Visualizing Blood Flow in Perfusion Angiography

Kang (Frank) Chen [UID: 204256656], Arvin Nguyen [UID: 304300135]
CS 188: Medical Imaging — University of California, Los Angeles

Abstract

Visualization of blood flow is essential for the diagnosis and treatment evaluation of cerebrovascular diseases. Perfusion angiography is a methodological framework for the quantitative analysis and visualization of blood flow parameters from DSA images [7]. However, animated flow visualization in angiography remains to be an unexplored field. In this work, we propose our method to visualize angiography using dynamic vector fields generated from a variant of the breadth-first search (BFS) algorithm. In the future, we also plan to release a web-based application that abstracts the implementation and provides a simplistic interface for researchers to drag and drop angiography files to be visualized.

1 Introduction

The primary contribution of our work include:

1. Isolate the blood vessels in our angiography videos through frame-by-frame *skeletonization*.
2. Compute path of blood flow frame-by-frame via a variant of BFS algorithm
3. Applying a JavaScript-based animation rendering that follows the vector field of our angiography blood flow.

It is important to keep in mind that visualizing blood flow is an unsupervised learning problem, as there are no labels in the dicom frames. It is our responsibility to find the correction direction of blood flow and visualize it through vector fields.

2 Methods

2.1 Isolate blood vessels

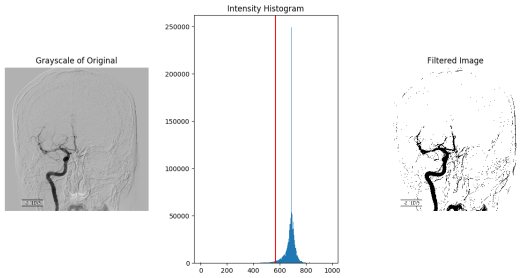
The initial challenge is to isolate the blood vessels from the input image. This poses a challenge because: the input image becomes noisy when blood vessels spread out from the initial artery, other noises persist in frame, such as watermark and skull outlines.

We approached using a 3-step image processing algorithm, illustrated in Figure 4.

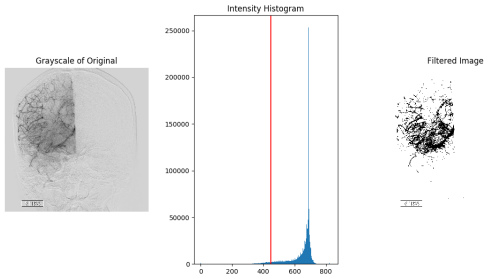
1. Perform a Percentile Intensity Threshold on the input image. A generalized version of adaptive thresholding [2], percentile thresholding attempts to find a percentage in terms of the current image frame's pixel intensity. As shown in Figure 1, this is more robuts than determining a hard-coded threshold value, because every frame's pixel intensity spectrum is different.

2. Invert the threshold image, then apply a median filter with a 18×18 mask to remove small noise [3].

3. Skeletonize the filtered image [6].



(a) Percentile intensity threshold on an initial frame



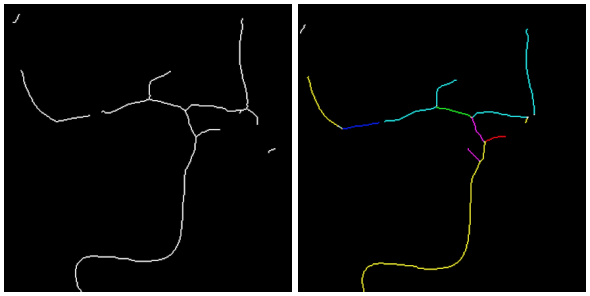
(b) The robustness of the percentile intensity threshold is shown when blood vessels spread out.

Figure 1: The top frame indicates that the artery is the primary focus for thresholding, while the bottom frame indicates that it is more important for us to separate the fine vessels. A hard-coded threshold value will not work here, so percentile thresholding (the red line in the middle graph) allows us to constantly find the 80 percentile.

2.2 Blood Flow Direction Algorithm

After obtaining the skeletonization, the next step is to generate a dictionary of segment objects, each containing an ordered path of coordinates. This will be passed to the generative vector field function, which can prepare the coordinates for visualization in our Javascript rendering library. Algorithm 1 describes the three primary procedures in determining *Flow Direction*.

Our flow direction algorithm uses BFS to traverse the path [5], with a starting heuristic as follows: On inital frame, we assume that blood flow begins with the artery (starting at the bottom of the frame). Therefore, we scan the image from the bottom-left corner until we hit our initial white pixel, which becomes our **root**. From there, we traverse until we reach a junction, at which point we add possible junction paths into a queue, and run BFS recursively until we have traversed the entire connected path. We save this into a dictionary of segments; this allows us to visualize **junction splits**.



(a) (b)

Figure 2: (a): Original skeletonization of frame 7; (b): Skeletonization of frame 7 with segments color-labeled (generated from Algorithm 1).

2.3 Vector Field Animation

We can learn vector fields by first identifying the direction of flow in the blood vessels via perfusion angiography [7]. Similar methods to enhance angiography include Digital Subtraction Angiography (DSA) [1], but a major drawback for DSA is that it cannot visualize blood flow.

