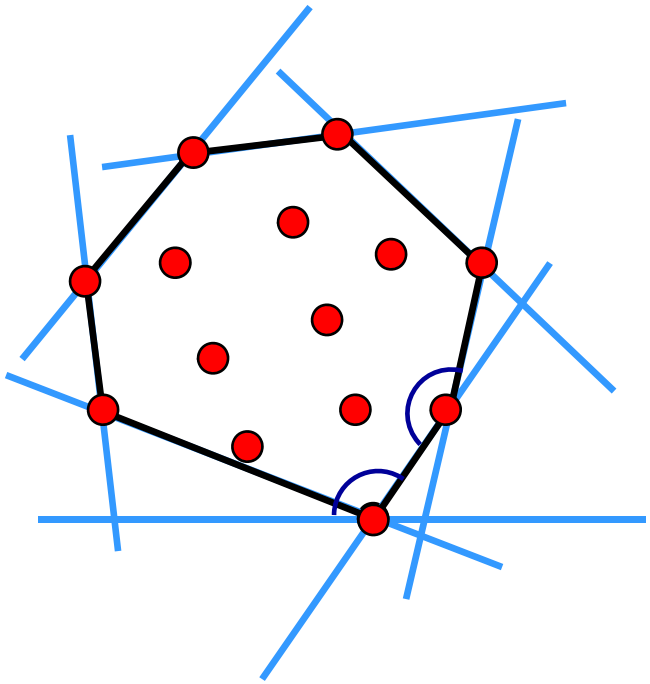
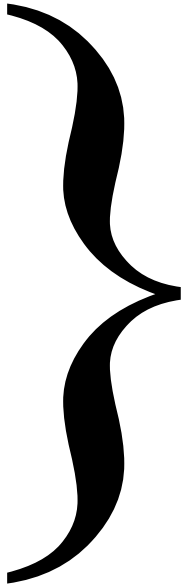


Figure credit: Craig Gotsman

The details…

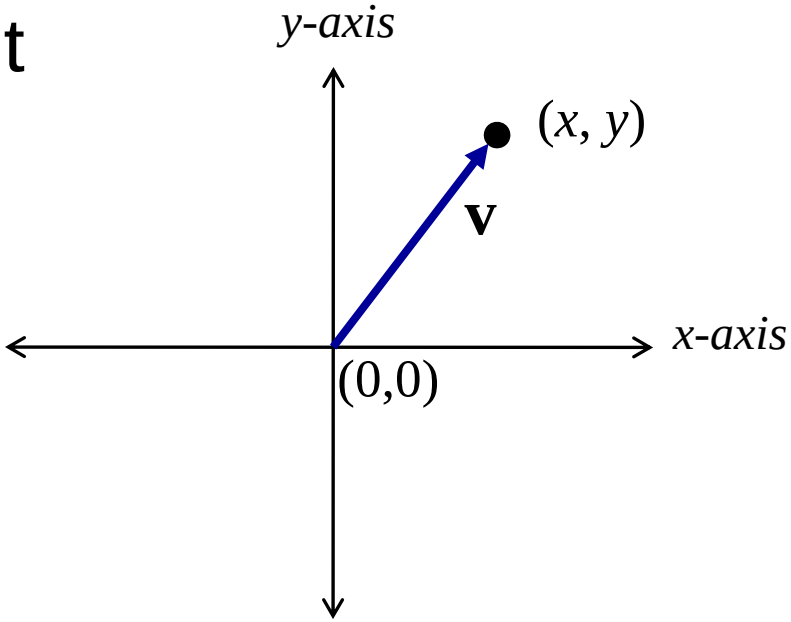
1. Start at lowest point
2. Rotate the line until we hit another point
   - All other points will lie on one side of this line
   - Look for the point that gives you the largest angle with the current line
3. Repeat
4. You're done when you get back to the starting point



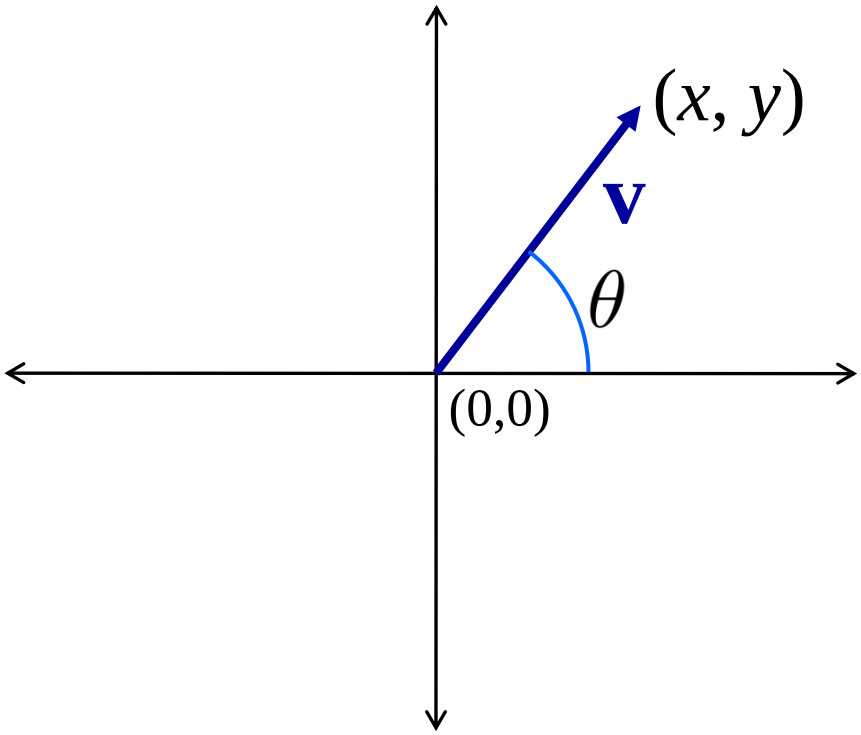
How do we implement this part?

