Contents

1 Project Overview

1.1 FullSystem Walkthrough

1. First we generate a 5 (layer) x 16 (hits) array of integer coordinates, where each coordinate corresponds to a pixel on the layer where a fragment was detected.

2. Declare MM, the 5 (layer) x 16 (nodes) x 2 (adjacent layers) x 16 (adjacent nodes) array that stores the links that denotes the path taken between nodes.

3. Declare tripletMatrix and tripletMatrixTranspose: tripletMatrix is a 3 (middleLayers) x 16 (nodes) x 16 (aboveNodes) x 16 (belowNodes) matrix that stores triplets. tripletMatrix-Transpose is a 3 (middleLayers) x 16 (nodes) x 16 (belowNodes) x 16 (aboveNodes) whose innermost 2D matrix stores the transpose of the innermost 2D matrix of tripletMatrix.

4. Enter powerLoop; The counter in powerLoop determines the max number of "good" triplets per node in a given stage of the algorithm, e.g. (power starts at 4) each node begins with 2^4 x 2^4 possible good triplets. Then in the next iteration of powerLoop (power = 3), each node has at most 2^3 x 2^3 good triplets, and so on.

5. initialize tripletMatrixPruned, which is filled with the remaining possible good triplets for each node following the pruning stage from the last iteration of powerLoop. E.g. after the 2^4 x 2^4 case is pruned, we have at max 2^3 x 2^3 possible triplets per node, and these are filled into tripletMatrixPruned.

6. Assume power == 4 for the moment, so we skip the power ! = 4 if statement.

7. Initialize MM to all reject links. This has to be performed in every iteration of the power loop, otherwise, bad links will persist in MM, and we won't be able to fully prune

8. Begin the middleLayer loop, where each middleLayer index corresponds to the index of the bottommost middleLayer, and iterates up to the top middleLayer

9. Begin the nodeIndex loop, which begins at node 0 and works through node numberOfNodes

10. power == 4 in this case, so enter the if statement; calculate the Laplacians for each possible triplet and store them in tripletMatrix and tripletMatrix transpose. Then, make a copy of these matrices. The copy is made because MinFinder directly modifies its input (i.e. triplet-Matrix). But we need to maintain the order of tripletMatrix to fill up tripletMatrixPruned in the next iteration. This is because MM stores acceptLinks, and the way we fill tripletMatrix-Pruned is by iterating through the acceptLinks of MM, and finding the triplet in tripletMatrix that corresponds to the MM acceptlinks; the way we find this correspondence is based on the ordering of triplets in tripletMatrix. So we cannot change this ordering.

11. Declare laplacianMinimums. laplacianMinimums stores the triplets with the smallest Laplacian values per adjacent node to nodeIndex in laplacianMinimums (above nodes) or laplacianMinimumsTranspose (below nodes). laplacianMinimums is an array of size 2^{power}, which

corresponds to the greatest possible number of triplets that were accepted the last time updateUpLink and updateDownLink were run

12. Given a node, for every above node, find the triplet consisting of this node, the above node in iteration, and the belowNode that has the smallest Laplacian value, and store this in laplacianMinimums. Do the reverse for the transpose (this node, below node in iteration, and the aboveNode that has the smallest Laplacian value).

13. Put the smallest sizeof(laplacianMinimums)/2 triplets stored in laplacianMinimums at the front of laplacianMinimums

14. For the smallest sizeof(laplacianMinimums)/2 triplets stored at the front of lapalcianMinimums, update the up links corresponding with the triplets to acceptLink in MM. For the smallest sizeof(laplacianMinimums)/2 triplets stored at the front of laplacianMinimumsTranspose, update the down links corresponding with the triplets to acceptLink in MM.

15. For each node in the middleLayer, reject all links that are currently accept that are only one way (i.e. a node has an acceptLink to an adjacent node, but that node does not have an acceptLink to this node).

16. power–

17. now go into the if (power ! = 4) loop.

18. We first initialize tripletMatrix pruned to have dummy triplets with an extremely large Laplacian value. This is because after the first power loop, we cannot gaurantee that we will have exactly 2^{power} x 2^{power} triplets for each node; that is just the max value. But we run MinFinder on tripletMatrixPruned, which tries to find the smallest Laplacian value in a subset of triplets. So we need to ensure that in the case where we have less than 2^{power} x 2^{power} triplets, only real triplets are returned as the minimums; so the dummy values are larger than any possible Laplacian value to ensure that a dummy value is never returned as the minimum.

19. Then we fill tripletMatrixPruned according to the remaining acceptLinks in MM after the last round of pruning. Lastly, we create a copy of tripletMatrixPruned for the same reason as we created a copy of tripletMatrix mentioned earlier.

20. Repeat the above process until we have at most 1 up link and 1 down link per node.