Delaunay Triangulation on the GPU

Dan Maljovec

CPU Delaunay Triangulation

Randomized Incremental Algorithm

1. Construct Bounding triangle
2. Choose point to insert randomly
3. Locate triangle containing point
4. Construct new children triangles by connecting new point to each point of containing triangle
5. Test edges and flip if necessary
   - If flipped, test edges and recursively flip if necessary
6. Repeat 2-5 until all points have been inserted

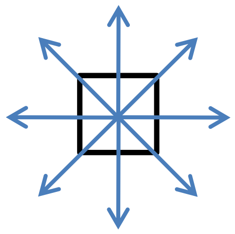
GPU Implementation

Use the dual: Voronoi Diagram

- Construct a discrete Voronoi Diagram (Jump Flooding Algorithm)
- Obtain a triangulation from the Voronoi
- Fix the estimations we made and perform edge flipping

Jump Flooding Algorithm

- Write the sites into a square texture
- Map each pixel in the texture (a point in discrete space) to a thread
- Initialize the step size to be half of the width/height of the texture
- Each pixel will look in 8 directions of size *step* to find closest site to it
- Divide the step size by 2 and Repeat



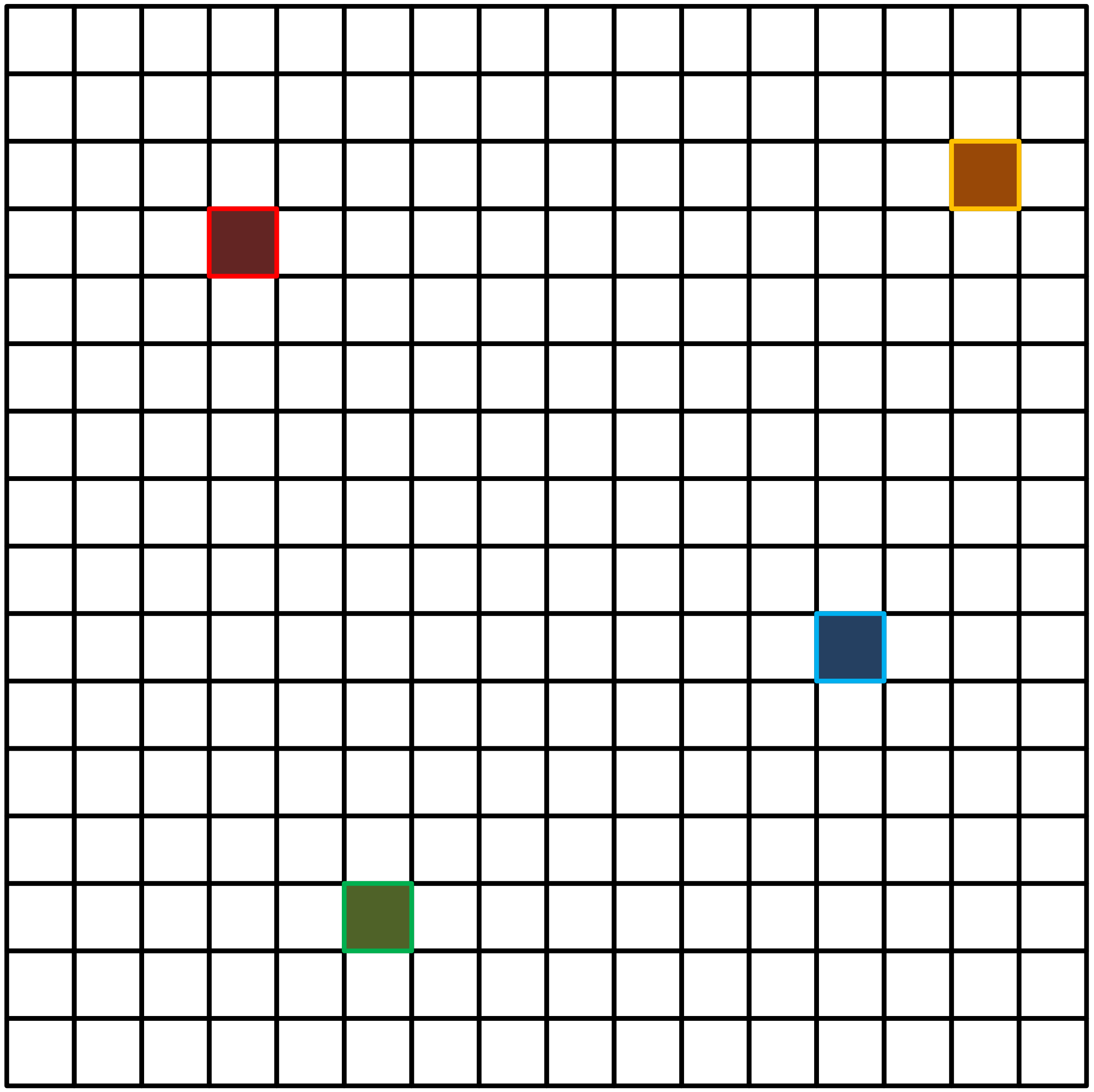
1+JFA

- Write the sites into a square texture

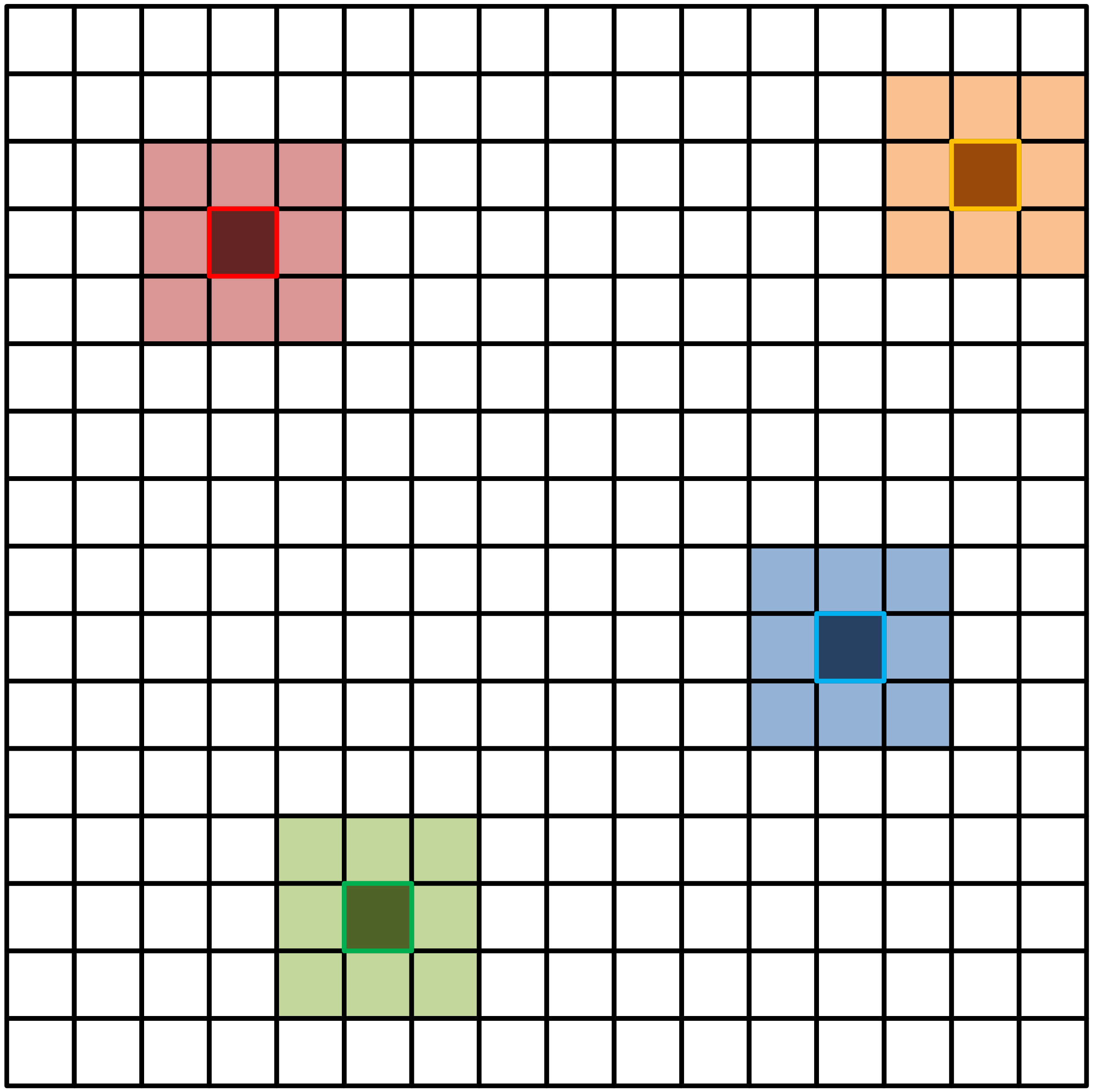
First do one round of step = 1,
then step = textureDimension / 2

- Map each pixel in the texture (a point in discrete space) to a thread
- Each pixel will look in eight directions of size step to find closest site to it
- Divide the step size by 2 and Repeat

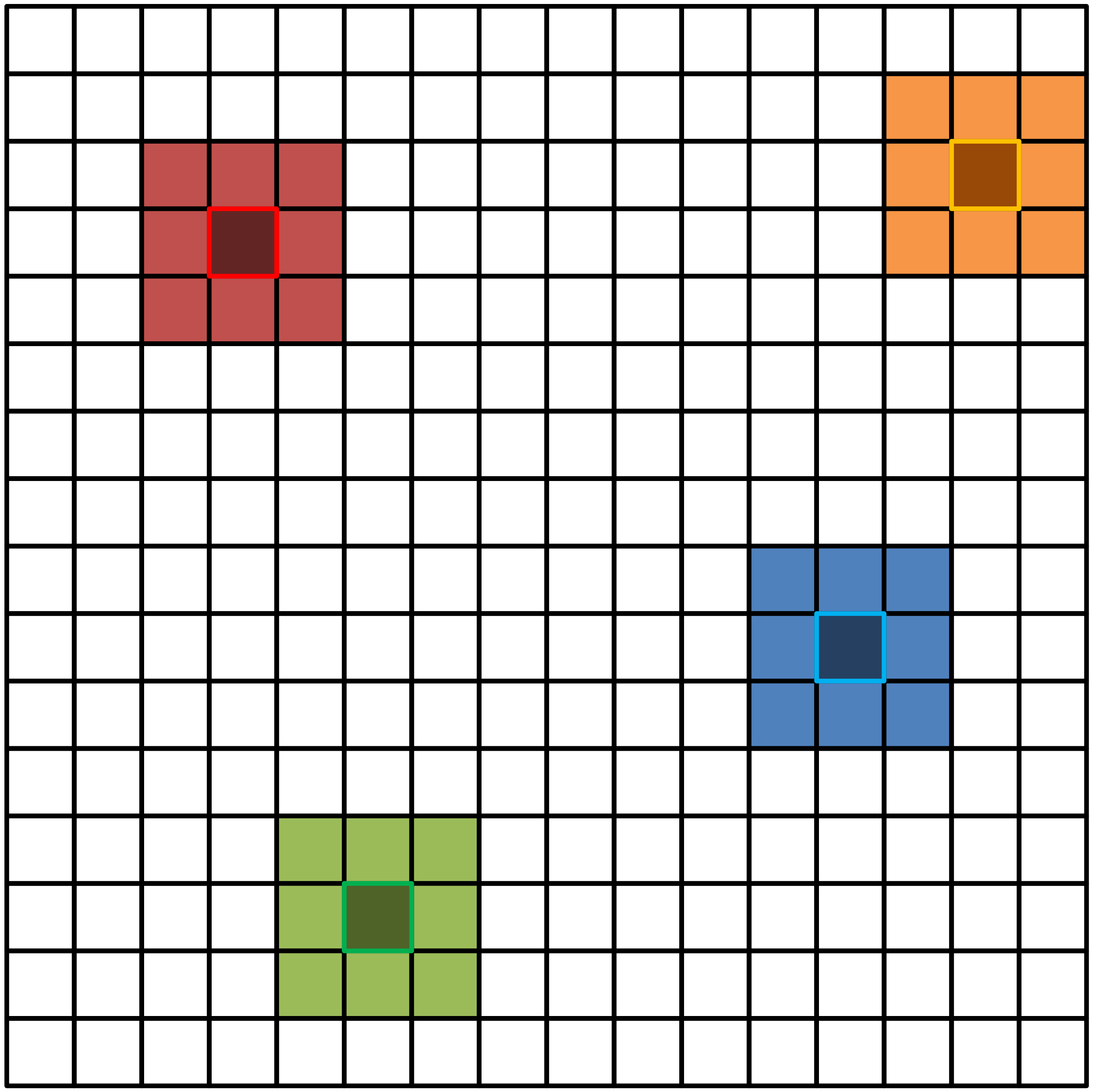
1+JFA (write sites)



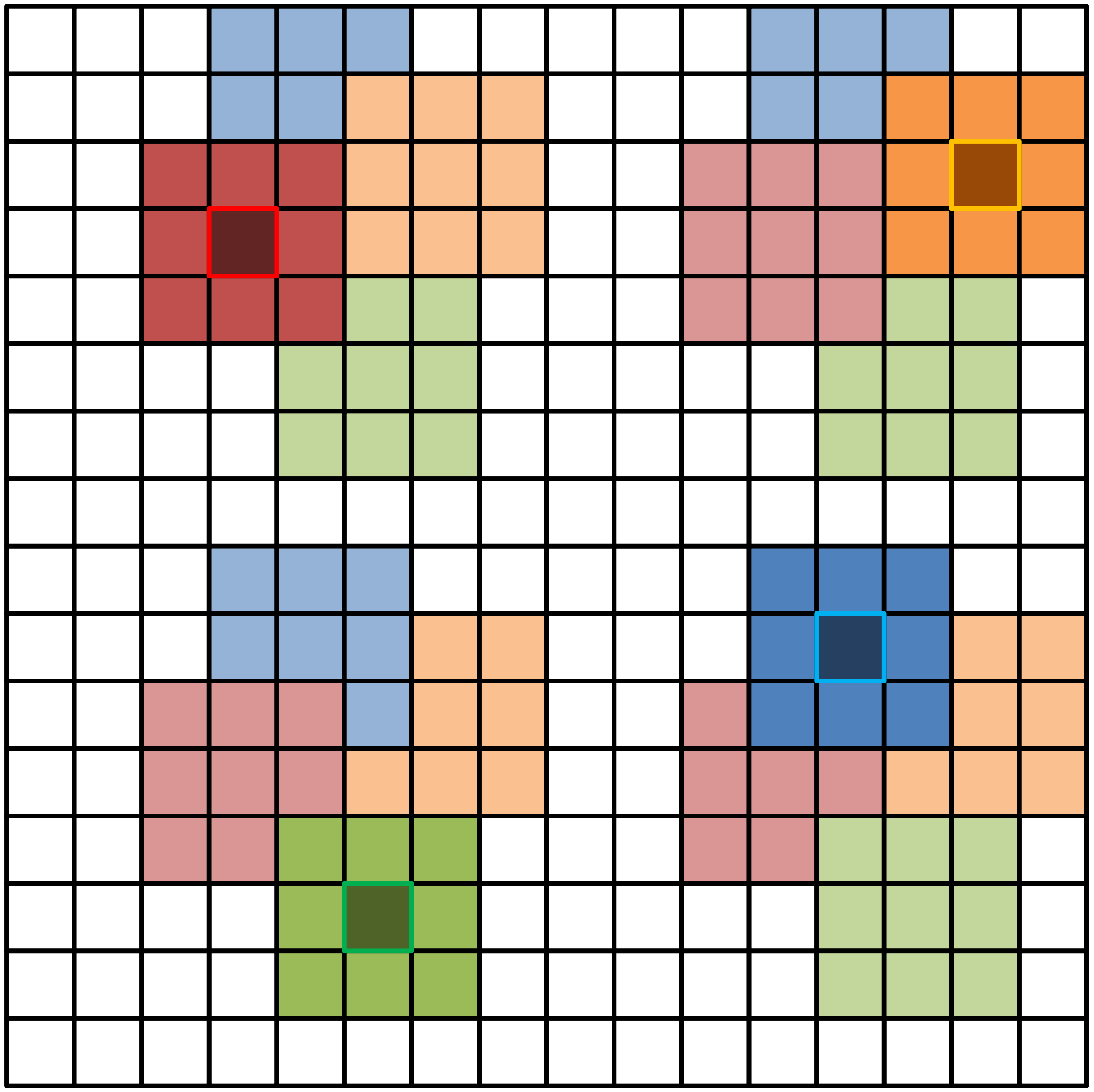
1+JFA (step size = 1)



