Efficient Snap Rounding with Integer Arithmetic

Binay K. Bhattacharya
Jeff Sember

Simon Fraser University

August 21, 2007

Outline

Outline

Outline

Outline

Outline

Outline

Outline

The Problem

- finding intersections of line segments

The Problem

- finding intersections of line segments
- often assumes exact real arithmetic

The Problem

- finding intersections of line segments
- often assumes exact real arithmetic
- impractical: i.e. displaying on limited-resolution displays

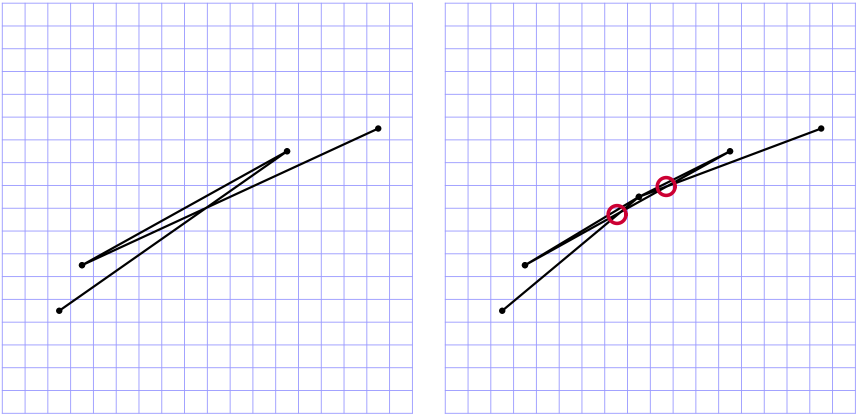
The Problem

- finding intersections of line segments
- often assumes exact real arithmetic
- impractical: i.e. displaying on limited-resolution displays
- approach: round endpoints, intersection points to integer grid

The Problem

- finding intersections of line segments
- often assumes exact real arithmetic
- impractical: i.e. displaying on limited-resolution displays
- approach: round endpoints, intersection points to integer grid
- problem: extraneous intersections

The Problem



Hobby (1996)

The Problem

- Solution: Snap Rounding

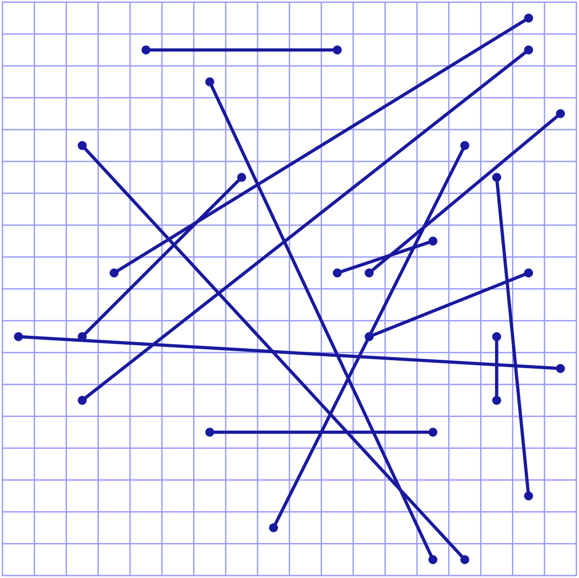
The Problem

- Solution: Snap Rounding
- Every pixel containing endpoint or intersection point is hot

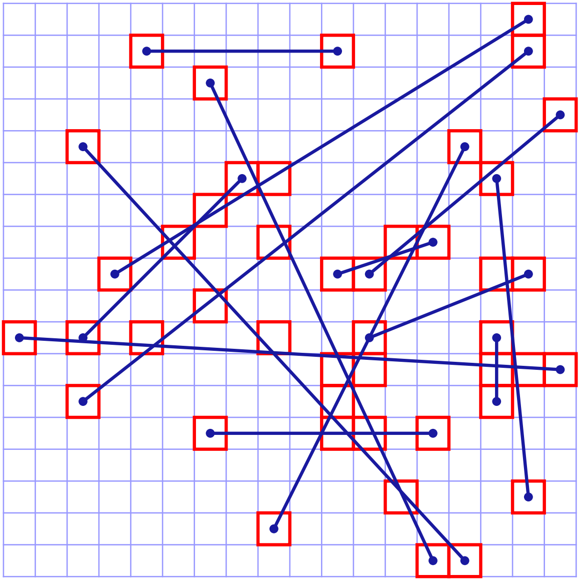
The Problem

- Solution: Snap Rounding
- Every pixel containing endpoint or intersection point is hot
- Every segment intersecting hot pixel is rerouted through that pixel's center

