Finding Red Pixels – Part 2

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

- You should all set up your CSUG accounts
- Your card should now unlock Upson 319
Administrivia

- Assignment 1 posted, due Friday, 2/10 by 5pm
  - Look under “Assignments!”
  - You should have seen the post on Piazza
    - If not, let me know

- Quiz 1 on Thursday
Academic Integrity

- You may speak to others about the assignments, but may not take notes
- All code you write must be your own
Administrivia

- Office hours:
  - Prof. Snavely: Th 1:30 – 3pm Upson 4157
  - All other office hours are held in the lab, see staff page for times
Even more compact code

D = [ 10 30 40 106 123 8 49 58 112 145 16 53 ]

\[
\begin{align*}
D(1) & = D(1) + 20; \\
D(2) & = D(2) + 20; \\
D(3) & = D(3) + 20; \\
D(4) & = D(4) + 20; \\
D(5) & = D(5) + 20; \\
D(6) & = D(6) + 20; \\
D(7) & = D(7) + 20; \\
D(8) & = D(8) + 20; \\
D(9) & = D(9) + 20; \\
D(10) & = D(10) + 20; \\
D(11) & = D(11) + 20; \\
D(12) & = D(12) + 20;
\end{align*}
\]

\[
\text{for } i = 1:12 \\
\quad D(i) = D(i) + 20; \\
\text{end}
\]

\[
D = D + 20;
\]

- Special Matlab “Vectorized” code
- Usually much faster than loops
- **But please use for loops for assignment 1**
Why 256 intensity values?

8-bit intensity \((2^8 = 256)\)

5-bit intensity \((2^5 = 32)\)

5-bit intensity with noise
Why 256 intensity values?

Today’s (typical) displays:
256 * 256 * 256 = 16,777,216 colors
function [ nzeros ] = count_zeros(D)
% Counts the number of zeros in a matrix
nzeros = 0;
[nrows,ncols] = size(D);
for row = 1:nrows
    for col = 1:ncols
        if D(row,col) == 0
            nzeros = nzeros + 1;
        end
    end
end
Save in a file named count_zeros.m

count_zeros([1 3 4 0 2 0])
What about red pixels?

\[
\begin{align*}
\text{red}(1,1) &= 255, \quad \text{green}(1,1) = \text{blue}(1,1) = 0 \\
\text{red}(1,1) &= 255, \quad \text{green}(1,1) = 255, \quad \text{blue}(1,1) = 0
\end{align*}
\]
Assignment 1: come up with a thresholding function that returns 1 if a pixel is “reddish”, 0 otherwise
Finding the lightstick

- We’ve answered the question: is there a red light stick?

- But the robot needs to know *where* it is!
Finding the rightmost red pixel

- We can always process the red pixels as we find them:

```plaintext
right = 0;
for row = 1:nrows
    for col = 1:ncols
        if red(row,col) == 255
            right = max(right,col);
        end
    end
end
end
```
Finding the lightstick – Take 1

- Compute the **bounding box** of the red points
- The bounding box of a set of points is the smallest rectangle containing all the points
  - By “rectangle”, I really mean “rectangle aligned with the X,Y axes”
Finding the bounding box

- Each red pixel we find is basically a point
  - It has an X and Y coordinate
  - Column and row
    - Note that Matlab reverses the order
What does this tell us?

- Bounding box gives us some information about the lightstick
  - Midpoint → rough location
  - Aspect ratio → rough orientation
  (aspect ratio = ratio of width to height)

Aspect ratio: 2.05/1.08 = 1.9
Computing a bounding box

- Two related questions:
  - Is this a good idea? Will it tell us **reliably** where the light stick is located?
  - Can we compute it quickly?
Computing a bounding box

- Lots of CS involves trying to find something that is both useful and efficient
  - To do this well, you need a lot of clever ways to efficiently compute things (i.e., algorithms)
  - We’re going to learn a lot of these in CS1114
Beyond the bounding box

- Computing a bounding box isn’t hard
  - Hint: the right edge is computed by the code we showed a few slides ago
  - You’ll write this and play with it in A2

- Does it work?
Finding the lightstick – Take 2

- How can we make the algorithm more robust?
  - New idea: compute the centroid

- Centroid:
  
  (average x-coordinate, average y-coordinate)

  - If the points are scattered uniformly, this is the same as the midpoint of the bounding box
  - Average is sometimes called the mean
  - Centroid = center of mass
Computing the centroid?