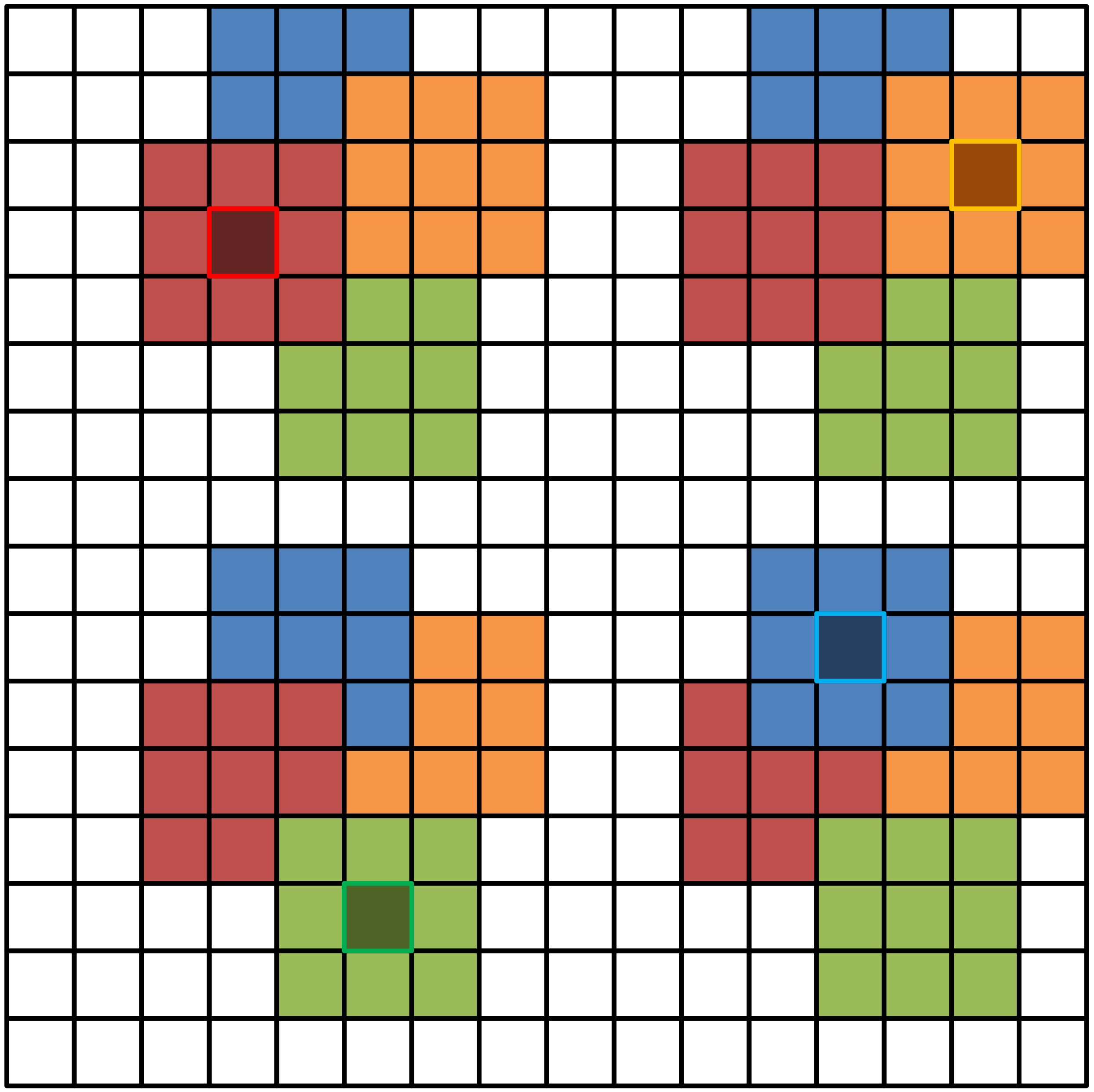
1+JFA (step size = 1)



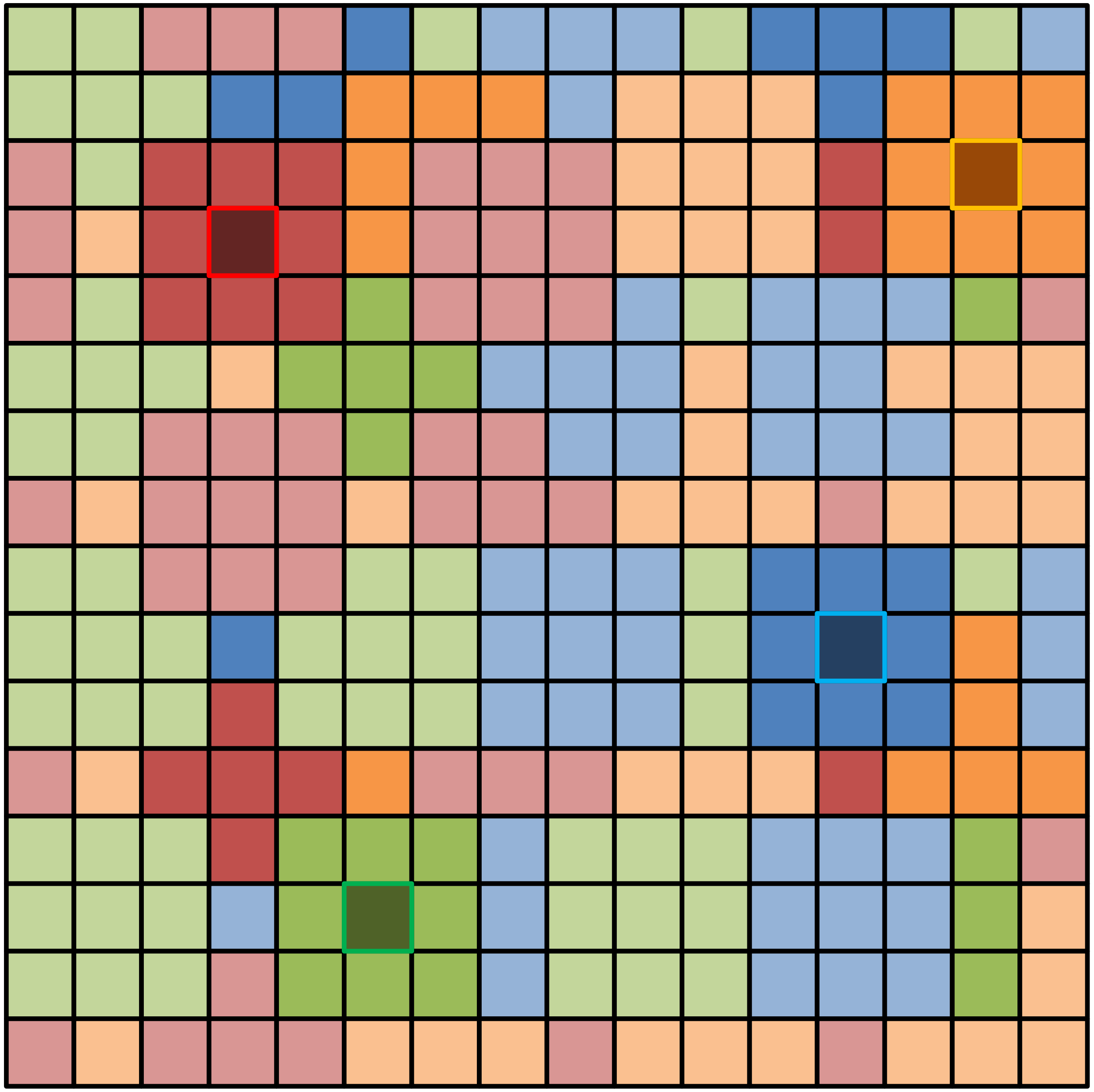
1+JFA (step size = 8)



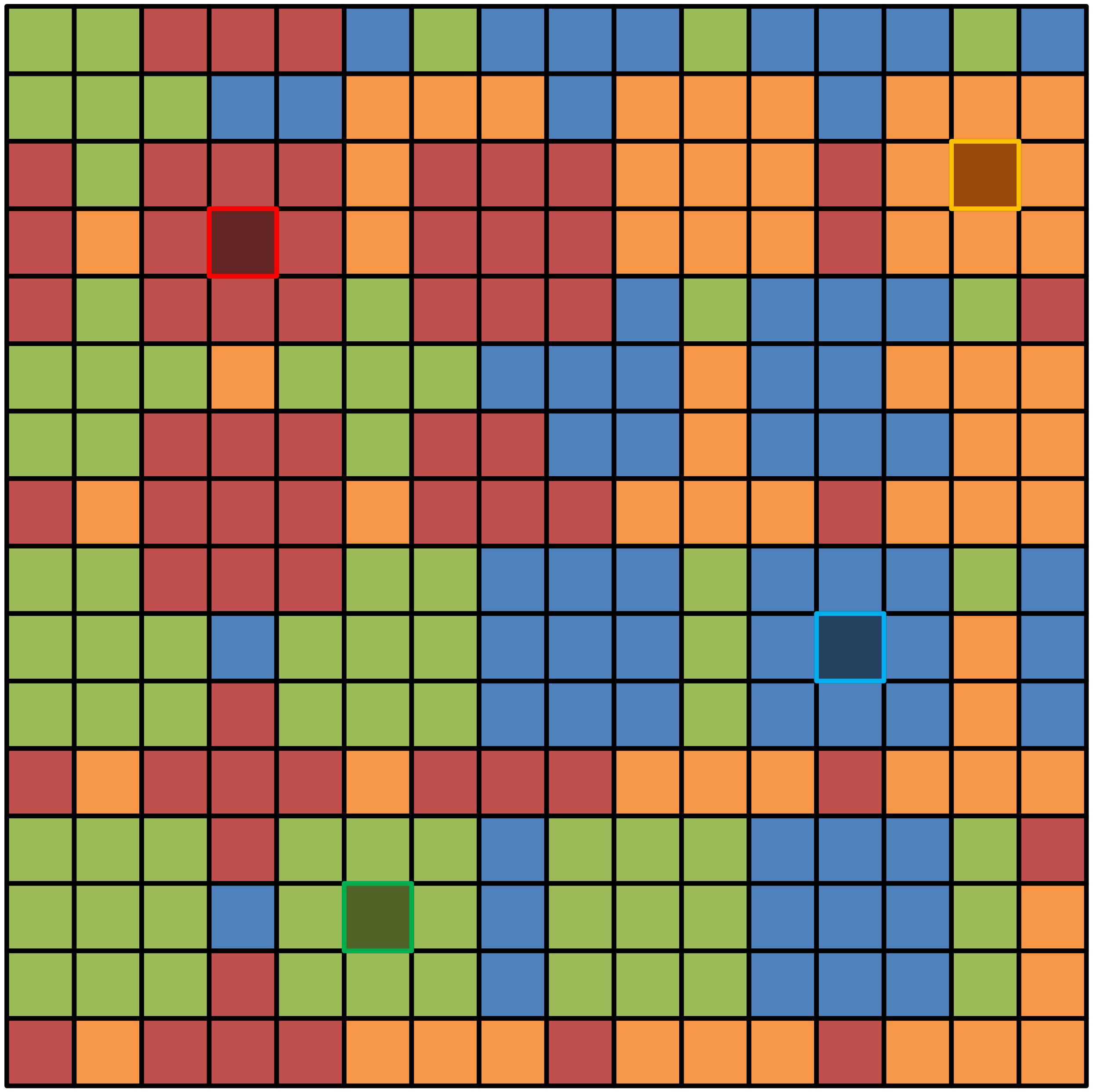
1+JFA (step size = 8)



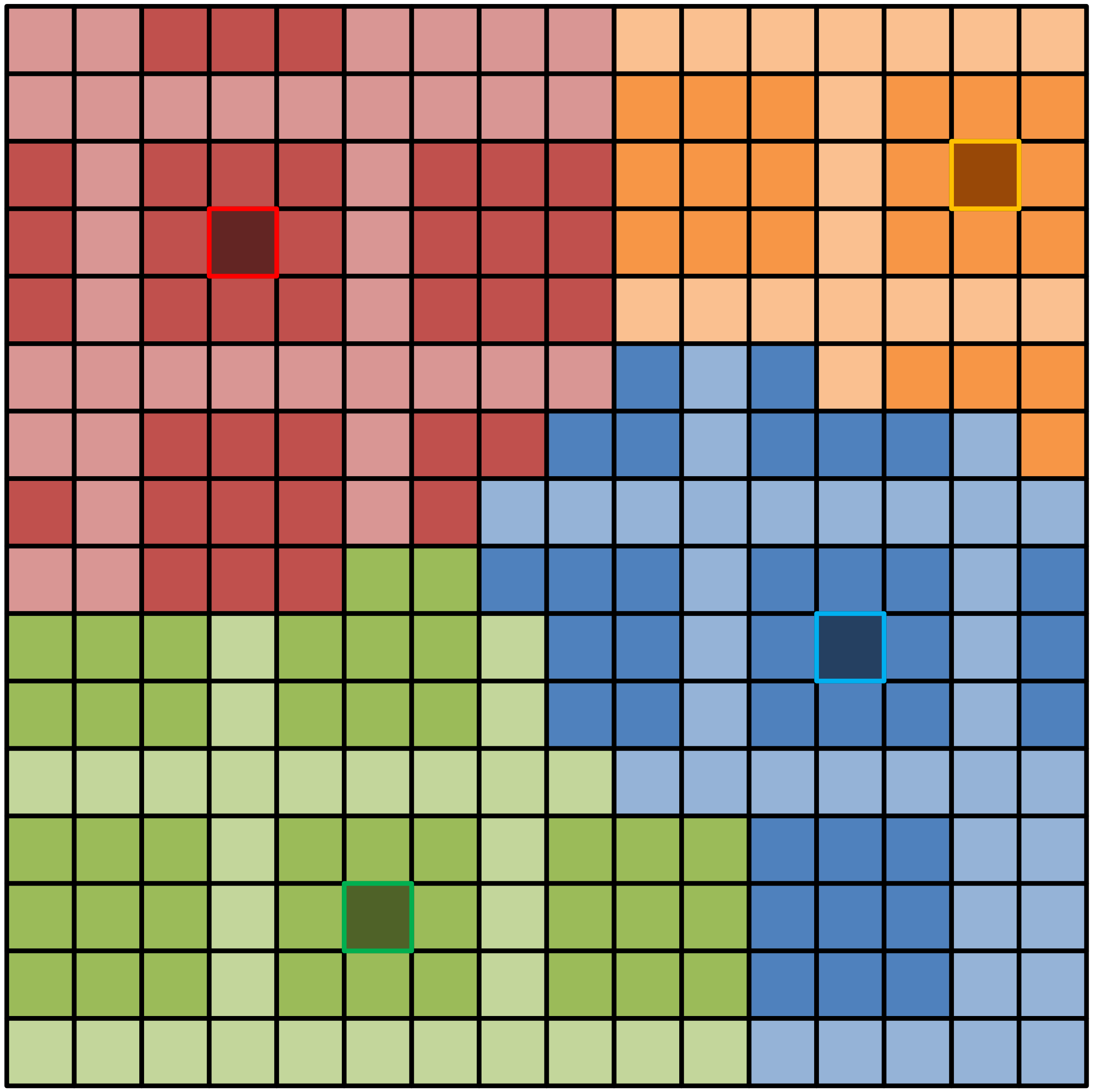
1+JFA (step size = 4)



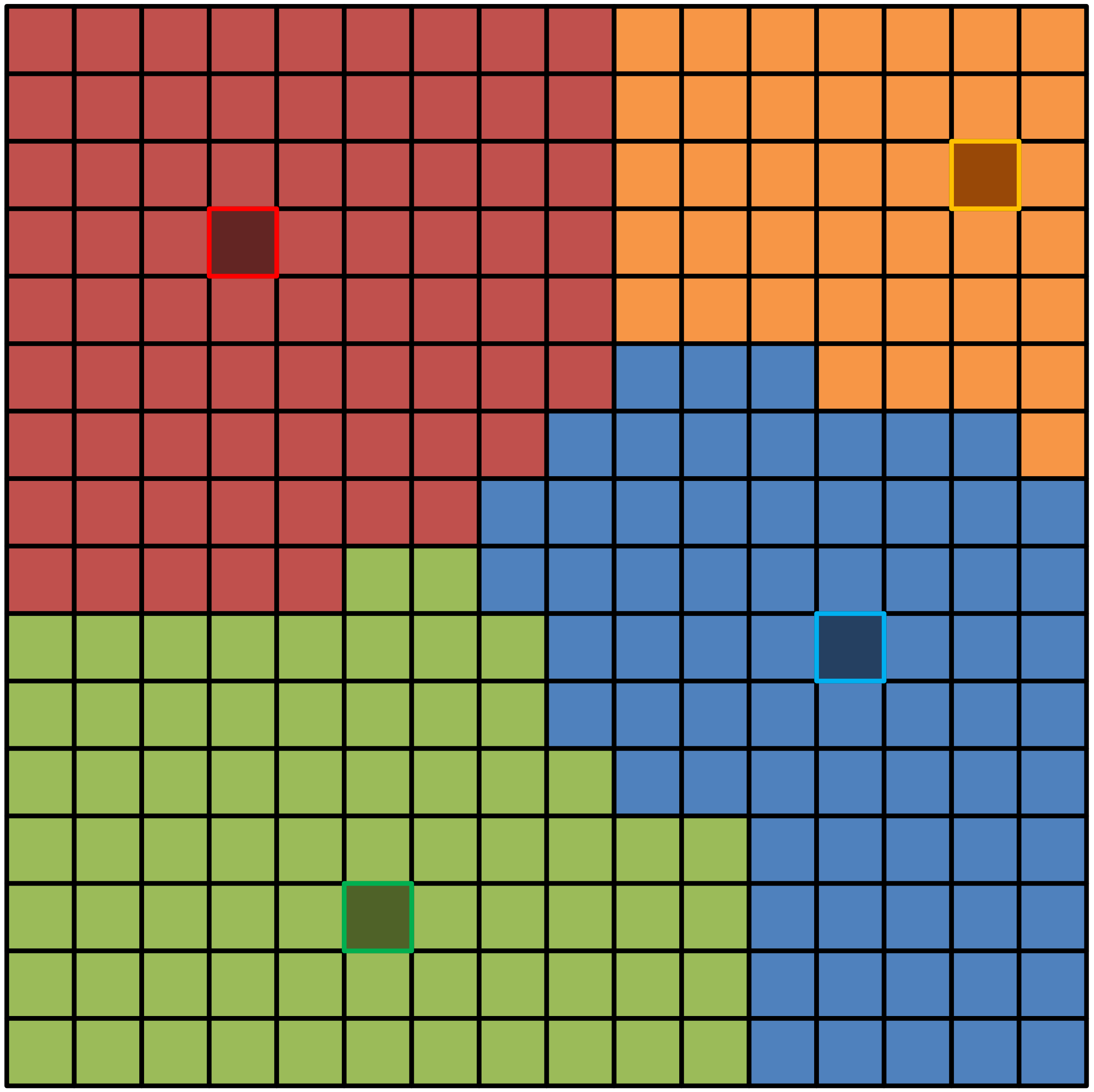
1+JFA (step size = 4)



