Goal of the Assignment

- Calculate Avogadro’s number
  - Using Einstein’s equations
  - Using fluorescent imaging

- Input data
  - Sequence of images
  - Each image is a rectangle of pixels
  - Each pixel is either light or dark

- Output
  - Estimate of Avogadro’s number
Assignment: Four Programs

- Blob data type
  - Maximal set of connected light pixels

- BlobFinder data type
  - Find all blobs in a JPEG image
  - List all the big blobs (aka beads)

- beadtracker
  - Track beads from one image to the next

- avogadro
  - Data analysis to estimate Avogadro’s number from the motion of beads
Atomic Theory Overview

- Brownian Motion
  - Random collision of molecules
  - Displacement over time fits a Gaussian distribution
Atomic Theory Overview

- **Avogadro’s Number**
  - Number of atoms needed to equal substance’s atomic mass in grams
    - $N_A$ atoms of Carbon-$12 = 12$ grams
  - Can calculate from Brownian Motion
    - Variance of Gaussian distribution is a function of resistance in water, number of molecules
blob.h, blob.cpp

- API for representing particles (blobs) in water
  - Blob()
  - void add(int i, int j)
  - int mass() // number of pixels
  - double distance_to(Blob b) // from center (average)
  - string to_string()

- Only need *three* values to efficiently store
  - Do *not* store the positions of every pixel in the blob

Center of mass, and # of pixels
blob.h, blob.cpp Challenges

- Write your own thorough tests (test_blob.cpp)
- Add header guards to blob.h to avoid linker errors:
  ```
  #ifndef BLOB_H
  #define BLOB_H
  ...
  #endif
  ```
- Use ostringstream to implement to_string()

Center of mass, and # of pixels
blobfinder.h, blobfinder.cpp

- Locate all blobs in a given image
  - And identify large blobs (called beads)

- API
  - BlobFinder(Picture picture, double threshold)
    - Calculate luminance
      - Include pixels with a luminance >= threshold
      - Use the luminance formula in assignment
    - Find blobs with DFS
      - The hard part, next slide…
    - This constructor should build a vector of all Blobs in picture
  - vector<Blob> get_beads(int minSize)
    - Returns all “beads:” Blobs with at least minSize pixels
    - vector must be of size equal to number of beads
BlobFinder - Depth First Search

- Use boolean 2D vector to mark visited
- Traverse image pixel by pixel
  - Dark pixel
    - Mark as visited, continue
  - Light pixel
    - Create new blob, call DFS
- DFS algorithm
  - Base case: simply return if
    - Pixel out-of-bounds
    - Pixel has been visited
    - Pixel is dark (and mark as visited)
  - Add pixel to current blob, mark as visited
  - Recursively visit up, down, left, and right neighbors
BlobFinder - Depth First Search

- Use boolean 2D vector to mark visited
- Traverse image pixel by pixel
  - Dark pixel
    - Mark as visited, continue
  - Light pixel
    - Create new blob, call DFS
- DFS algorithm
  - Base case: simply return if
    - Pixel out-of-bounds
    - Pixel has been visited
    - Pixel is dark (and mark as visited)
  - Add pixel to current blob, mark as visited
  - Recursively visit up, down, left, and right neighbors
BlobFinder - Depth First Search

- Use boolean 2D vector to mark visited
- Traverse image pixel by pixel
  - Dark pixel
    - Mark as visited, continue
  - Light pixel
    - Create new blob, call DFS
- DFS algorithm
  - Base case: simply return if
    - Pixel out-of-bounds
    - Pixel has been visited
    - Pixel is dark (and mark as visited)
  - Add pixel to current blob, mark as visited
  - Recursively visit up, down, left, and right neighbors
BlobFinder - Depth First Search

- Use boolean 2D vector to mark visited
- Traverse image pixel by pixel
  - Dark pixel
    - Mark as visited, continue
  - Light pixel
    - Create new blob, call DFS
- DFS algorithm
  - Base case: simply return if
    - Pixel out-of-bounds
    - Pixel has been visited
    - Pixel is dark (and mark as visited)
  - Add pixel to current blob, mark as visited
  - Recursively visit up, down, left, and right neighbors
BlobFinder - Depth First Search

- Use boolean 2D vector to mark visited
- Traverse image pixel by pixel
  - Dark pixel
    - Mark as visited, continue
  - Light pixel
    - Create new blob, call DFS
- DFS algorithm
  - Base case: simply return if
    - Pixel out-of-bounds
    - Pixel has been visited
    - Pixel is dark (and mark as visited)
  - Add pixel to current blob, mark as visited
  - Recursively visit up, down, left, and right neighbors
Use boolean 2D vector to mark visited

Traverse image pixel by pixel
- Dark pixel
  - Mark as visited, continue
- Light pixel
  - Create new blob, call DFS

DFS algorithm
- Base case: simply return if
  - Pixel out-of-bounds
  - Pixel has been visited
  - Pixel is dark (and mark as visited)
- Add pixel to current blob, mark as visited
- Recursively visit up, down, left, and right neighbors
BlobFinder Challenges

- Data members?
  - vector of Blobs for sure

- Variables to facilitate DFS?
  - 2D vector of bool for “visited”
  - Either store as data member, or declare as local to constructor and pass by pointer/reference to recursive function
  - Other variables involved in recursive DFS should be treated similarly

- Don’t try to mimic BFS too closely
  - No Location, Itinerary, predecessor necessary
Track beads between successive images

Single main function

- Take in a series of images
- Output distance traversed by all beads for each time-step
  - For each bead found at time $t+1$, find closest bead at time $t$ and calculate distance
    - Not the other way around!
    - Don’t include if distance $> 25$ pixels (new bead)
beadtracker Challenges

- Reading multiple input files
  - `./beadtracker 25 180.0 25.0 run_1/*.jpg`
  - Expands files in alphabetical order
  - End up as argv[4], argv[5], ...

- Avoiding memory issues
  - Don’t use `new` or `delete` at all
  - Avoid opening > 2 Pictures at once

- Recompiling
  - Recompile if Blob or BlobFinder change
avogadro.cpp

- Analyze Brownian motion of all calculated displacements
  - Lots of crazy formulas, all given, pretty straightforward
  - Be careful about units in the math, convert pixels to meters, etc.
- Can test without the other parts working
  - We provide sample input files
  - Can work on it while waiting for help
Conclusion: Final Tips

- Avoiding subtle bugs in blobfinder
  - Blobs or vectors passed into/out of functions may get copied (pass-by-value)
  - Use data member, pointer or reference

- Common errors
  - Segfault from stack overflow: no base case
  - Out of bounds: use `.at()` and valgrind
  - Use test_blobfinder as soon as possible

- Look at checklist Q&A