Our visualization was created using JavaScript and the HMTL5 Canvas API. We generated a vector field and randomly create particles on a grid. Every second, each particle moves along the vector and its new position is painted onto the grid. If the particle reaches a dead end it stops moving, dies, and gets regenerated at another random point. Results are covered in Section 4 (Results).

Algorithm 1 Flow Direction Pursuit

procedure FINDPATH(img)
  if *queue* $\leftarrow$ empty **then**
    for `(r,c)` in img **do**
      if img`(r,c)` $= 1$ **then**
        Add current key to list of *root* keys
        Create new Segment object S_{new}
        Add `(r,c)` to S_{new}'s path
        Run **RecursiveSearch**`(r,c)` to populate S_{new}'s path and *queue* for next iteration
        Add S_{new} to dictionary of *Segments*
        return `return code` $= 1$, dictionary of *Segments*, list of *root* keys
    return `return code` $= 0$, dictionary of *Segments*, list of *root* keys
  else
    while *queue* is not empty **do**
      Last visited coord obj $C \leftarrow queue.pop()$
      Create new Segment object S_{new}
      Add $C.row$ and $C.col$ to S_{new}'s path
      Set S_{new}'s parent as $C.parent$
      Run **RecursiveSearch**`(C.row,C.col)` to populate S_{new}'s path and *queue* for next iteration
    return `return code` $= -1$, dictionary of *Segments*, list of *root* keys
procedure RECURSIVESEARCH(img)
  Stopping Condition: offsets are 0, goes out of bound, **OR** reaches a junction
  Generate 3×3 mask with `(r,c)` as center
  Run **NextPixel**`(r,c)` to get (r_{new}, c_{new}), and whether junction is reached
  return *Segment* object with updated path, *queue* with updated coord objects
procedure NEXTPIXEL(img)
  Check 3×3 mask for white pixels except `(r,c)`
  if number of white pixels > 1 **then**
    Set junction to $True$
    Create Coord object
    Add junction's point and current segment's parent to Coord object
    Enqueue each junction into *queue*
    return (r,c) and junction $= True$
  else
    if number of white pixels $= 1$ **then**
      return (r_{new}, c_{new}) and junction $= False$
    else
      return `(r, c)` and junction $= False$

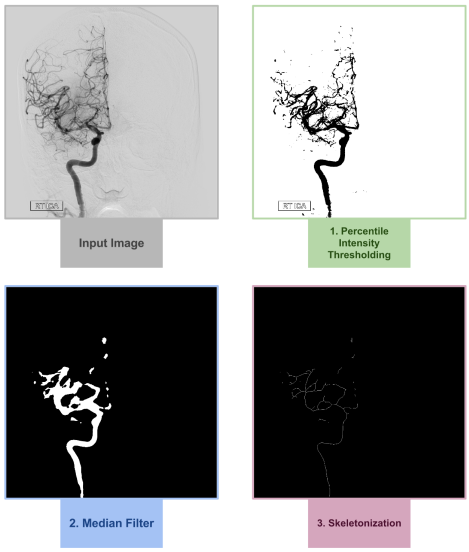
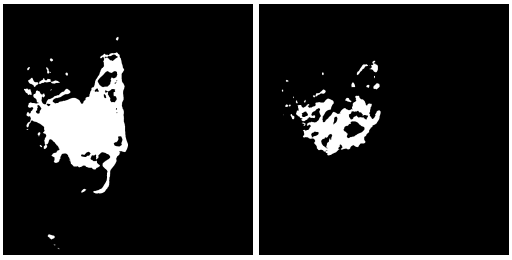


Figure 4: Our 3-step image processing algorithm: 1. Percentile Intensity Threshold; 2. Binary Invert + Median Filter 3. Skeletonization.

3 Evaluations

3.1 Isolate blood vessels



(a) (b)

Figure 3: (a): Naive threshold; (b): Percentile Intensity Threshold.

We had some issues filtering the images initially because we used a hard-coded threshold value. This is generally not a good idea because it doesn't focus on which pixels are considered to be the focus in the frame. Figure 3 shows the difference.

After using our percentile intensity threshold, we had some promising skeletonizations. Figure **??** shows the steps in our processing algorithm.

3.2 Blood Flow Direction Algorithm

For segments that are not connected to the main artery, a challenge arise when we are not sure if we got the direction of the segment correct. A potential error we encountered is when a new segment's endpoint is chosen as root, but that endpoint is very far from the initial artery endpoint, making the vector field of the segment flow back into the artery, which is not our desired result.

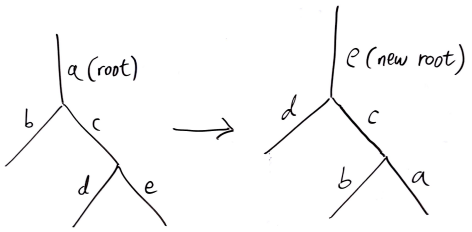


Figure 5: A simplified example: when segment *e* becomes the new root, we perform a reorder of the tree so that the paths are reversed and the direction is changed so the flow comes from *e*.