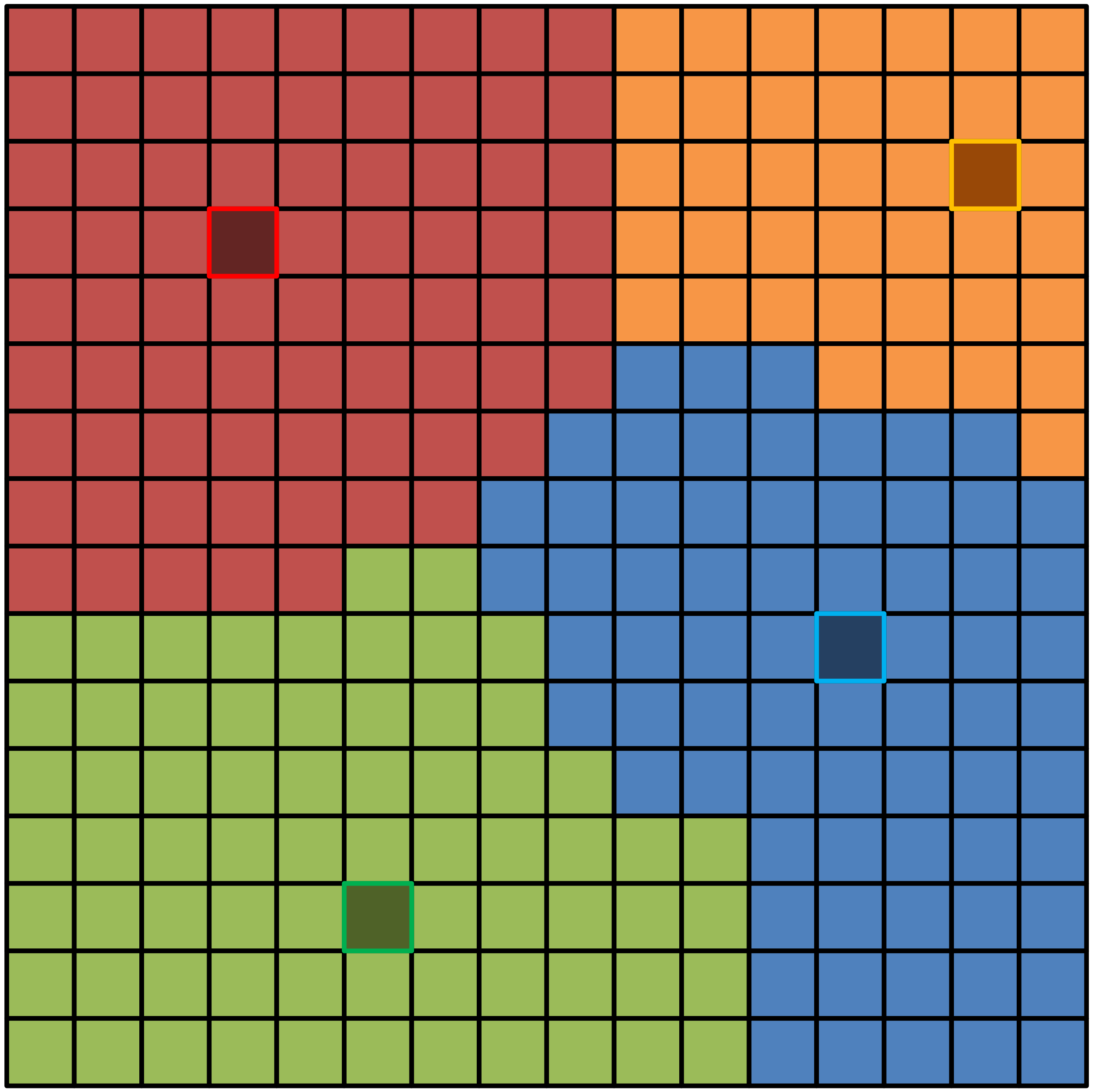
1+JFA (step size = 2)



1+JFA (step size = 2)

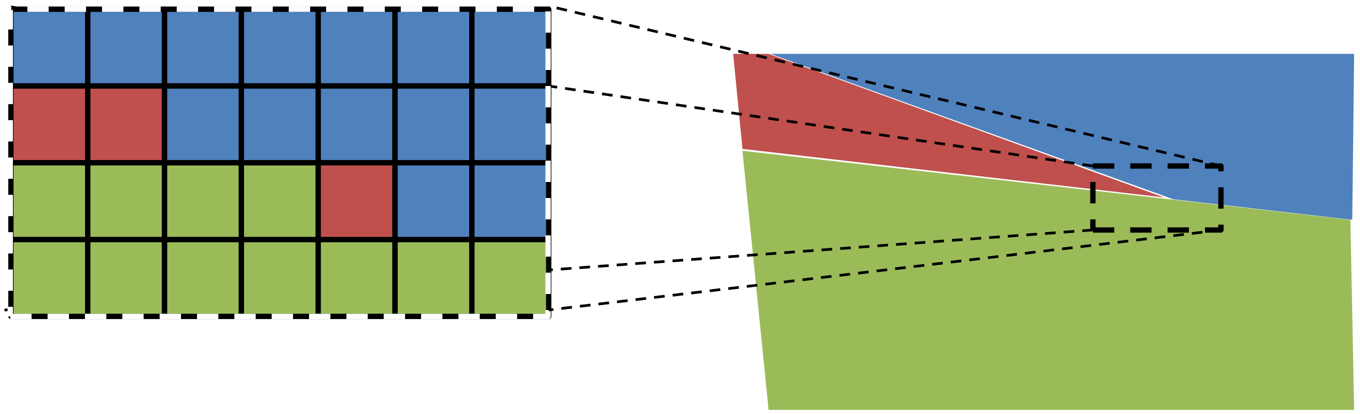


1+JFA (step size = 1)



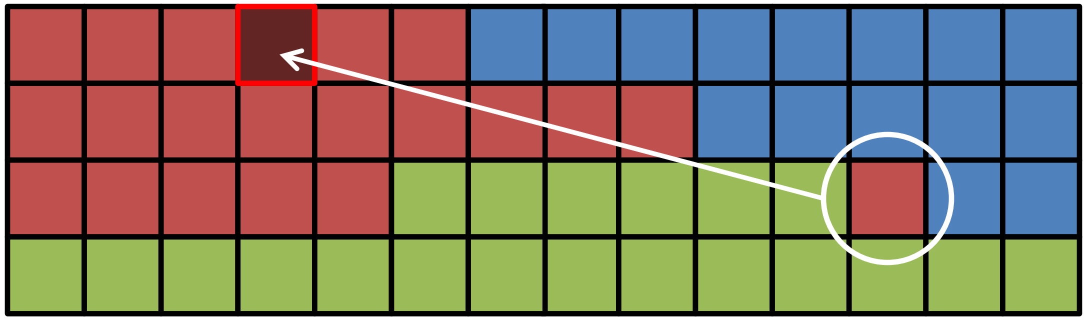
Errors from JFA

- JFA can introduce errors in the Voronoi diagram, which would lead to crossing triangles. These errors are manifested in "island" pixels as shown below:



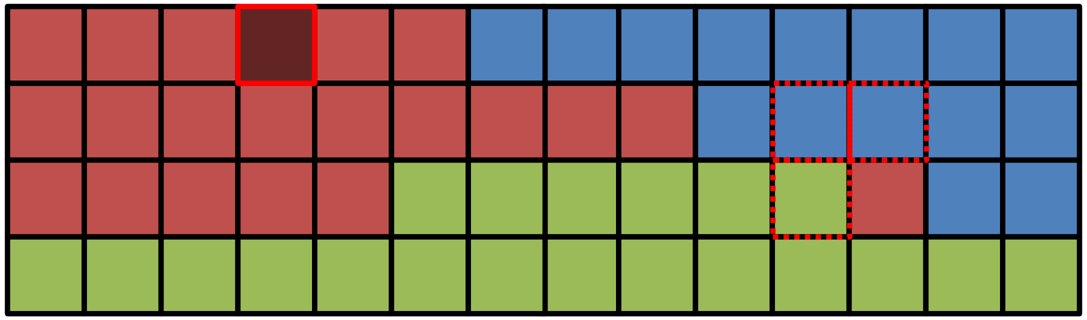
Detect and Remove Islands

- Map pixels to threads
- Knowing where a site is with respect to a pixel, we can look in that direction and see if this pixel is stranded
- Iterate this process until no more islands are found



Detect and Remove Islands

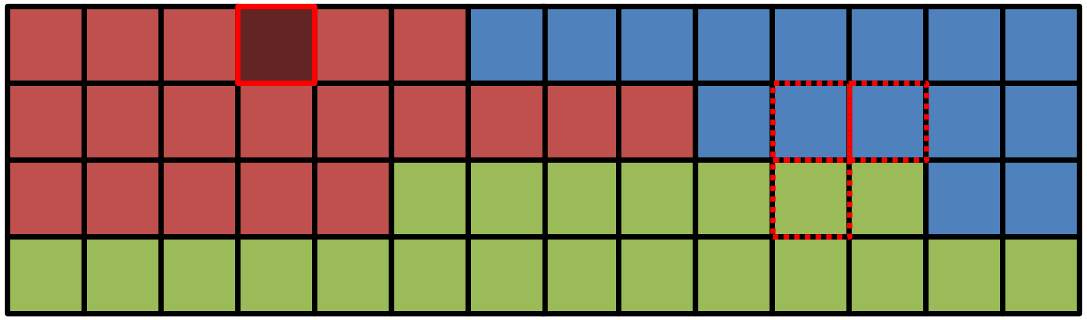
- Map pixels to threads
- Knowing where a site is with respect to a pixel, we can look in that direction and see if this pixel is stranded
- Iterate this process until no more islands are found



This pixel's site is up and to the right, so we will check neighbors in those directions to see if this is an island.

Detect and Remove Islands

- Map pixels to threads
- Knowing where a site is with respect to a pixel, we can look in that direction and see if this pixel is stranded
- Iterate this process until no more islands are found

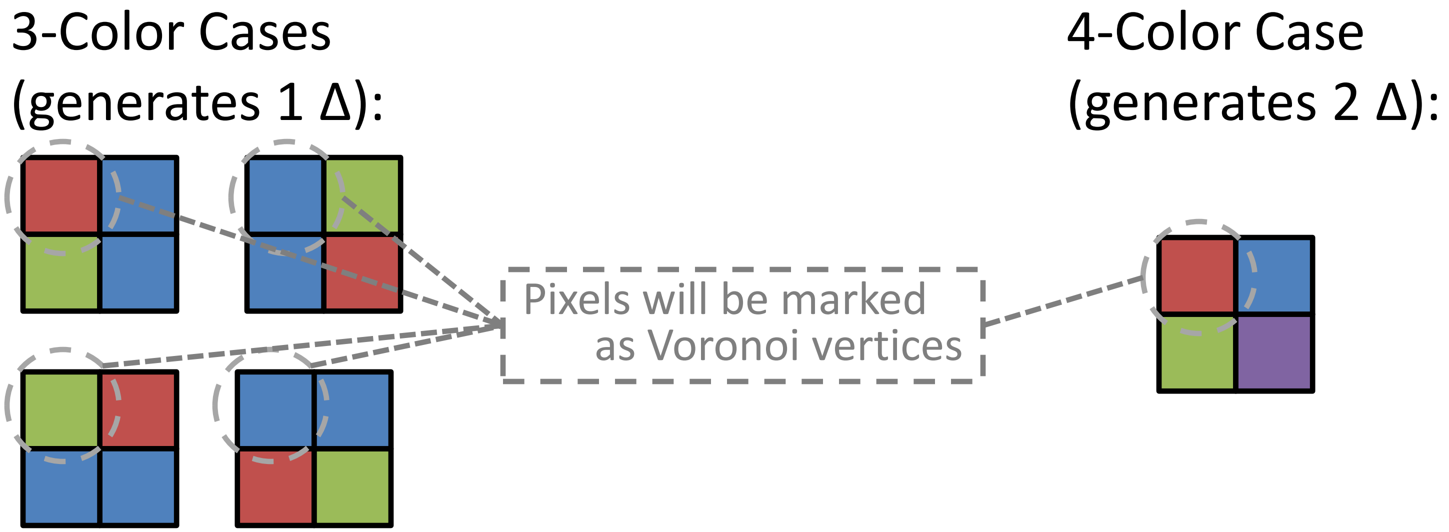


Update it to the closest site of the colors we find at the neighboring pixels

Find Voronoi Vertices

Vertices in the discrete Voronoi diagram, lie at junctions of 3 or 4 different colored pixels.

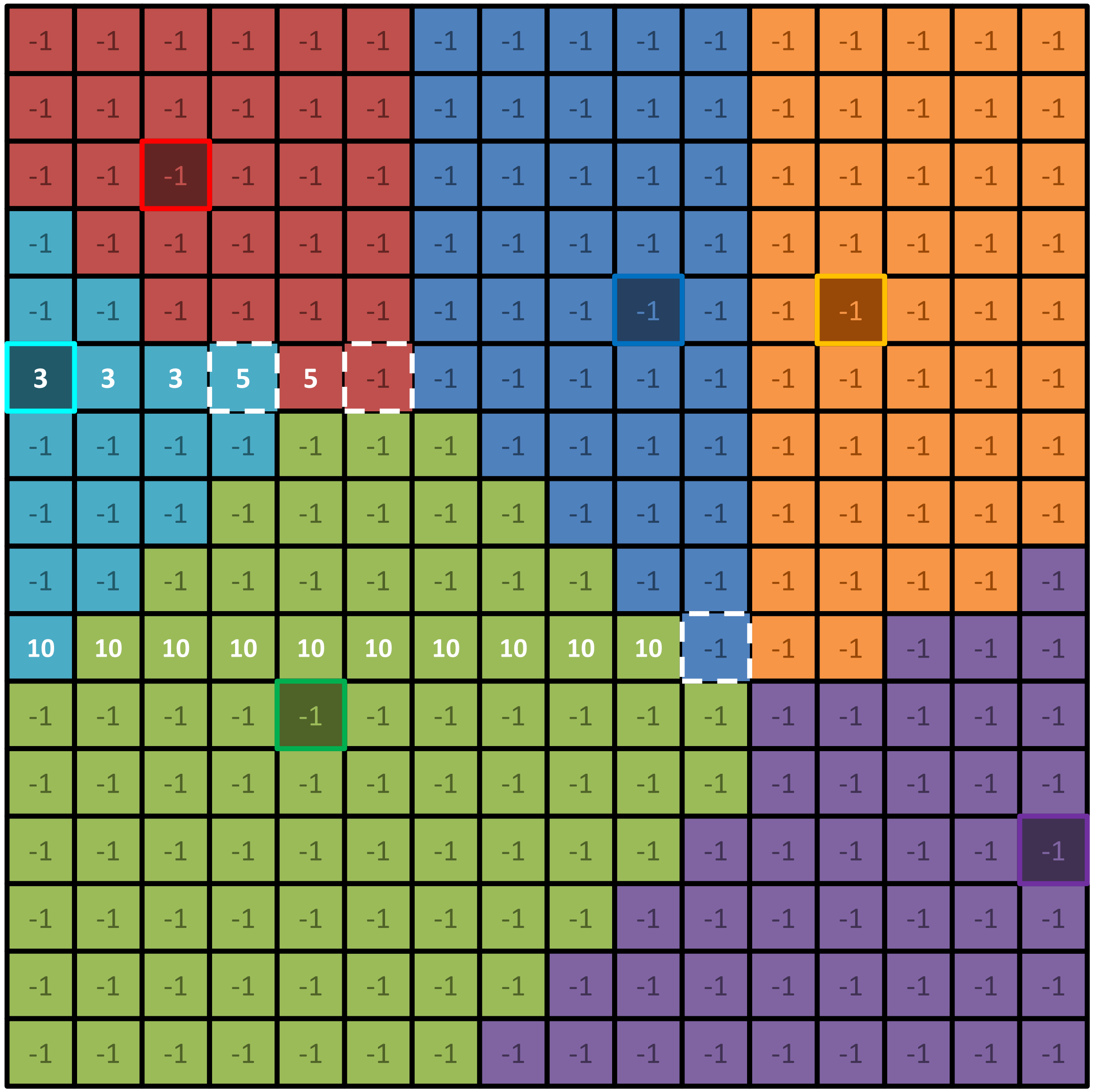
Map pixels to threads, and look right, down, and diagonally down and to the right to test if there exists a vertex at this pixel.



