CME 213

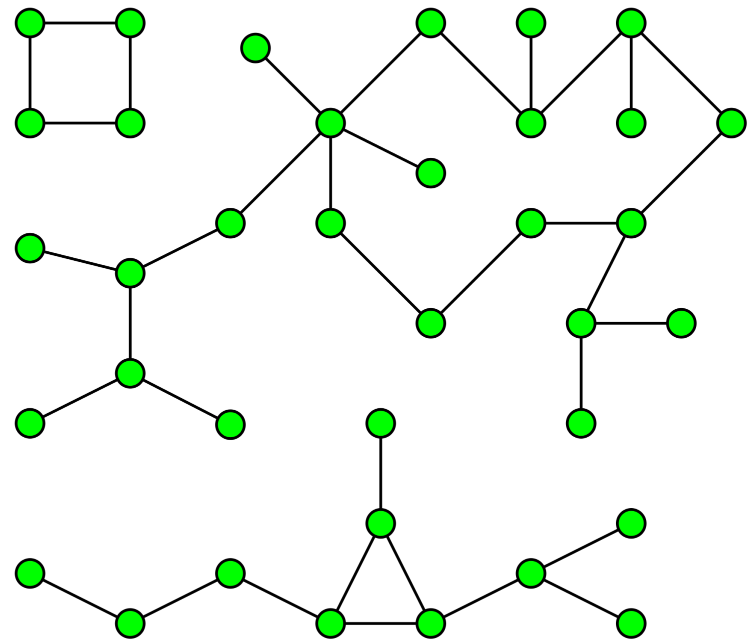
Spring 2012-2013

Eric Darve

Final project

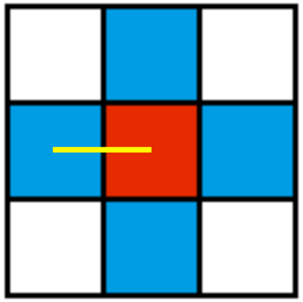
Connected Component problem

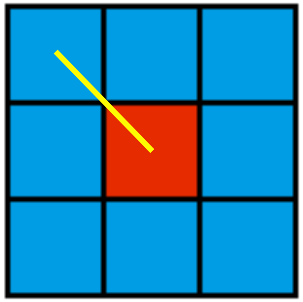
- Project is about computing connected components in a graph.
- General definition:
  - Graph: vertices and edges connecting vertices
  - Connected component:
    - two vertices that are connected by an edge belong to the same component
    - if (a,b) is in the same component then there must be a path going from a to b, following edges of the graph.



Images

- We are going to consider the problem of connected components in the context of images, more specifically black and white images.
- Pixels in the image are vertices of the graph.
- Edges exist only between neighboring pixels:

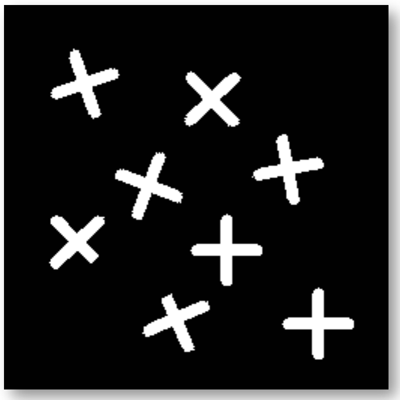




4 connected   8 connected

- Two pixels are connected if:
  - They are neighbors (see definition above)
  - Their color is the same

