Preprocessing to accelerate 1-NN

Prof. Ramin Zabih

http://cs100r.cs.cornell.edu

Cornell University
Computer Science

Administrivia

- A6: rotation, overhead robot tracking
- Final project proposals due Friday
  - Not graded, but required
- Last quiz: Thursday 11/15
- Prelim 3: Thursday 11/29 (last lecture)

Final projects

- Due Friday, Dec 7, 4-5PM
- Goal: "Do something cool"
- Grading will come primarily from effort
  - Something interesting that works occasionally is fine
- Example projects:
  - Robot "Boston driver"
  - Robot tracking another robot
  - Dancing Aibo's

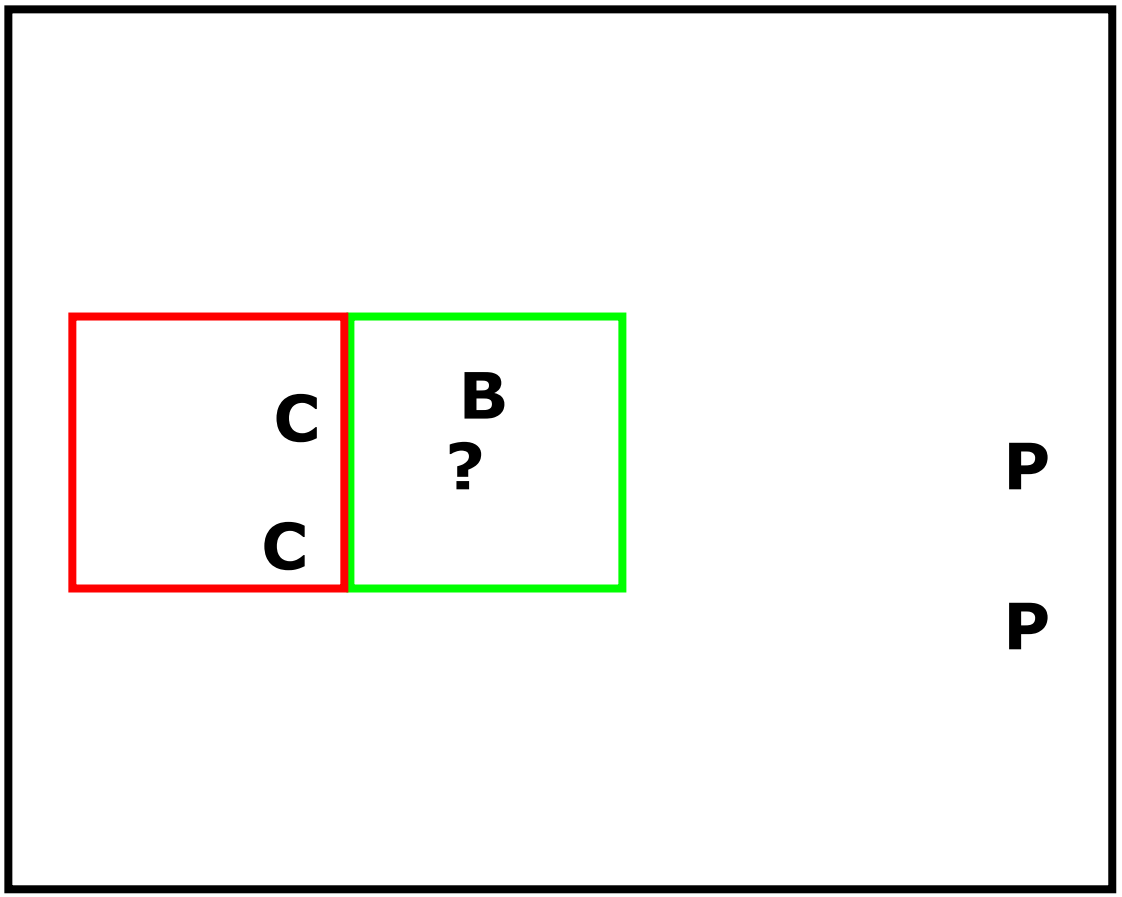
Recall: 1-NN algorithm

- Inputs: labeled points in a space, and a query point whose label you want to know
- Specification: the answer is the label of the nearest point
  - Any care to guess what k-NN does?
- Today: algorithms to speed this up
  - Spend some time with the labeled points in advance, to make queries fast
  - Amortize the cost of pre-processing

How to pre-process?

- There are many ways to do this
  - We want to not bother to compute the distance to model pixels that are very far away
- Simple, initial approach
  - We'll draw our examples in 2D colorspace
  - Suppose we break our space into big squares and keep track of what model pixels are in each square.
    - Can this help us?