Chain Up Voronoi Vertices

- Parallel Prefix operation:
  - Map rows to threads, and for each pixel store the closest vertex in the row, to its right.
  - Allows us to apriori know how many triangles we are about to create, and allows us to be output sensitive in the next step.

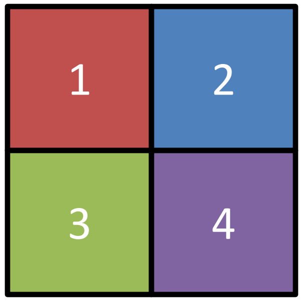
Chain up Vertices



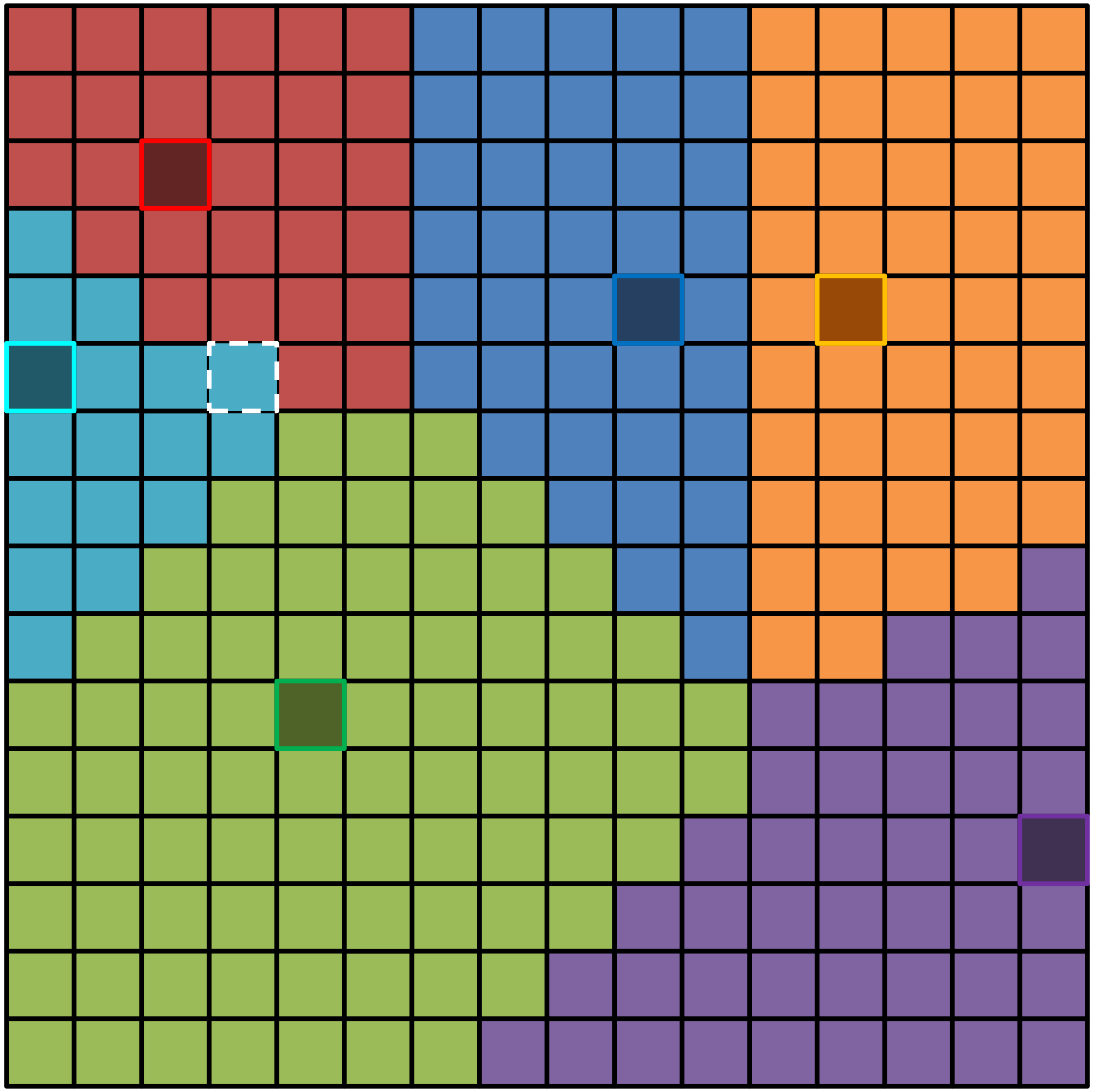
																#Vertices Row	Δ to Generate Row
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	0
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	0
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	0
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	0
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	0
3	3	3	5	5	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	**2**	**2**
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	0
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	0
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	0
10	10	10	10	10	10	10	10	10	10	-1	-1	-1	-1	-1	-1	**1**	**2**
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	0
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	0
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	0
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	0
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	0
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	0

Triangle Generation

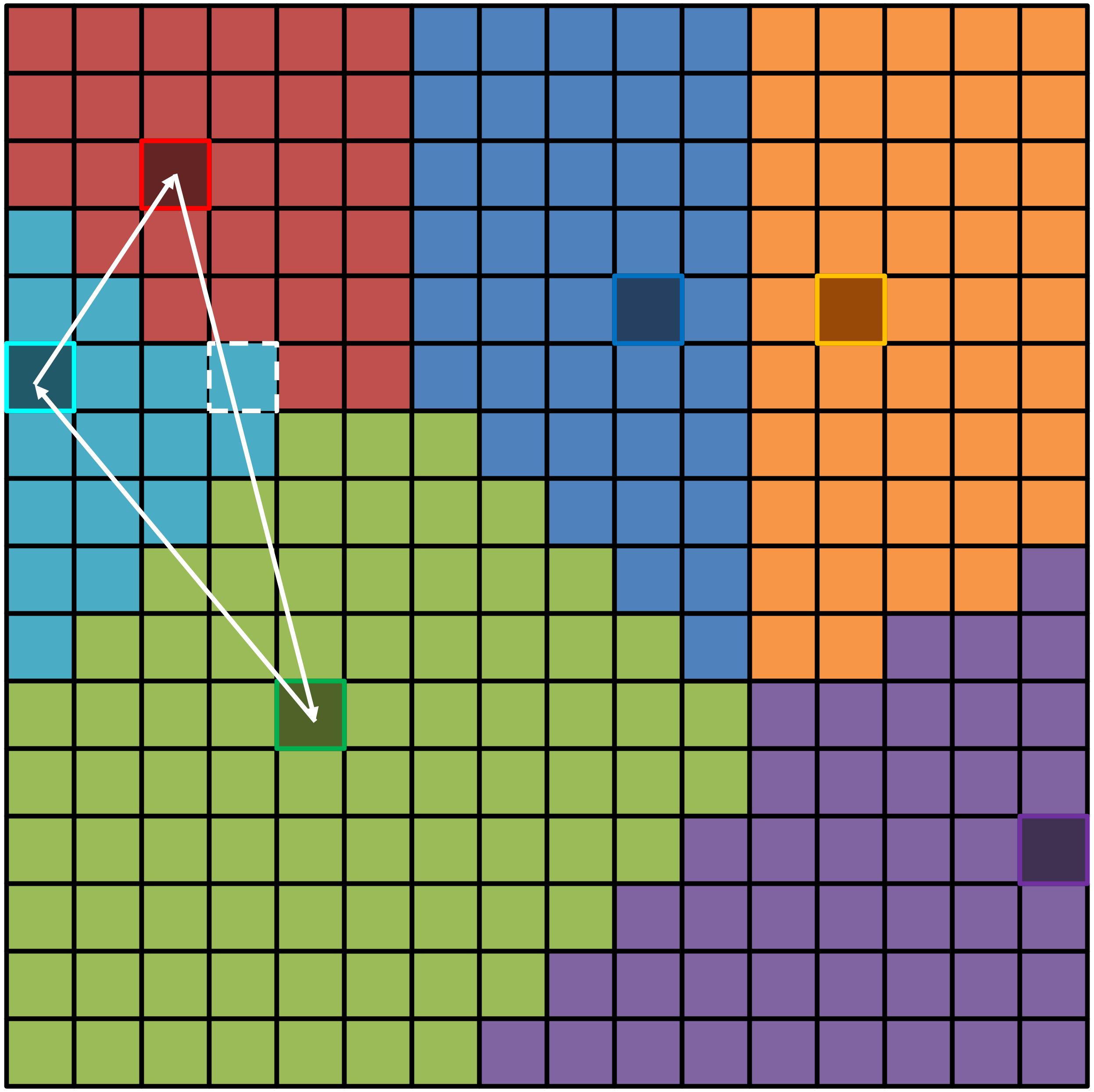
- Map thread to each row
- Create a triangle at each 3-Color vertex by connecting the 3 sites that contribute to the vertex
- Create 2 triangles at each 4-Color vertex by connecting sites of pixels 1, 2, & 3, then sites of pixels 2,4, & 3