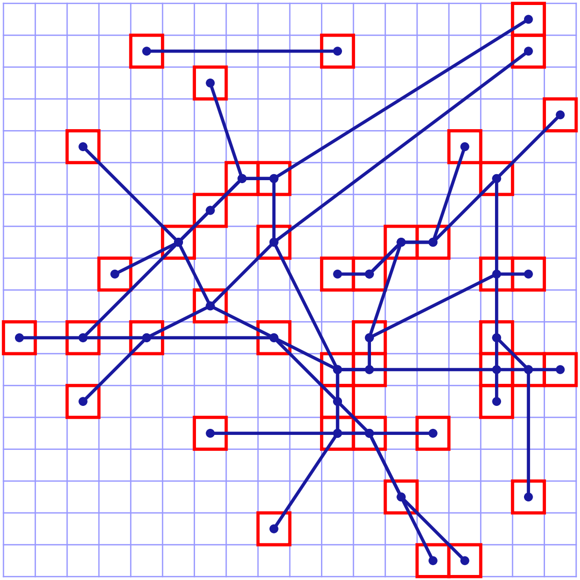
The Problem



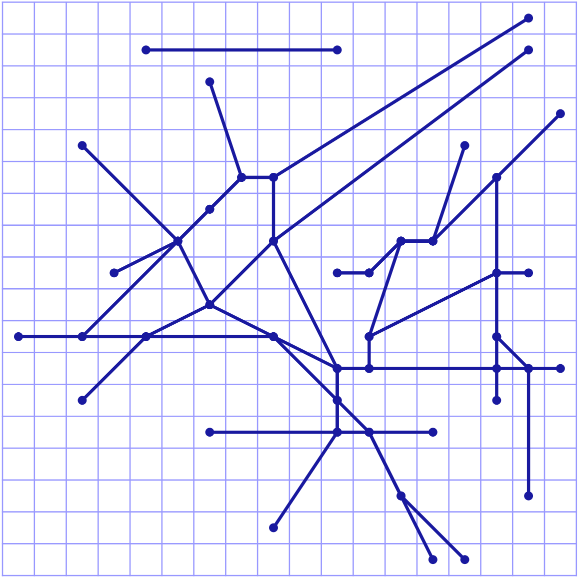
The Problem



The Problem



The Problem



Previous Results

- Introduced independently by Hobby (1996), Greene

Previous Results

- Introduced independently by Hobby (1996), Greene
- $O(n \log n + \Sigma_{h \in H} |h| \log n)$ Goodrich et al (1997)

Previous Results

- Introduced independently by Hobby (1996), Greene
- $O(n \log n + \Sigma_{h \in H} |h| \log n)$ Goodrich et al (1997)
- $O((n + |I|) \log n)$ de Berg et al (2007)

Previous Results

- Introduced independently by Hobby (1996), Greene
- $O(n \log n + \Sigma_{h \in H} |h| \log n)$ Goodrich et al (1997)
- $O((n + |I|) \log n)$ de Berg et al (2007)
- $O(\Sigma_{h \in H} is(h) \log n)$, $O(\Sigma_{h \in H} ed(h) \log n)$ Hershberger (2006)

Previous Results

- Introduced independently by Hobby (1996), Greene
- $O(n \log n + \Sigma_{h \in H} |h| \log n)$ Goodrich et al (1997)
- $O((n + |I|) \log n)$ de Berg et al (2007)
- $O(\Sigma_{h \in H} is(h) \log n)$, $O(\Sigma_{h \in H} ed(h) \log n)$ Hershberger (2006)
- all rely on high precision computation (i.e., clipping segments to pixel boundaries)

Our Results

- first algorithm: generates set of snapped segments in $O(|I| + \Sigma_c \, is(c) \log n + |I_m^*|)$ time