EXAMPLE



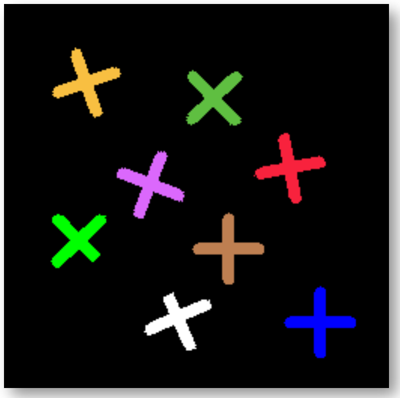
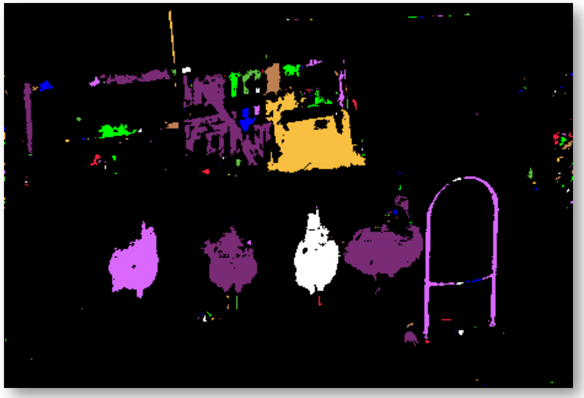


Image that defines the graph



Components are colored

COMPONENT LABELING PROBLEM

Objective of the final project: write a CUDA code to

- Identify all the connected components.
- Assign a unique label to each connected component.
- Assign to each pixel its unique connected component label.

- We give you several references in the handout that you can consult for this project.
- We can read any material you want! Our list is just a suggestion to get you started.

GTC On Demand

- Connected Components Revisited on Kepler, Gernot Ziegler
- GTC 2013 - ID S3193
- Video: 9:31

GRADING

Rules

For the final project, the rules of the Honor Code are defined as follows:

1. You cannot copy any code from someone else (except the starter code given by the instructors). To be safe, do not look at other students' code and do not show yours.
2. You can read any document you like about this problem. In fact being able to read, understand and apply techniques and algorithms from the literature to solve this problem is part of the exercise. Make sure you reference all your sources in your paper.

SCHEDULE

Date	Grade percent	Item
June 7	20%	One-page report
June 12	80%	4-page report or oral presentation

- The deadlines are strict. There is no grace period for the final project.
- Coursework has two online assignments, one for each deadline.

ONE-PAGE REPORT

- At the end you will give either a 4-page report or an oral presentation.
- The one-page report will be used to determine who gives the oral presentation vs 4-page report.
- Grading of the one-page report:
  - 17%: provide a code that does the calculation correctly
  - 3%: describe an approach to optimize your code
  - Total: 20%

FINAL REPORT/ORAL PRESENTATION

- 55%: un-optimized CUDA code that runs correctly. Code should not use shared memory. Write tests for your code.
- 17%: optimized CUDA code that runs correctly. An example of expected optimization is to use shared memory. This is not the only optimization that is possible. Write tests for your code.
- Discussion:
  - 2%: quality of report: correctness, clarity
  - 2%: discussion of performance: bottleneck, usage efficiency of hardware resources
  - 2%: analysis of data: plots and benchmarks
  - 2%: discussion of literature

Overall grading curve:

- 72%: basic CUDA code
- 92%: optimized CUDA code
- 100%: relevant and meaningful discussion of optimization strategies

Advice

- Algorithms for this problem can become very complicated.
- Don’t overdo it! We are not expecting an *optimal* code.
- You can get an A+ with only the basic optimization strategies implemented.
- As a reference, we explicitly refer to Kernel A and B in Hawick et al (see paper on coursework) as unoptimized and optimized implementations. However you may implement other methods. This is simply to give you an idea of the level of “complexity” expected in your algorithms.
- Remember the advice from David Goodwin. To optimize your code:
  1. Identify the hardware resource whose usage is maximized. This means that resource is the performance bottleneck.
  2. Change your algorithm such that that resource is no longer the limiting factor.
  3. Go to 1 and iterate the process.
- Beware of general absolutist rules, such as: best performance requires full SM occupancy, I need to use the largest possible number of threads, 192 is the optimal number of threads per block.

QUESTIONS?

Stanford University