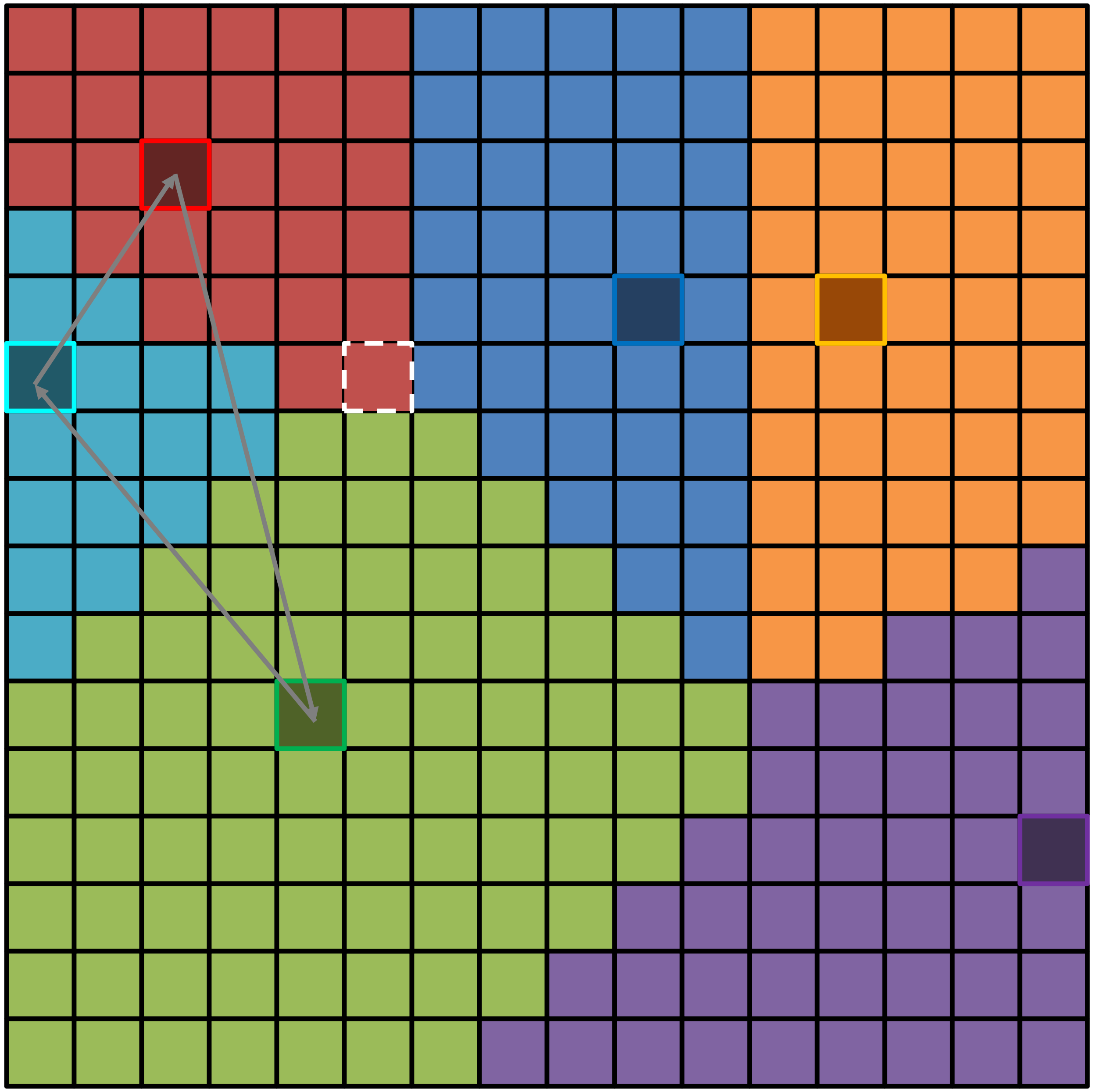
Generate Triangles



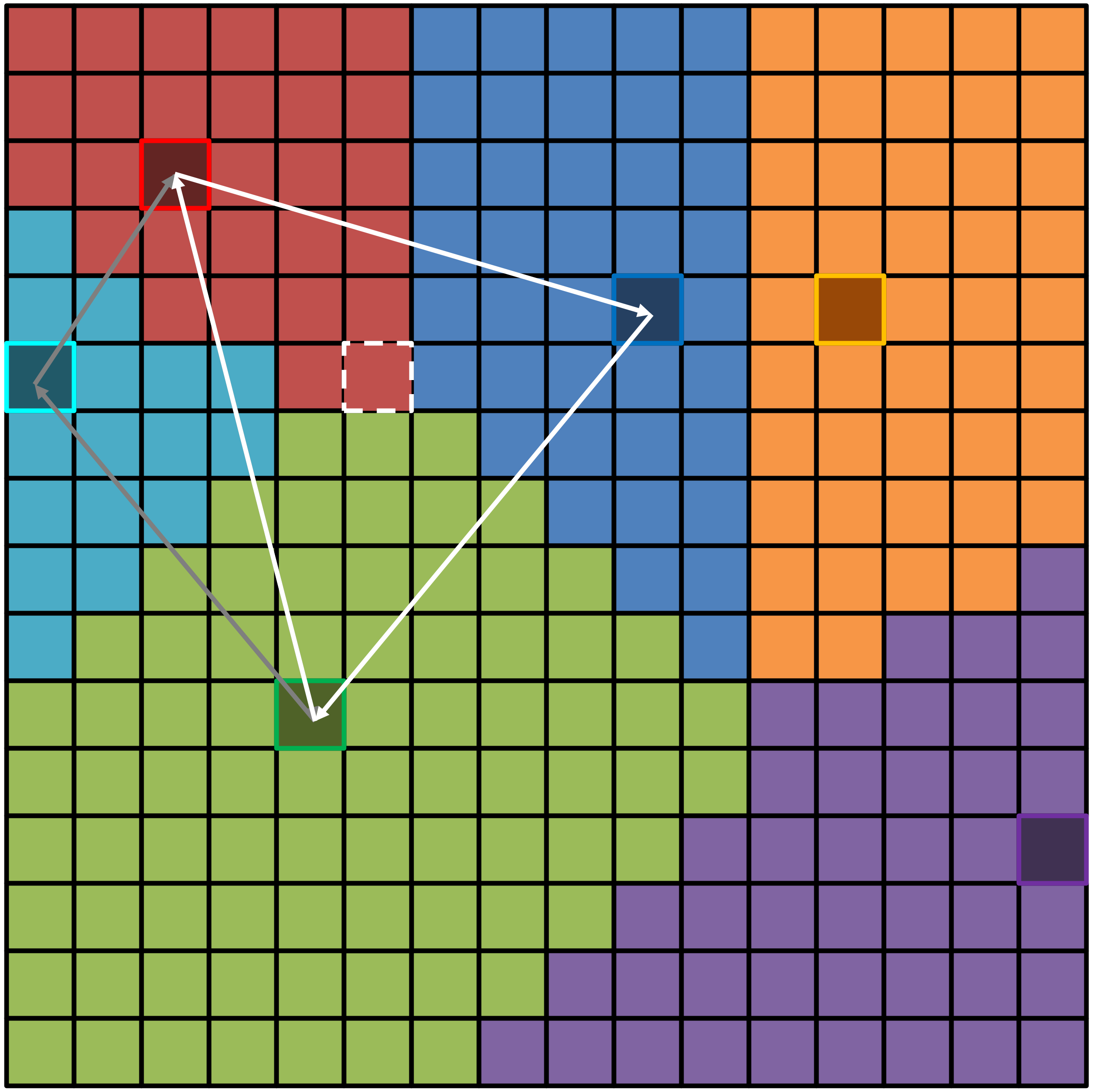
Generate Triangles



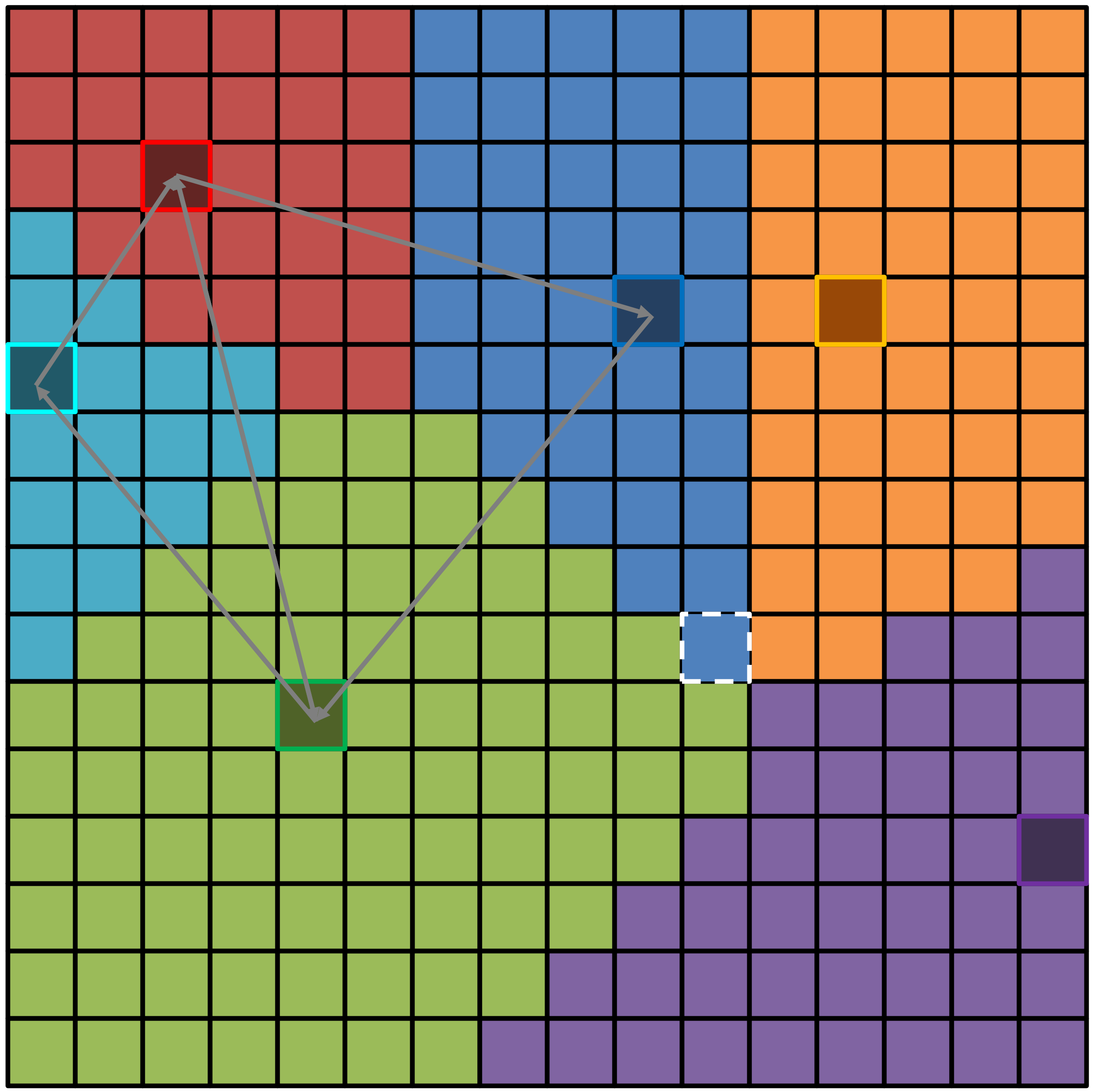
Generate Triangles



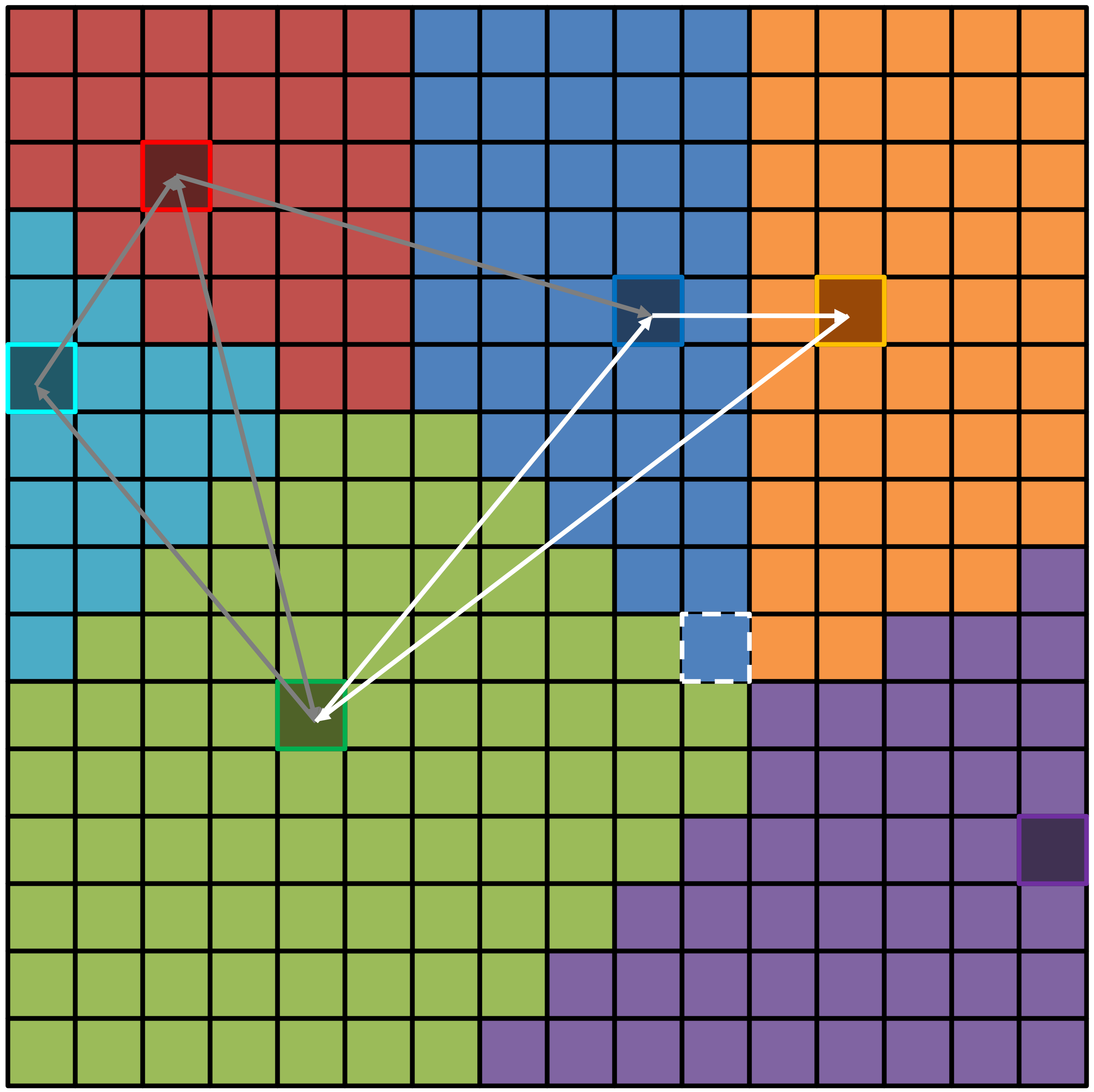
Generate Triangles



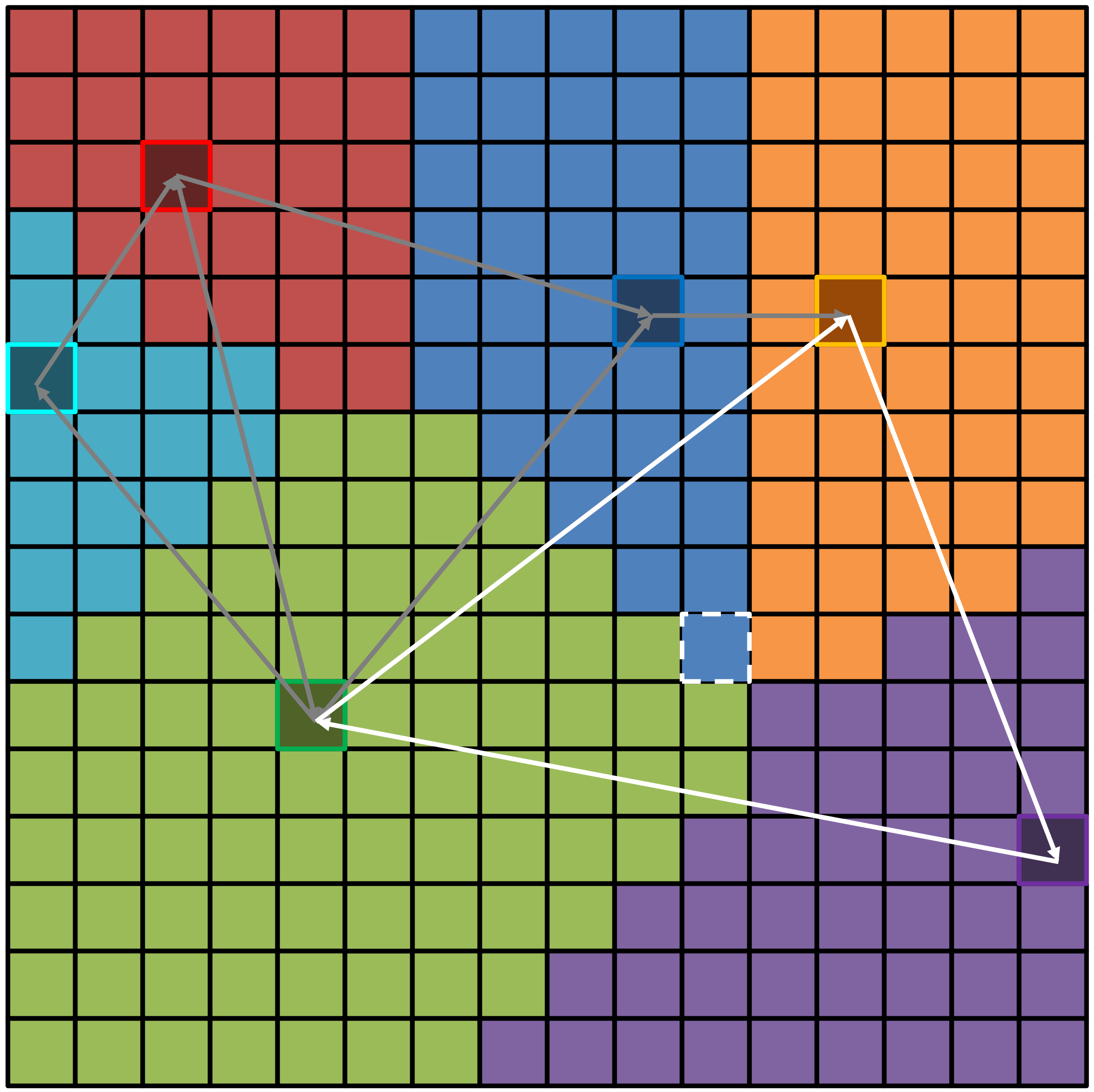
Generate Triangles



Generate Triangles



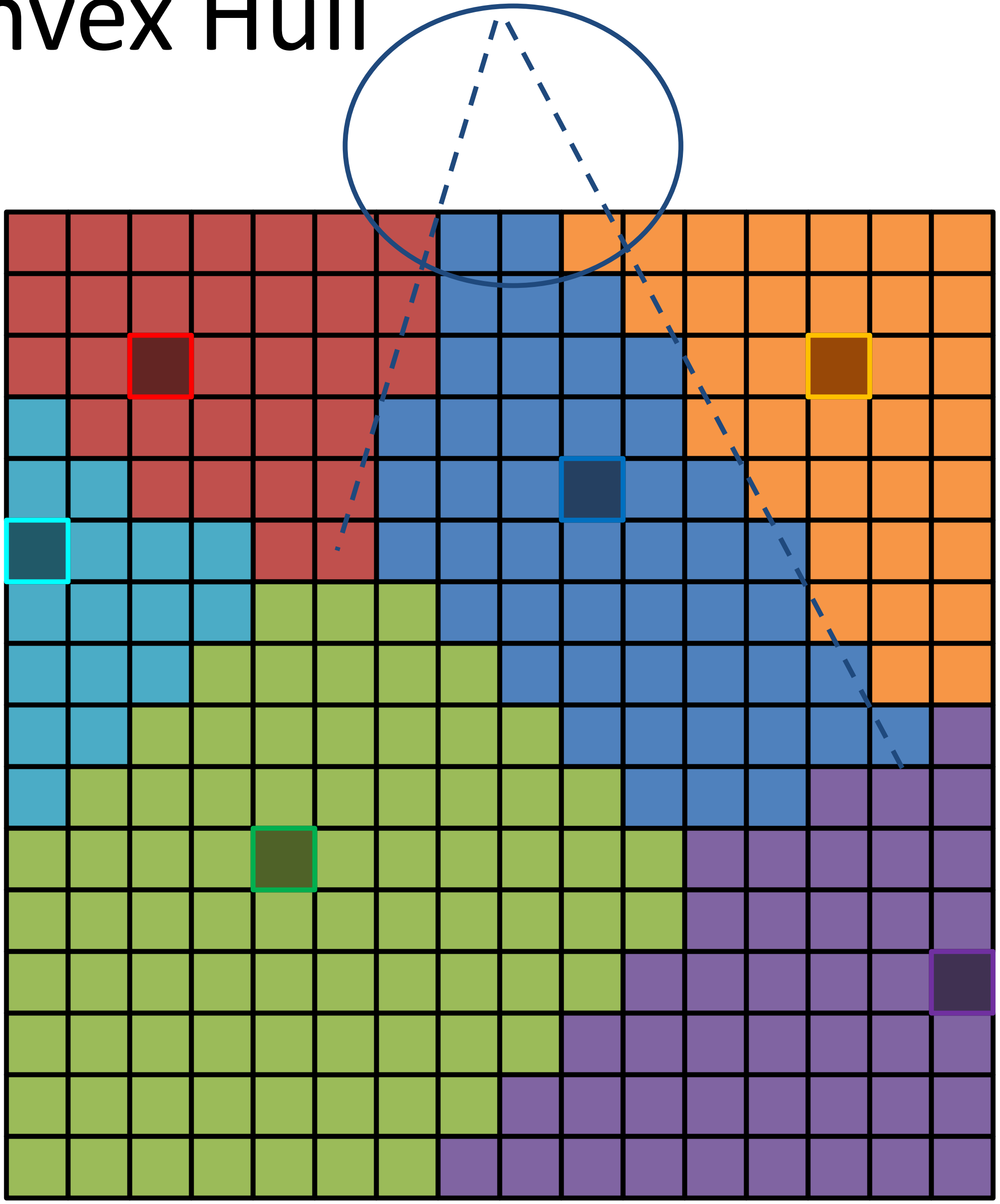
Generate Triangles



Fix Convex Hull

What's wrong here?

Vertex will not be captured in texture, but it does exist.