Our Results

- first algorithm: generates set of snapped segments in $O(|I| + \Sigma_c\, is(c) \log n + |I_m^*|)$ time
- second algorithm: generates rounded arrangement of segments in $O(|I| + \Sigma_c\, is(c) \log n + |I^*| \log n)$ time

Algorithm One

- uses modified Bentley & Ottman (1979) plane sweep

Algorithm One

- uses modified Bentley & Ottman (1979) plane sweep
- sweep line is actually a column of half pixels (to process endpoints cleanly)

Algorithm One

- uses modified Bentley & Ottman (1979) plane sweep
- sweep line is actually a column of half pixels (to process endpoints cleanly)
- active list, intersection events stored in a modified B+ tree

Algorithm One

- uses modified Bentley & Ottman (1979) plane sweep
- sweep line is actually a column of half pixels (to process endpoints cleanly)
- active list, intersection events stored in a modified B+ tree
- endpoint events stored in separate priority queue

Algorithm One

- each segment with endpoint or intersection point in column is associated with a set of hot pixels

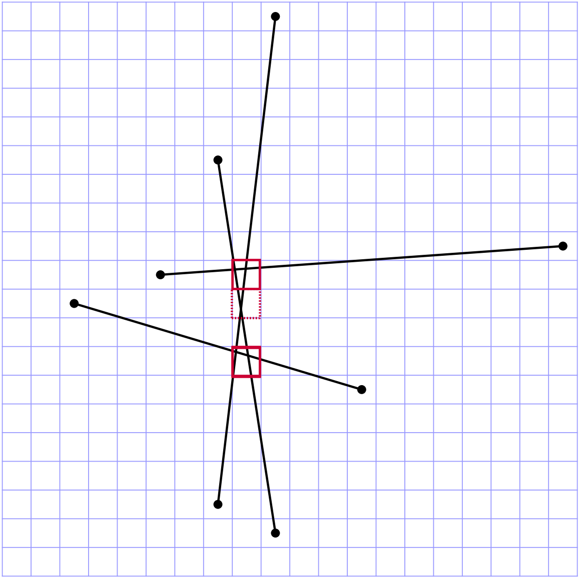
Algorithm One

- each segment with endpoint or intersection point in column is associated with a set of hot pixels
- modify snap rounding definition slightly

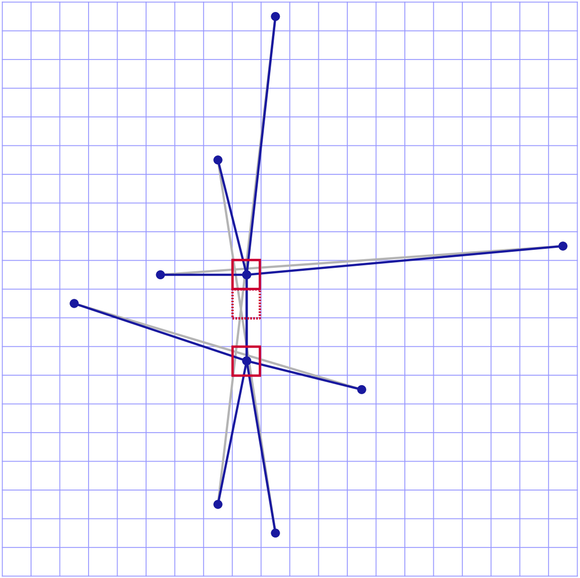
Algorithm One

- each segment with endpoint or intersection point in column is associated with a set of hot pixels
- modify snap rounding definition slightly
- hot pixel set can now be represented as hotRange: a lowest and highest hot pixel for the segment