Vectors

- To do this, let's talk about *2D vectors*
- A vector $\mathbf{v} = (x, y)$ is an "arrow" with a direction and length
- Similar to a 2D point



Vectors



length of **v** : ||**v**||

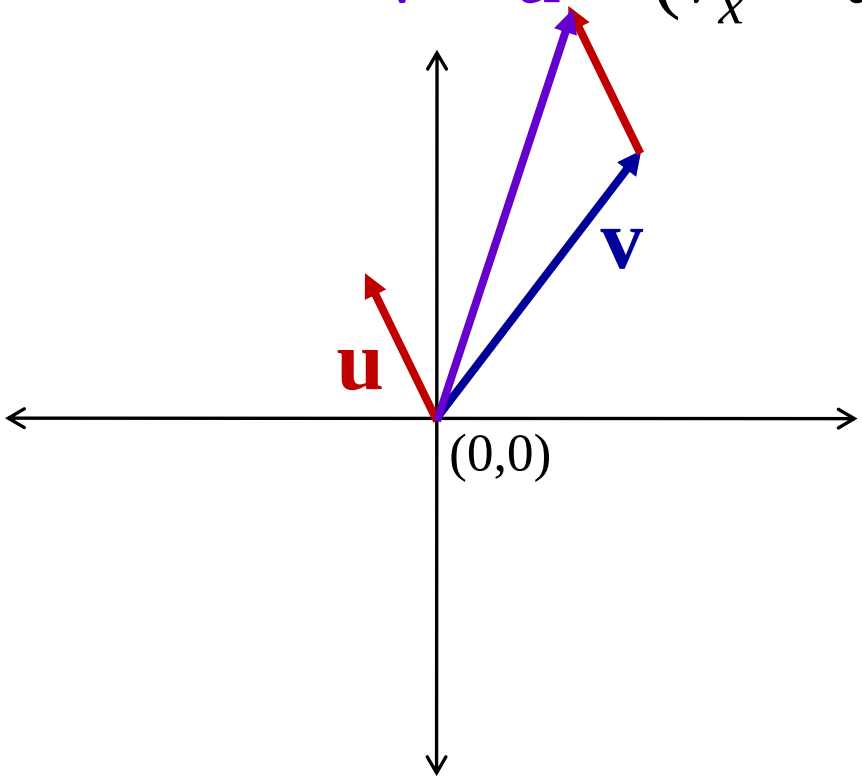
$$\|\mathbf{v}\| = \sqrt{x^2 + y^2}$$

direction of **v**:

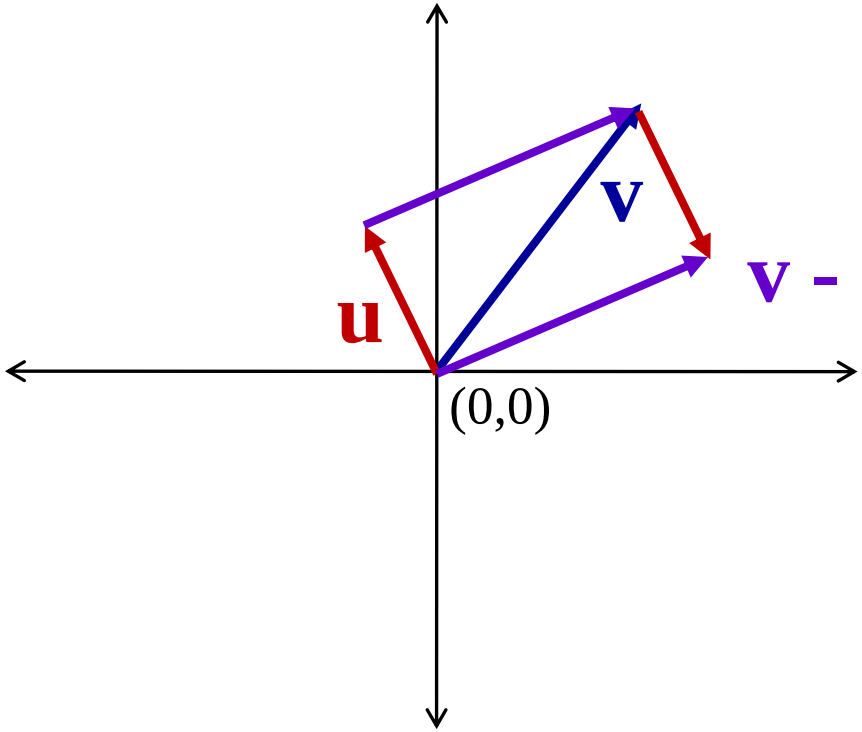
$$\theta = atan\left(\frac{y}{x}\right)$$