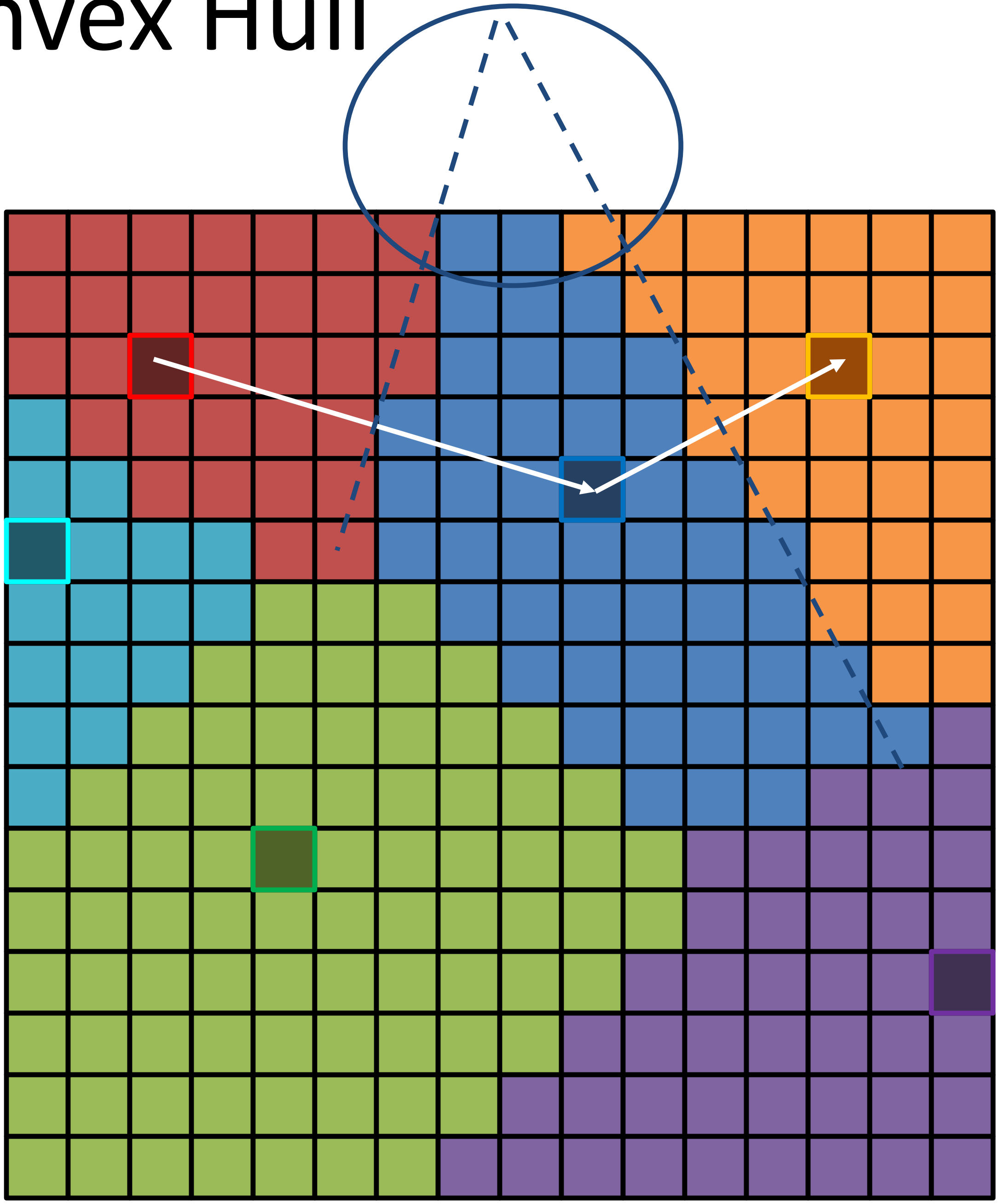
Fix Convex Hull

What's wrong here?

- Vertex will not be captured in texture, but it does exist.

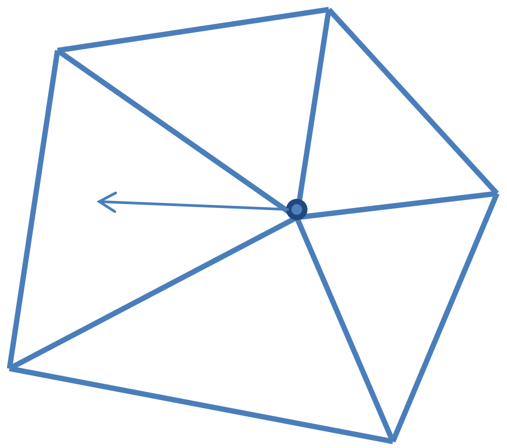
Is there a specific case where this occurs?

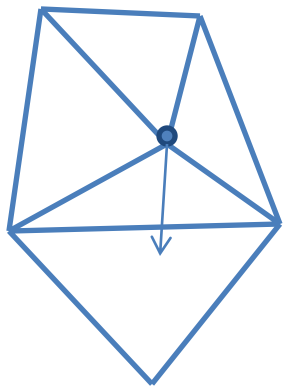
- Moving clockwise around cells who touch the border, sites that form a left turn will have a vertex outside the texture

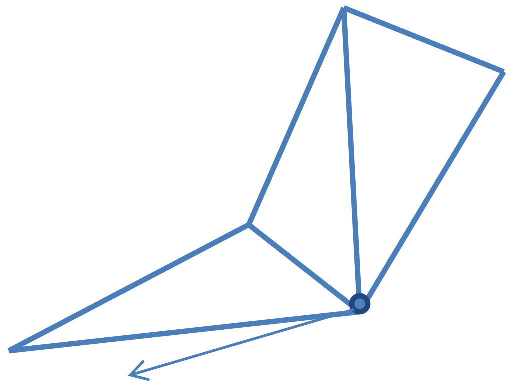


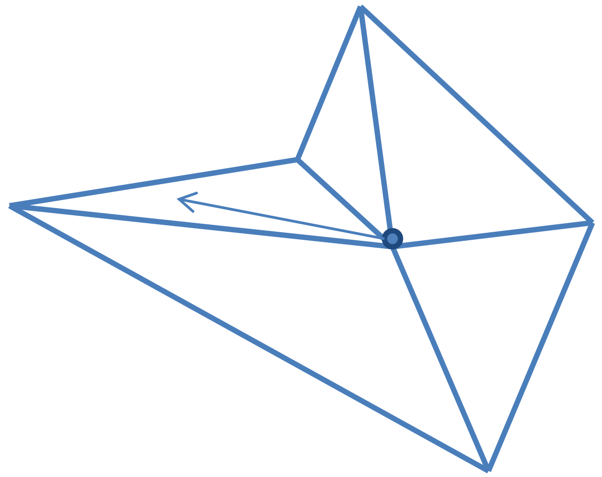
Shift Sites

We shifted all the sites into discrete space, now we have to put them back without causing any intersecting triangles

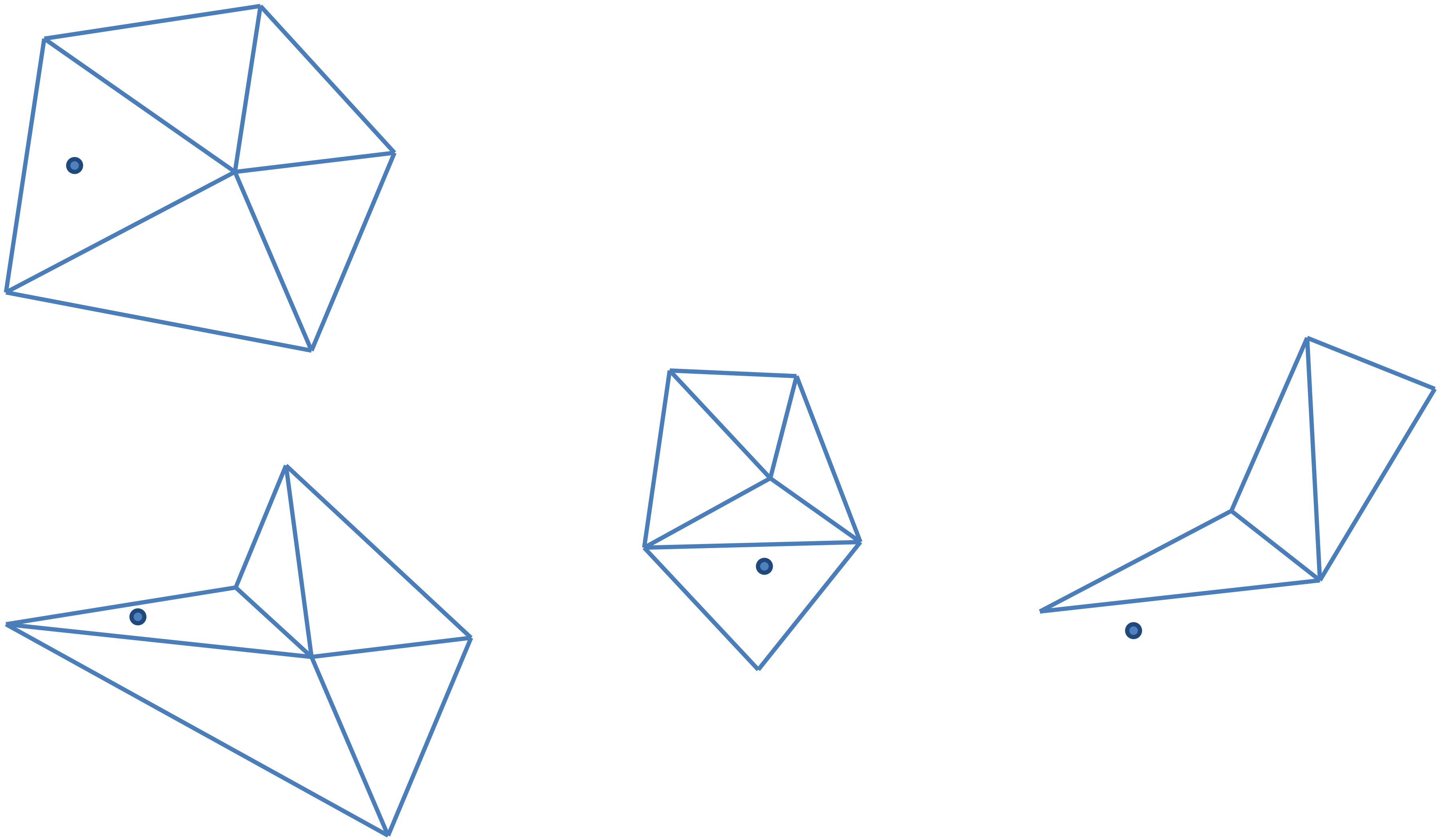




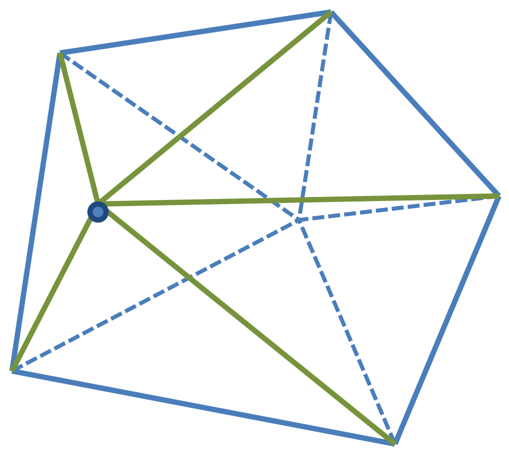




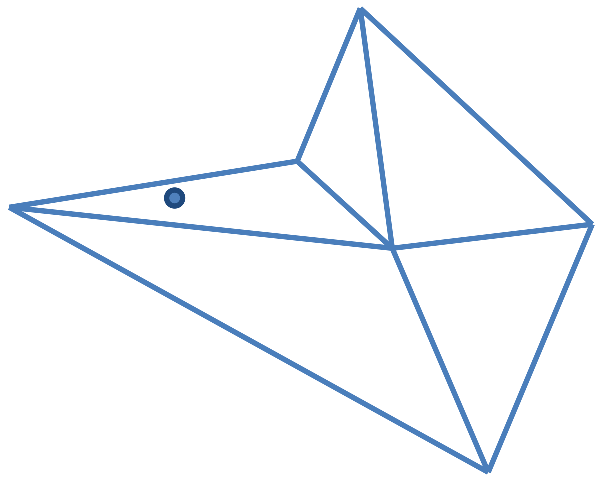
Shift Sites

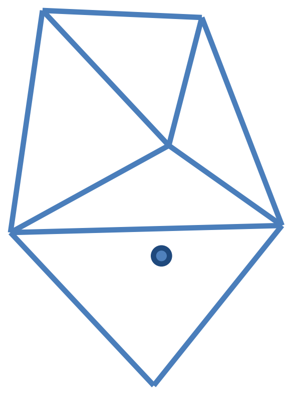


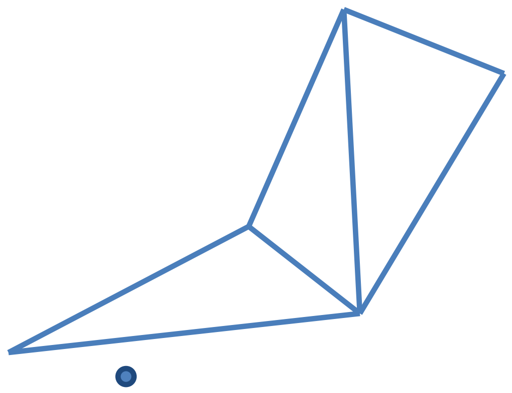
Shift Sites



1st case, we relocate to the inside of a triangle that lies within the fan without crossing any edges

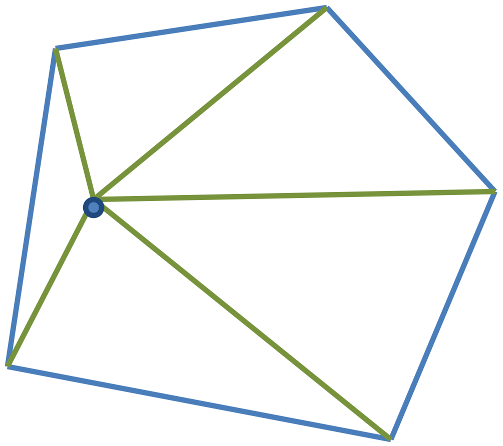


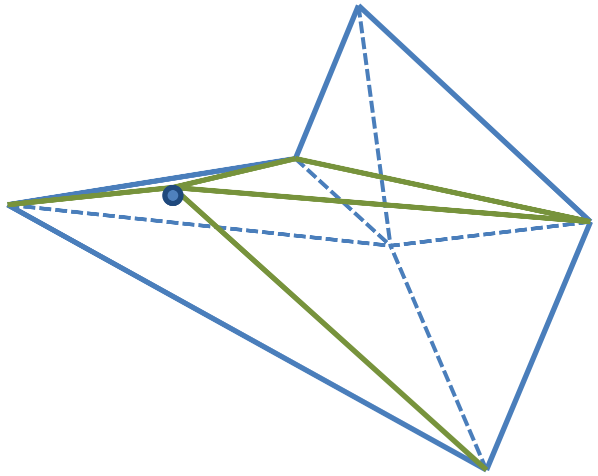


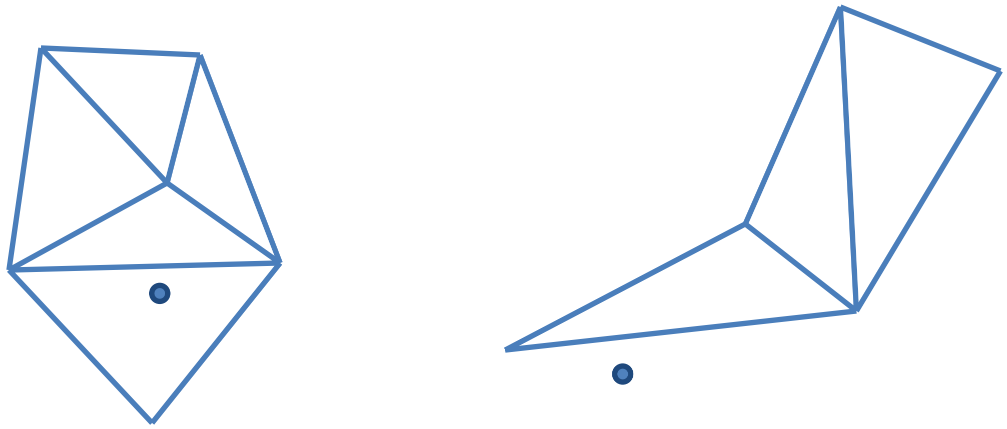


Shift Sites

2nd case, we relocate to the inside of a triangle that lies within the fan but we cause edges to cross

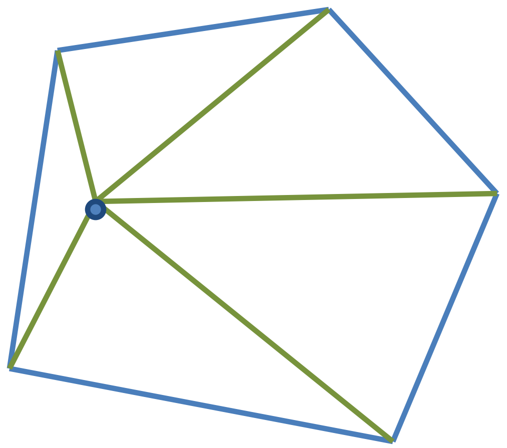


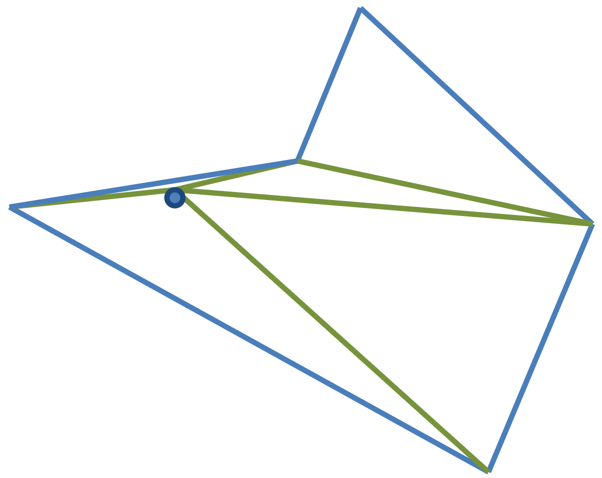


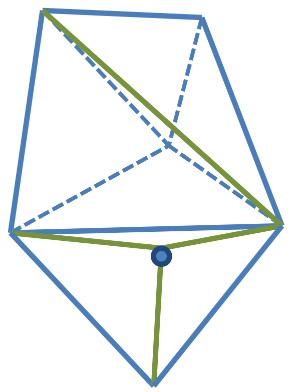


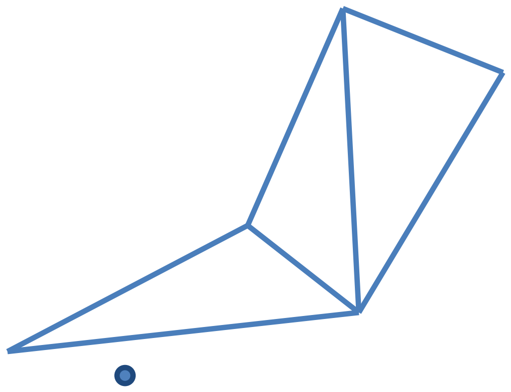
Shift Sites

3rd case, we relocate to the
inside of a triangle
outside of the triangle
fan connected to the site