Proof of Lemma 1

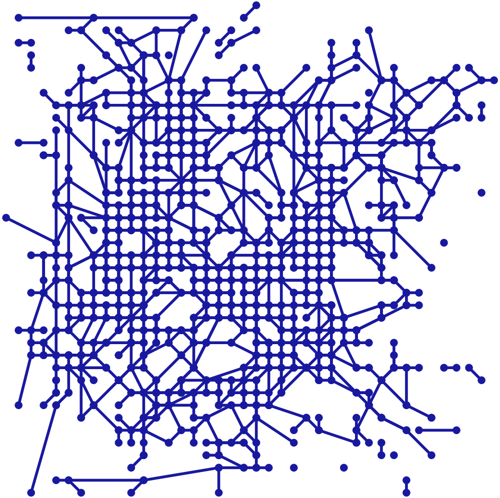


Proof of Lemma 1



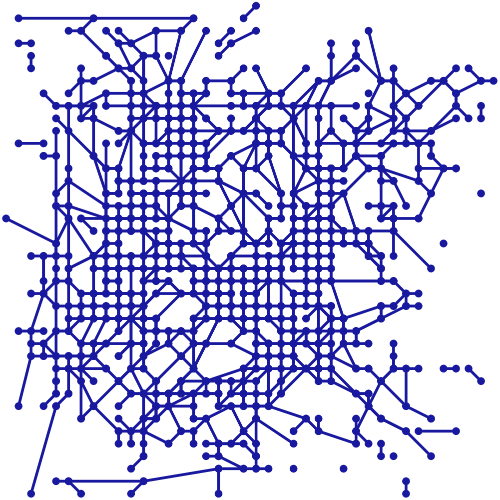
Modified Definition

An example snapped arrangement using the original definition:



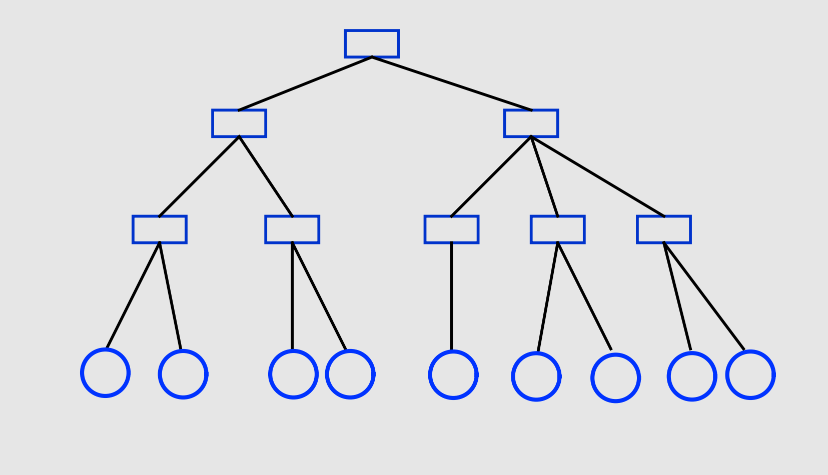
Modified Definition

The same arrangement using our definition:



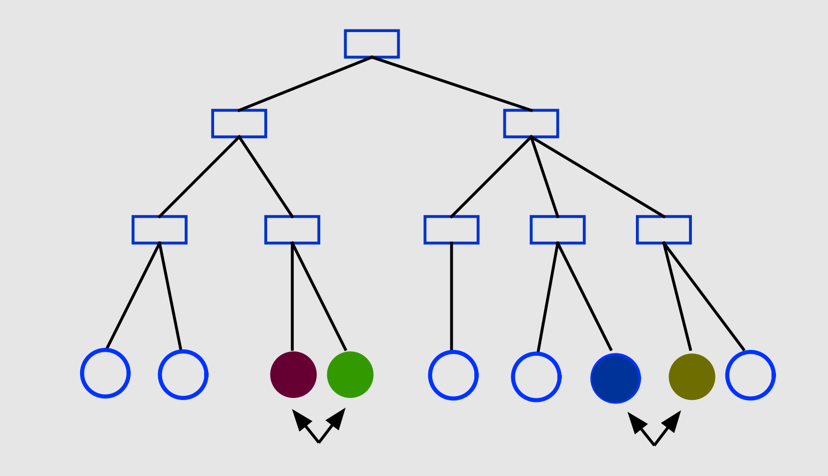
Algorithm One

Active list is leaf level of tree



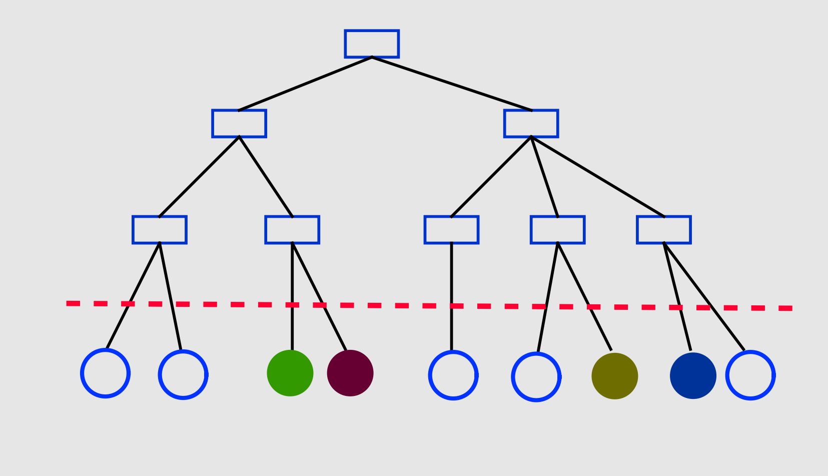
Algorithm One

Detect seed events: neighboring segments intersecting within column



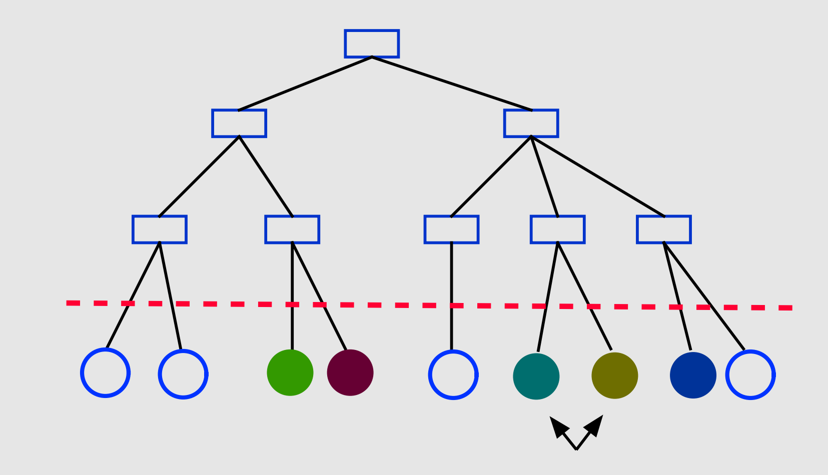
Algorithm One

Manipulate active list, ignore rest of tree data structure



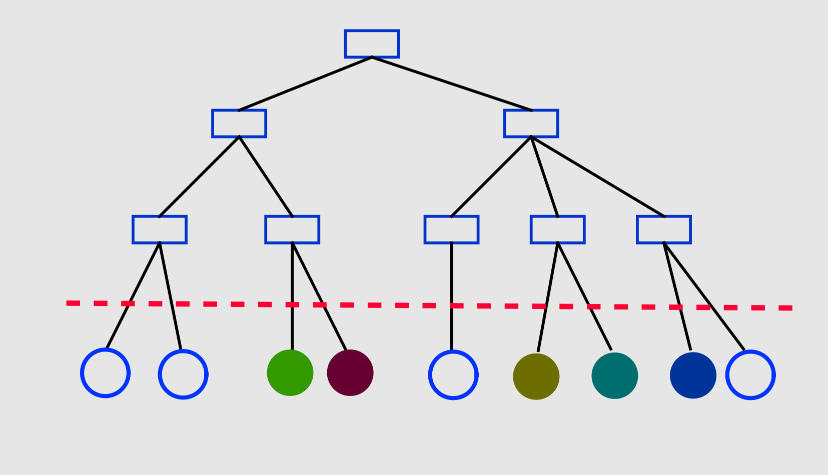
Algorithm One

Detect additional intersection events



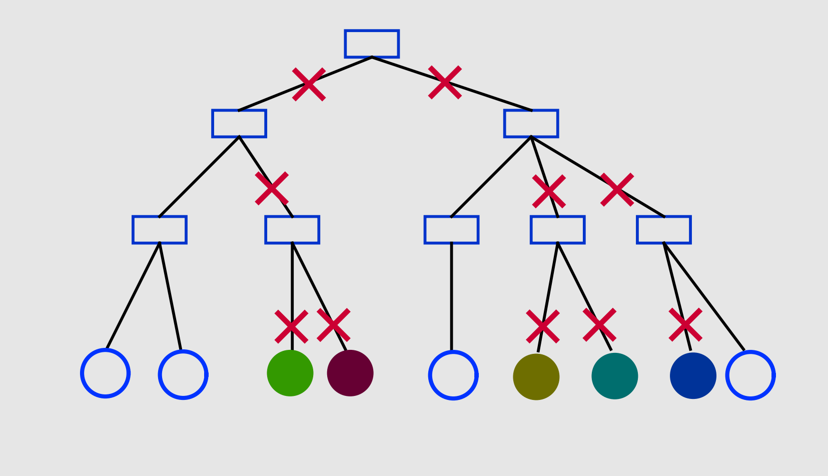
Algorithm One

Continue until order stabilizes



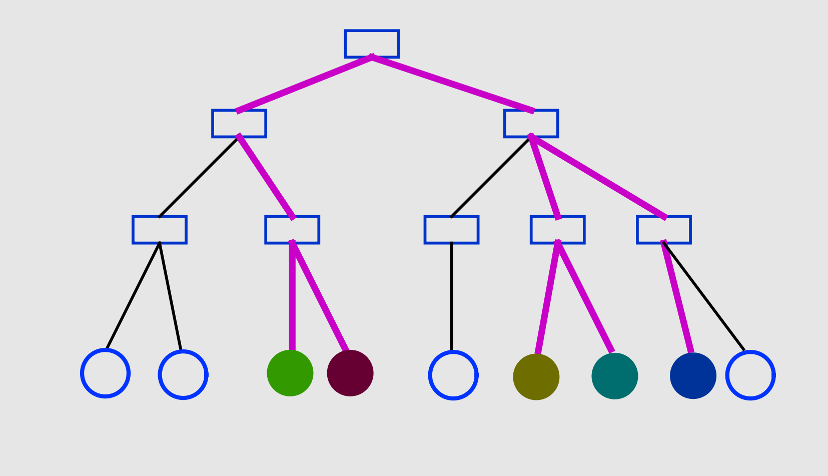
Algorithm One

Recalculate internal paths from moved segments



Algorithm One

Tree now ready for right side of sweep column



Algorithm One

At each sweep column position, three processes:

- sweep process

Algorithm One

At each sweep column position, three processes:

- sweep process
- hot pixel process