Vectors

$$\mathbf{v} + \mathbf{u} = (v_x + u_x, v_y + u_y)$$



Vectors



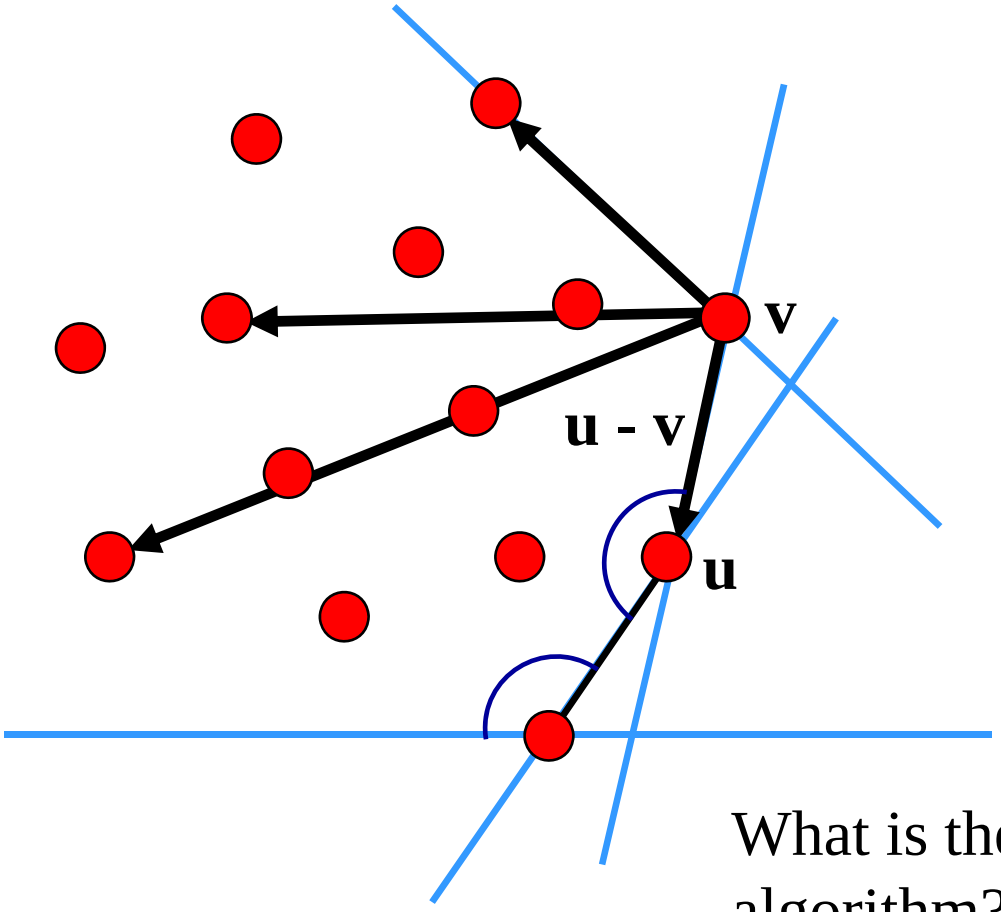
$\mathbf{v - u} = (v_x - u_x, v_y - u_y)$

Vectors

- Can also "multiply" two vectors:
  - Dot product: $\mathbf{v} \cdot \mathbf{u} = v_x u_x + v_y u_y$
  - Useful fact: $\mathbf{v} \cdot \mathbf{u} = ||\mathbf{v}||||\mathbf{u}|| \cos\theta$

$$\cos\theta = \frac{\mathbf{v} \cdot \mathbf{u}}{||\mathbf{v}||||\mathbf{u}||}$$

Back to the convex hull



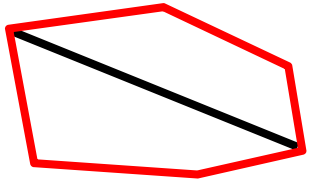
Which point is next?

Answer: the point **w** that maximizes the angle between **u – v** and **w – v**

What is the running time of this algorithm?

Lightstick orientation

- We have a convex shape
  - Now what?



- Want to find which way it's pointed

- For now, we'll find the two points that are furthest away from each other, and call that the "major axis"

Questions?