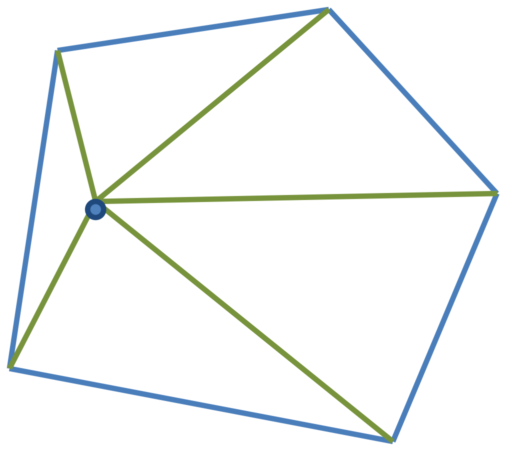
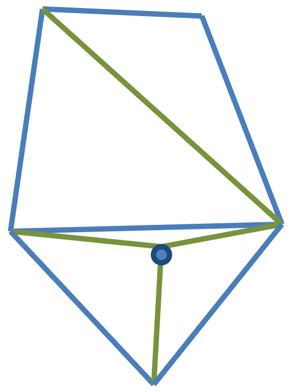
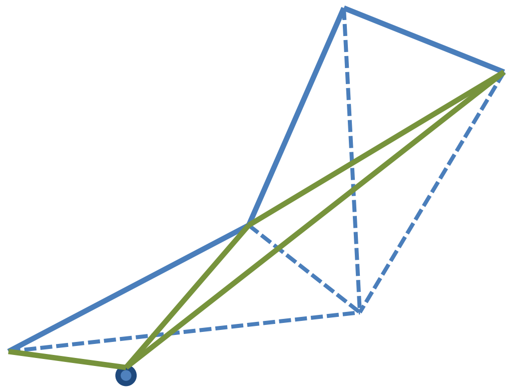
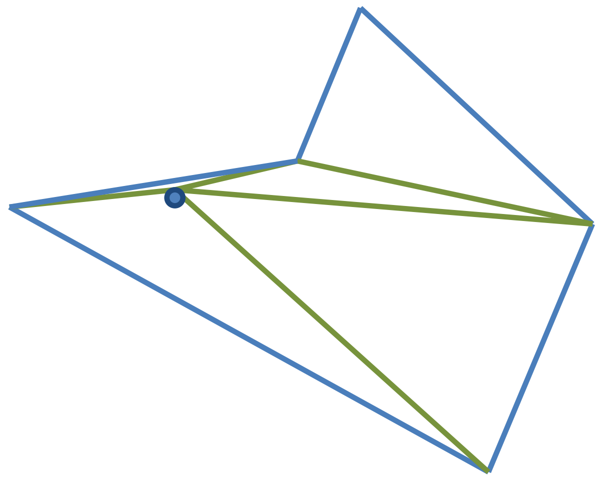




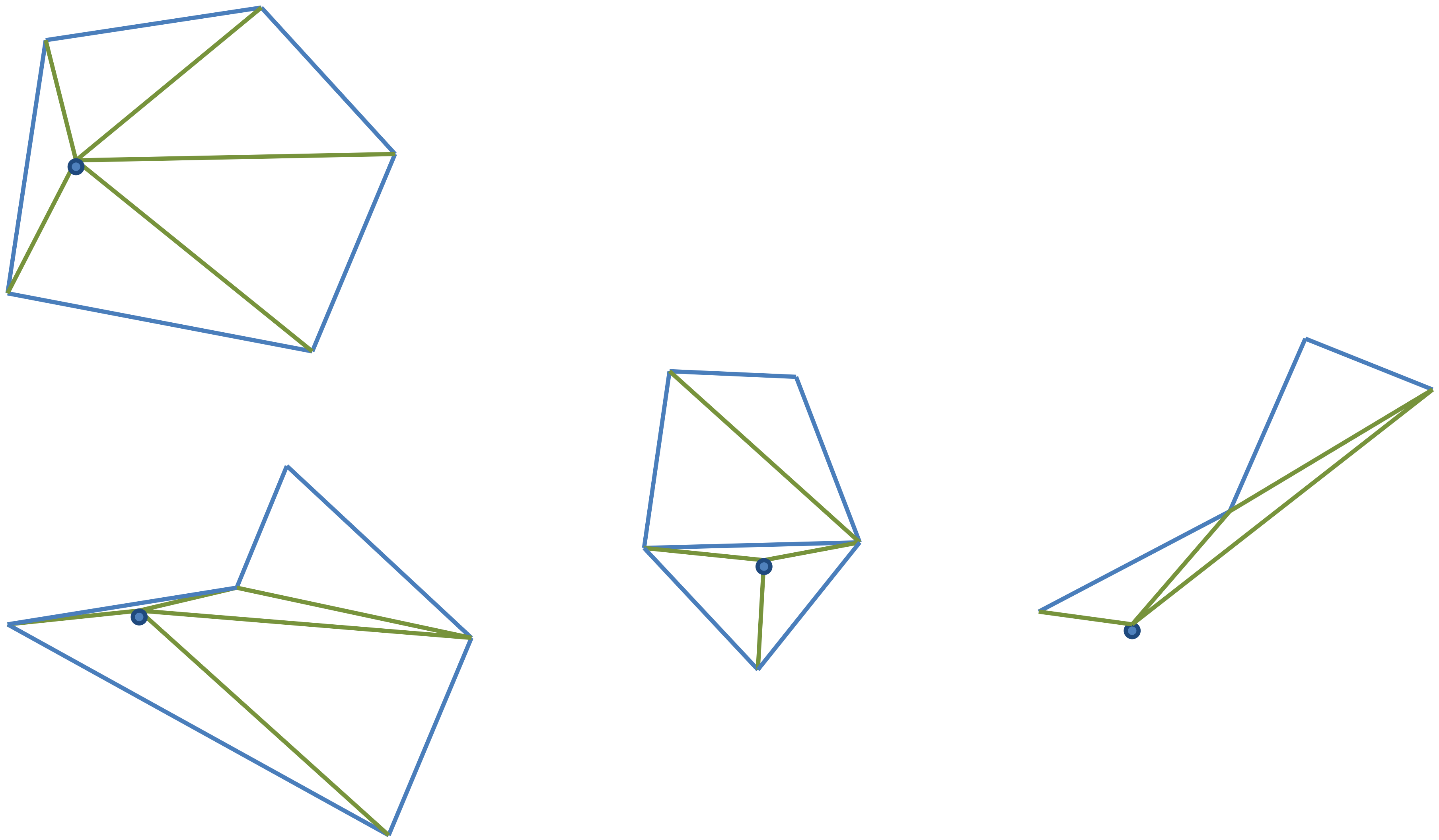


Shift Sites

4th case, we relocate outside of the mesh

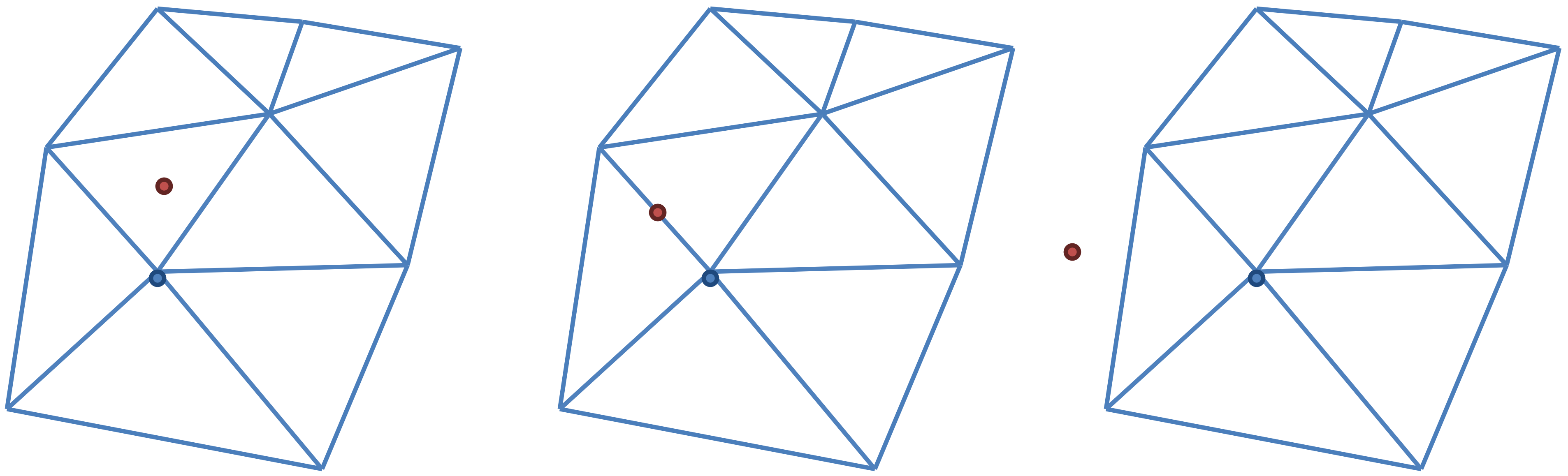





Shift Sites



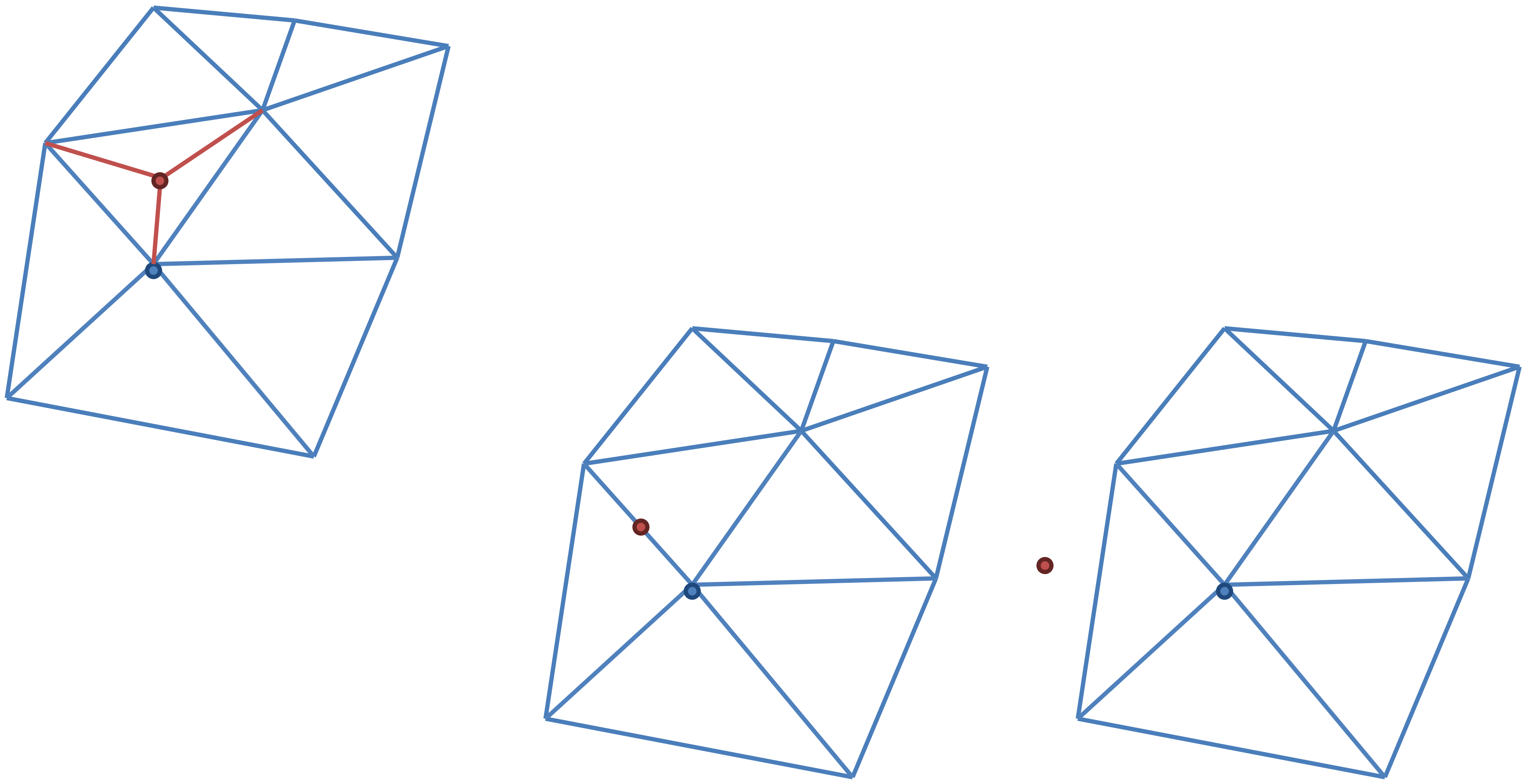
Insert Missing Sites

- Similar to what we did with the incremental approach, point locate a site in a triangle and subdivide it, but since we know who took the missing site's position, we can start with the triangle fan of that point.