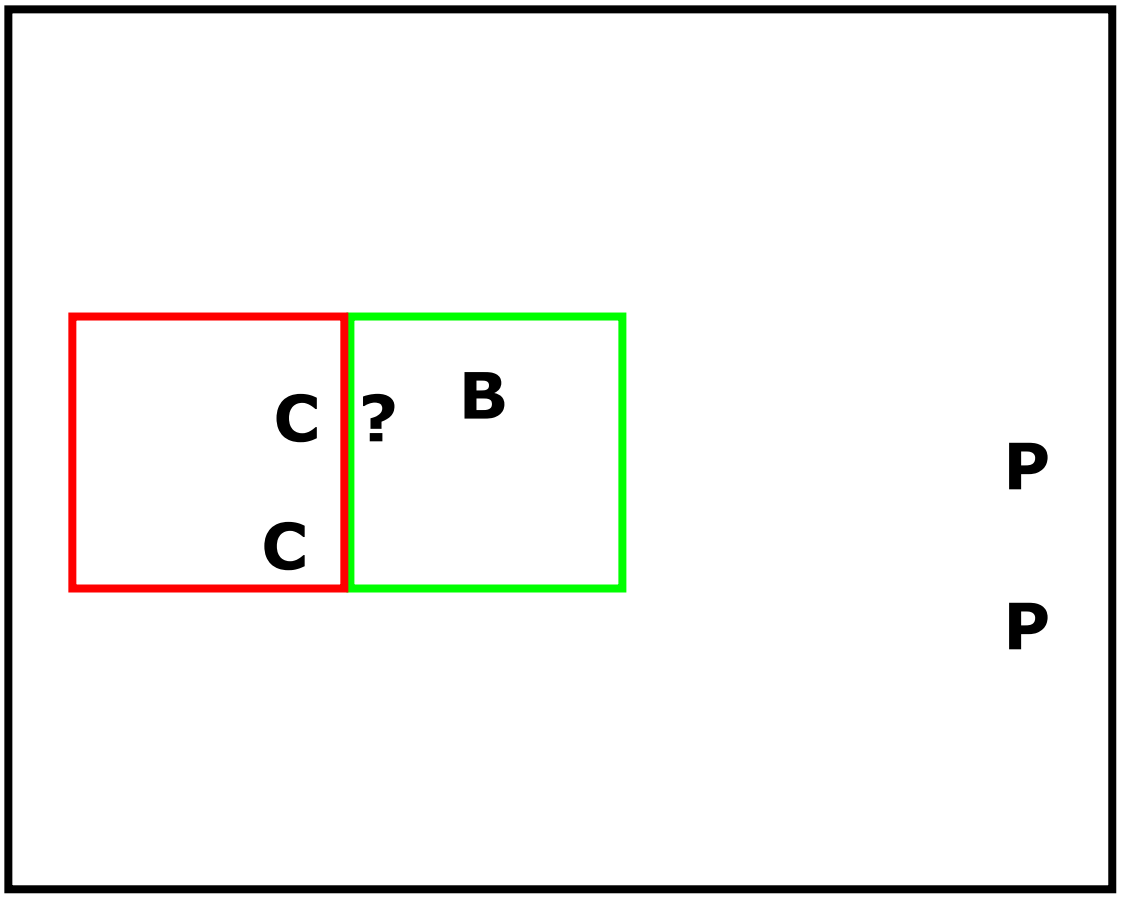
What can we eliminate?



What we need

- For every square, we need to know what model pixels are in it
  - Note that we are assuming the model pixels are reasonably spread out (not intermixed)
- There are several variants
  - The easiest one is to store a single bit to indicate the presence of any **C**, **B** or **P** pixel
- Suppose that the query falls into a box that contains only **C** model pixels
  - Is the nearest model pixel a **C**?

What can we soundly eliminate?



Strategy

- For each block, we record the presence of any model pixel from **C**, **B** or **P**
  - We'll call a block with exactly one type of model pixel "pure"
- Suppose that P falls into a pure **C** block
  - And the 8 neighboring blocks are also pure **C**
    - Why isn't 4 enough?
  - We'll call this an 8-pure block
  - Then the class of P is **C**!

Questions about our strategy

- What is the effect of block size?
  - Big blocks tend not to be pure
  - Little blocks tend to be empty
    - But, we could look for the nearest pure block
    - In the limit, this is something we've seen:
      - Bad idea #2
- For a 8-pure block, we know the class for any pixel in this block, without search
  - You can actually do this for **any pixel**, if you are willing to spend time pre-processing

Implementing our strategy

- We are given a pixel P in RGB space
  - We want to know if its block is pure
  - If we can tell this, we can also figure out if it is 8-pure (how?)
- We could create a version of colorspace where each color contains a block number
  - But there is a much better solution if the block size is a power of 2 (such as 64 by 64)
  - We can tell the block number directly from the color of P (i.e., coordinates in colorspace)

Binary numbers

- Binary numbers make certain operations fast due to their representation
  - Example: is this number odd or even?
- If computers processed in base ten, the following operations would be easy:
  - Given a number, how many complete thousands does it contain?
    - 107,259 contains 107 complete thousands
- A computer can easily tell how many complete 64's a number contains

Handling non-pure blocks

- If a block isn't pure, it still contains some useful information
  - Assuming it isn't empty
  - It's not hard to find a small number of blocks that contain the right answer
    - I.e., the model pixel that P is closest to
- Can we pre-process colorspace so that we can easily look at all the model pixels in a given block
  - Without examining every pixel in the block?
  - We are assuming model pixels are sparse

Smarter pre-processing

- Consider a model pixel, call it P
  - There is a certain region, call it reg(P), where P is the nearest labeled point
  - If we know reg(P), for every P, then we can figure out the closest point for every query
- We could compute reg(P) by asking, for every query, what the closest point is
  - But this wouldn't be very smart
    - Especially if the space is continuous…

Voronoi diagrams

- Divide our space into regions, where each region reg(P) consists of the points closest to a labeled point P
  - This is a Voronoi diagram
  - Long history (Descartes, 1644)