To resolve this, we use a comparison heuristic to find the minimum euclidean distance between the endpoints of a new segment vs. the leaf endpoints of our segment with a known direction. If the endpoints pairs with the minimum euclidean distance [4] is note the intial root we chose, then it means that we have to *reverse* the entire path, with the correct leaf being the root. Figure 5 shows a simplified example. This becomes a tree-reversal problem, which has been accomplished before. Note: this is not like an AVL tree, which is a form of binary tree that self balances.

4 Vector Field Animation

The skeletonizations were key in performing vector field animation. From the skeletons, we can create vectors to describe the path. We then create a vector field by mapping the coordinates on the original image to the nearest neighboring skeleton. More results are shown in the Results section.

5 Results

5.1 Blood Flow Direction Algorithm

Figure 6 shows a screenshot of the rendered visualization. Notice that vectors are moving up the artery and down. Figure 7 shows some detailed sections, including the artery and the blood vessels.

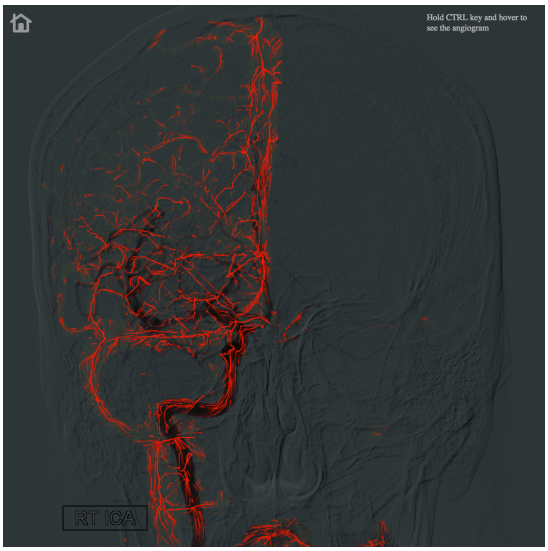
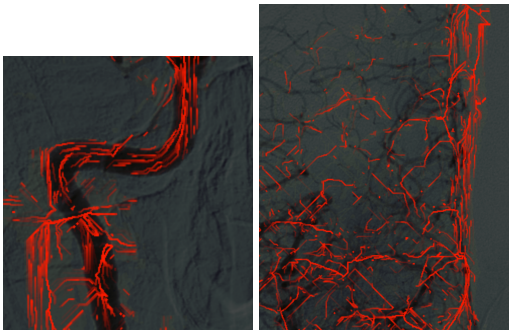


Figure 6: A simplified example: when segment e becomes the new root, we perform a reorder of the tree so that the paths are reversed and the direction is changed so the flow comes from e.



(a) (b)

Figure 7: (a): Details on the artery; (b): Details on the small vessels as they spread out.

We have processed and calculated our dicom video's vector field, then rendered the visualization online at `https://arvinn.github.io/cs188-test/`. Futhermore, notes on various papers and links that we've read throughout the class can be found at `http://kfrankc.me/cs188/`.

Lastly, our codebase is on Github at `https://github.com/kfrankc/cs188`.

6 Discussion and Acknowledgements

As mentioned in the introduction, visualizing this is an unsupervised learning problem, as we were not given any labels to this data; in fact, the only data we were given is the dicom file itself.

In the future, we hope to work with dicom files that have pre-labeled data describing the behavior of the blood flow; we can then train some supervised learning algorithms to better predict direction. In addition, we hope to clean up the codebase, then make the project open-source; intelligent path finding in dicom files seems to be a relatively new field, so this is exciting, and we are looking forward to seeing what the community contributes.

In this project, we were exposed to an application field of Computer Science we were not familiar with before: Medical Imaging. Throughout the 10 weeks, we have learned not only the various technologies behind these medical devices, but also the algorithms that make them possible. We want to thank Prof. Scalzo for his constant support in our project.

References

[1] BRODY, W. R. Digital subtraction angiography. *IEEE Transactions on Nuclear Science 29*, 3 (1982), 1176–1180.

[2] CHANG, S. G., YU, B., AND VETTERLI, M. Spatially adaptive wavelet thresholding with context modeling for image denoising. *IEEE Transactions on image Processing 9*, 9 (2000), 1522–1531.

[3] HUANG, T., YANG, G., AND TANG, G. A fast two-dimensional median filtering algorithm. *Acoustics, Speech and Signal Processing, IEEE Transactions on 27*, 1 (Feb. 1979), 13–18.

[4] LIN, M. C., AND CANNY, J. F. A fast algorithm for incremental distance calculation. In *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on* (1991), IEEE, pp. 1008–1014.

[5] MOORE, E. F. The shortest path through a maze. In *Proc. Int. Symp. Switching Theory, 1959* (1959), pp. 285–292.

[6] MOZER, M. C., AND SMOLENSKY, P. Skeletonization: A technique for trimming the fat from a network via relevance assessment.

[7] SCALZO, FABIEN, L. D. S. Perfusion angiography in acute ischemic stroke. *Computational and Mathematical Methods in Medicine 2016*, 3 (2016), 1–14.