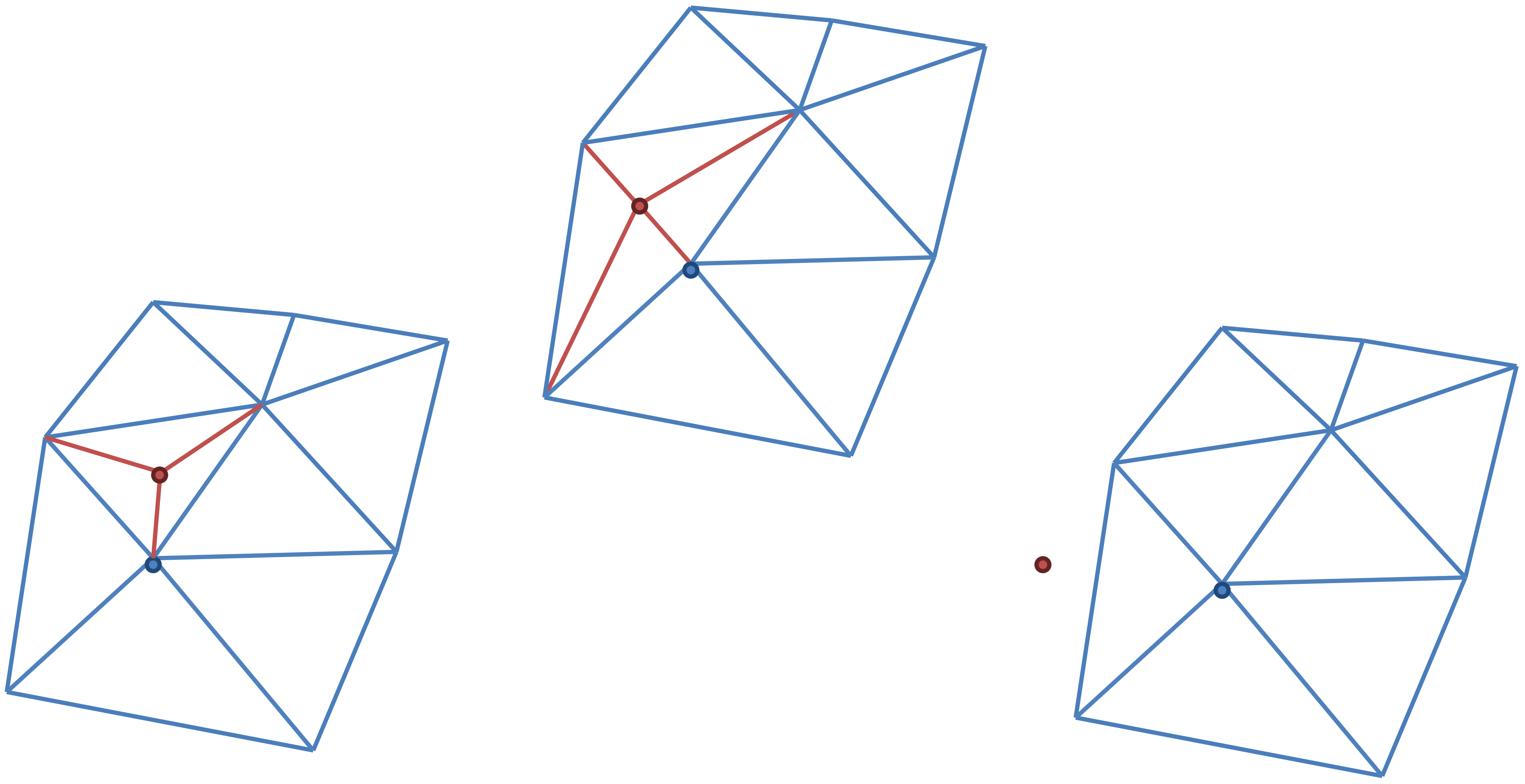
Insert Missing Sites

- 1st case, point lies in the interior of a triangle



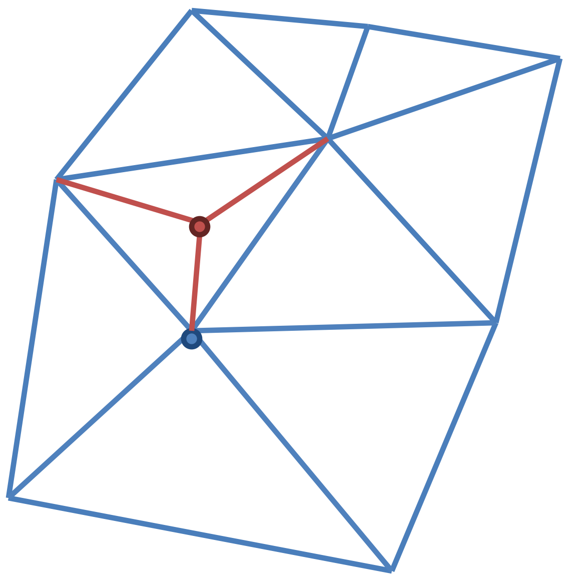
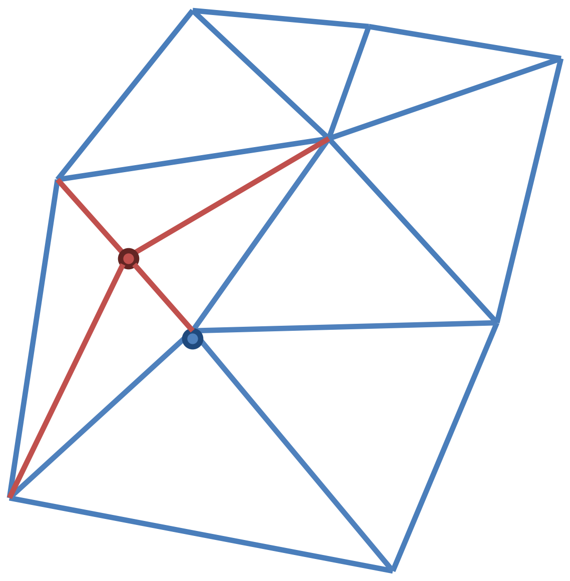
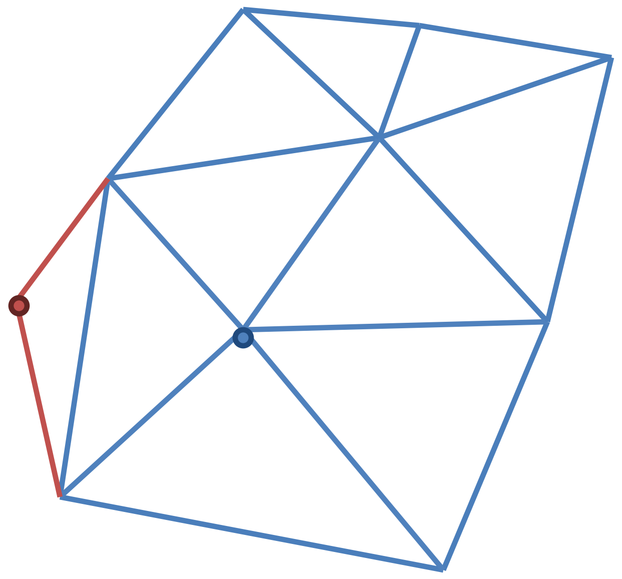
Insert Missing Sites

- 2nd case, point lies on an edge

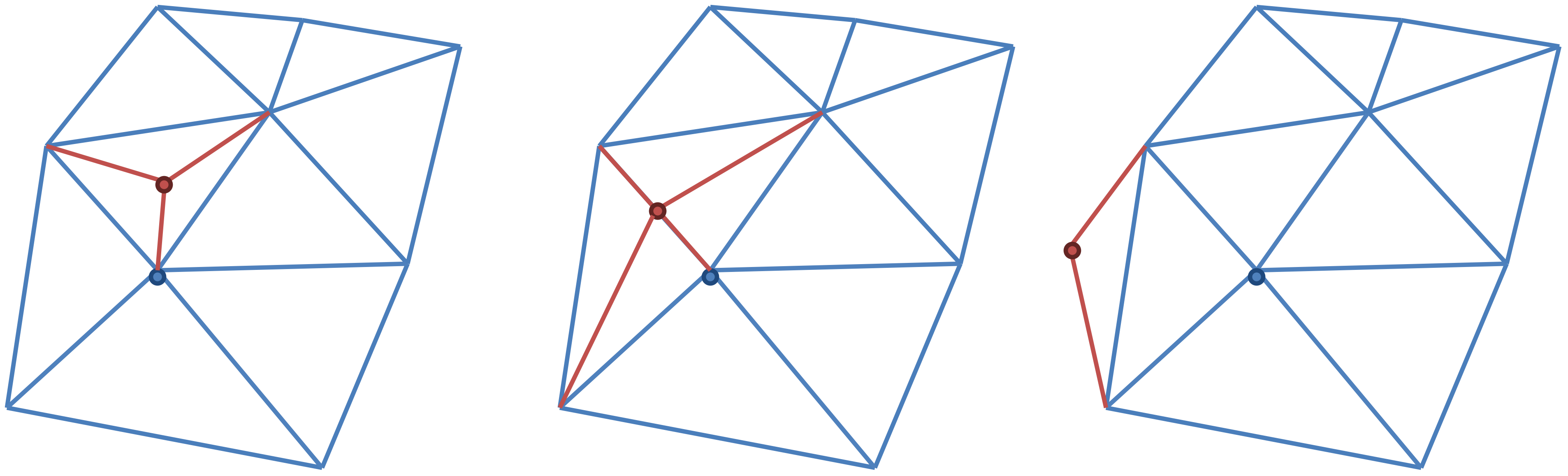


Insert Missing Sites

- 3rd case, point lies outside of the mesh

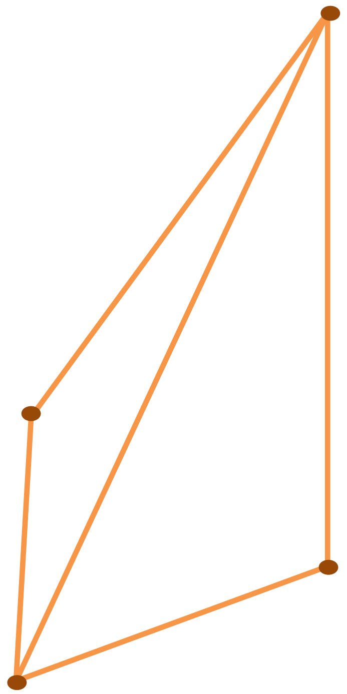
Insert Missing Sites



Flip Edges

- We have seen this before

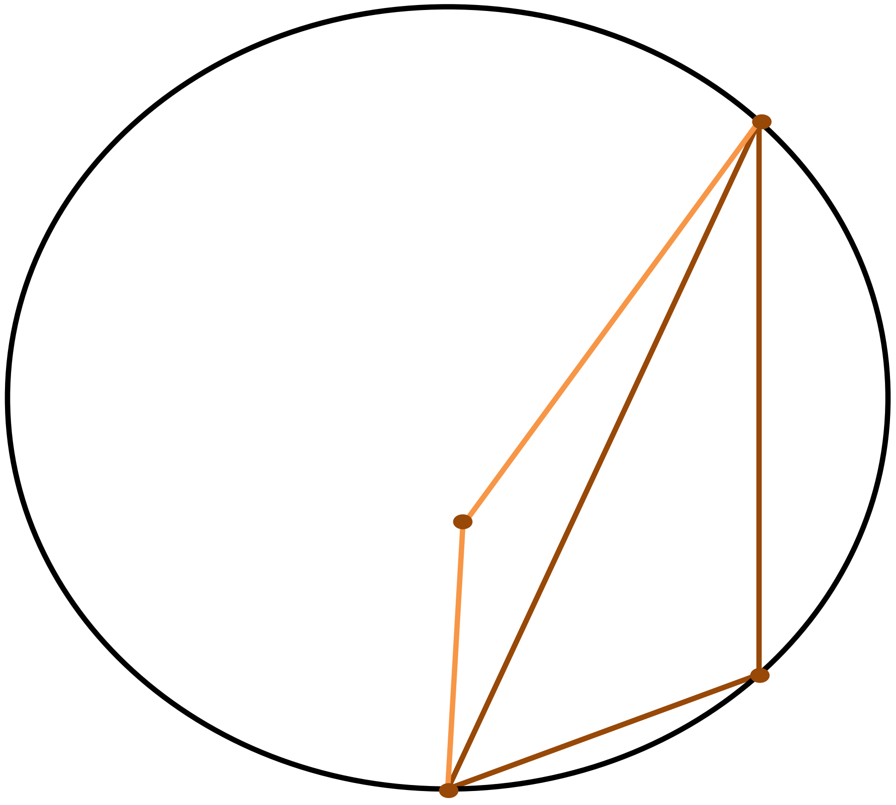
  Empty Circumcircle test



Flip Edges

- We have seen this before

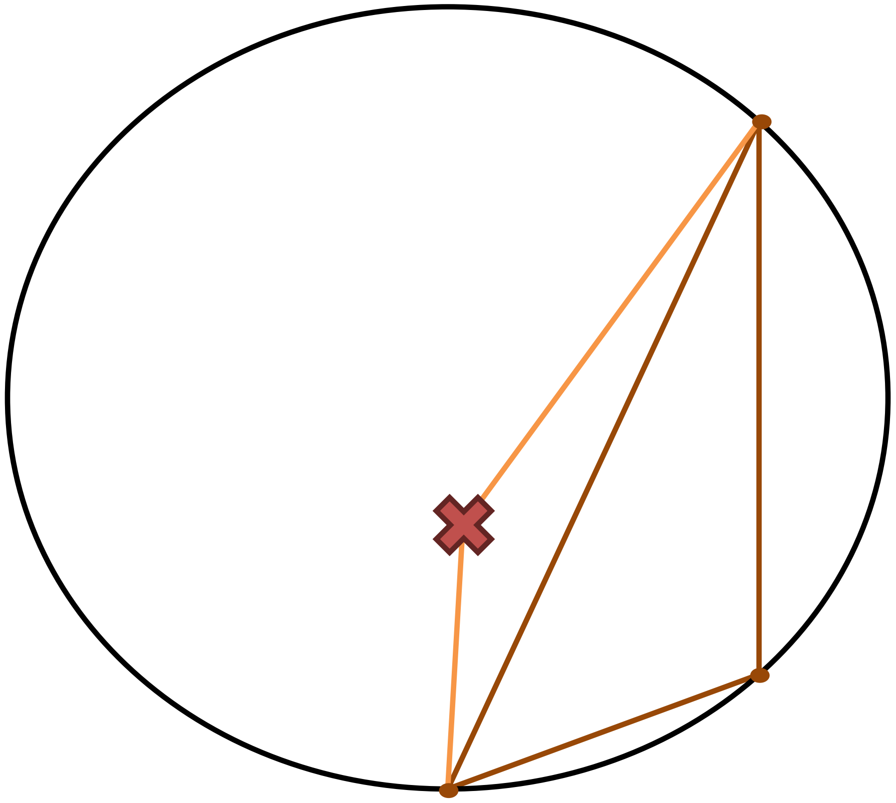
  Empty Circumcircle test



Flip Edges

- We have seen this before

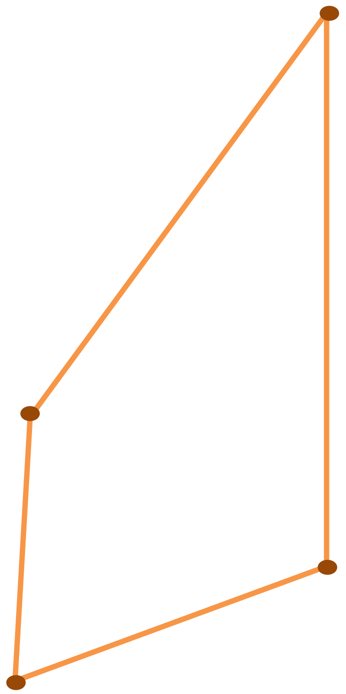
Empty Circumcircle test



Flip Edges

- We have seen this before

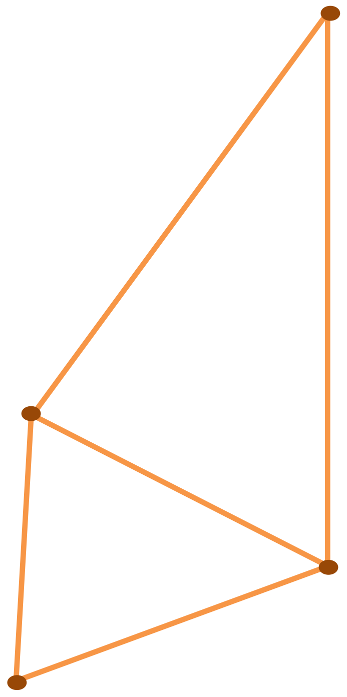
  Empty Circumcircle test



Flip Edges

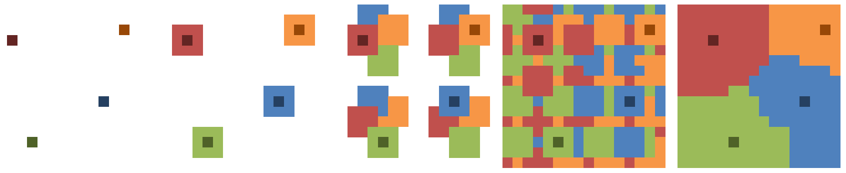
- We have seen this before

  Empty Circumcircle test

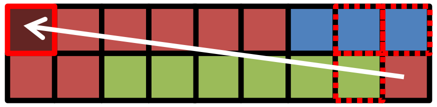


Summary

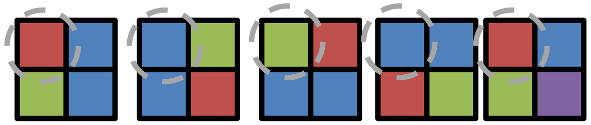
- 1+JFA

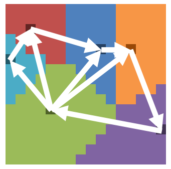


  - Island removal

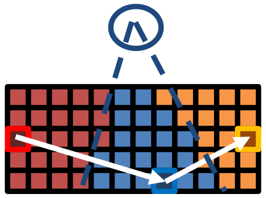


    - Find vertices

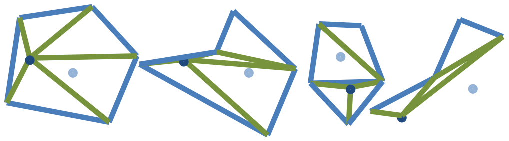




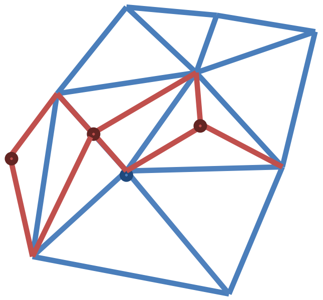
      - Generate Δs



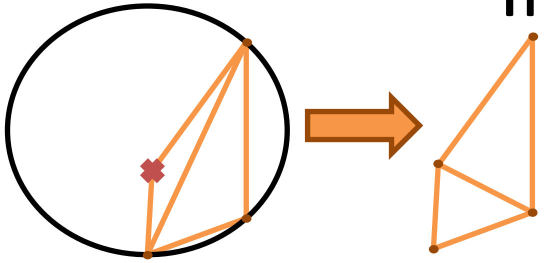
        - Fix convex hull



          - Shift sites



            - Insert missing sites



              - Flip edges

References

Hoff, K. E., Keyser, J., Lin, M., Manocha, D., Culver, T. 1999. Fast Computation of Generalized Voronoi Diagrams Using Graphics Hardware. *In Proceedings of ACM SIGGRAPH 1999, ACM Press / ACM SIGGRAPH,* New York. Computer Graphics Proceedings, Annual Conference Series, ACM, 277-286.

Denny, M. O. 2003. *Algorithmic geometry via graphics hardware. PhD thesis,* Universität des Saarlandes.

Rong, G., Tan, T.-S. 2006. Jump flooding in GPU with applications to Voronoi diagram and distance transform. In *Proceedings of the Symposium on Interactive 3D Graphics and Games, ACM Press, 109–116.*

Rong, G., Tan, T.-S. 2007. Variants of jump flooding algorithm for computing discrete Voronoi diagrams. In *Proceedings of the 4th International Symposium on Voronoi Diagrams in Science and Engineering (ISVD'07),* 176–181.

Rong, G., Tan, T.-S., and Cao, T.-T. and Stephanus. 2008. Computing Two dimensional Delaunay Triangulation Using Graphics Hardware. In *Proceedings of the Symposium on Interactive 3D Graphics and Games,* ACM Press, 89–97.