- We could do everything we want by simply iterating over the image as before
  - Testing each pixel to see if it is red, then doing something to it
- It’s often easier to iterate over just the red pixels
- To do this, we will use the Matlab function called **find**
The `find` function

Your thresholding function

$$[X, Y] = \text{find}(\text{thresh});$$

$X$: x-coords of nonzero points

$Y$: y-coords of nonzero points
Using find on images

- We can get the x- and y-coordinates of every red pixel using **find**
  - Now all we need to do is to compute the average of these numbers
  - We will leave this as a homework exercise
    - You might have done this in high school
Q: How well does this work?

- A: Still not that well
  - One “bad” red point can mess up the mean

- This is a well-known problem
  - What is the average weight of the people in this kindergarten class photo?
How well does this work?
How can we do better?

- What is the average weight of the people in this kindergarten class photo?

12 kids, avg. weight = 40 lbs
1 Arnold, weight = 236 lbs

Mean: \((12 \times 40 + 236) / 13 = 55 \text{ lbs}\)
How can we do better?

- Idea: remove maximum value, compute average of the rest

12 kids, avg. weight = 40 lbs

Mean: \( \frac{12 \times 40 + 236}{13} = 55 \text{ lbs} \)

1 Arnold, weight = 236 lbs

Mean: 40 lbs

Mean: \( \frac{12 \times 40 + 236}{13} = 55 \text{ lbs} \)
How can we avoid this problem?

- Consider a simple variant of the mean called the “trimmed mean”
  - Simply ignore the largest 5% and the smallest 5% of the values
  - Q: How do we find the largest 5% of the values?

D.E. Knuth, *The Art of Computer Programming*
Chapter 5, pages 1 – 391
Easy to find the maximum element in an array

A = [11 18 63 15 22 39 14 503 20];
m = -1;  \% Why -1?
for i = 1:length(A)
    if (A(i) > m)
        m = A(i);
    end
end
\% At the end of this loop, m contains the
\% biggest element of m (in this case, 503)
How to get top 5%?

- First, we need to know how many cells we’re dealing with
  - Let’s say `length(array)` is 100
    → want to remove top 5

- How do we remove the biggest 5 numbers from an array?
Removing the top 5% -- Take 1

% A is a vector of length 100
for i = 1:5
  % 1. Find the maximum element of A
  % 2. Remove it
end
How good is this algorithm?

% A is a vector of length 100
for i = 1:5
  % 1. Find the maximum element of A
  % 2. Remove it
end

- Is it correct?
- Is it fast?
- Is it the fastest way?
How do we define fast?

- It’s fast when $\text{length}(A) = 20$
- We can make it faster by upgrading our machine

- So why do we care how fast it is?
- What if $\text{length}(A) = 6,706,993,152$?
How do we define fast?

- We want to think about this issue in a way that doesn’t depend on either:
  A. Getting really lucky input
  B. Happening to have really fast hardware
How fast is our algorithm?

- An elegant answer exists
- You will learn it in later CS courses
  - But I’m going to steal their thunder and explain the basic idea to you here
  - It’s called “big-O notation”

- Two basic principles:
  - Think about the average / worst case
    - Don’t depend on luck
  - Think in a hardware-independent way
    - Don’t depend on Intel!
For next time

- Attend section tomorrow in the lab
- Reminder: Quiz on Thursday, beginning of class