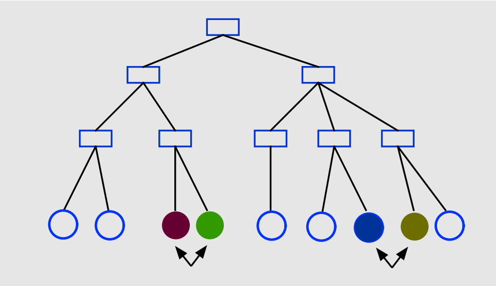
Algorithm One

At each sweep column position, three processes:

- sweep process
- hot pixel process
- snap process

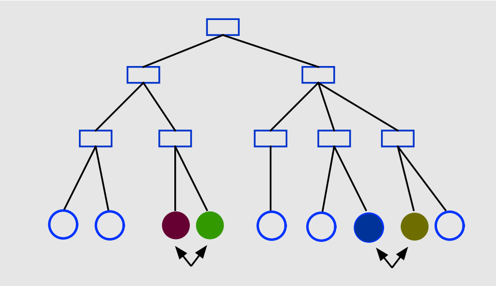
Sweep Process

- query tree for seed events



Sweep Process

- query tree for seed events



- push seed events onto a stack

Sweep Process

While seed event stack is not empty:

- pop event

Sweep Process

While seed event stack is not empty:

- pop event
- add pixel containing intersection to each segment's hotRange

Sweep Process

While seed event stack is not empty:

- pop event
- add pixel containing intersection to each segment's hotRange
- exchange position of segments within active list

Sweep Process

While seed event stack is not empty:

- pop event
- add pixel containing intersection to each segment's hotRange
- exchange position of segments within active list
- add segments, old, and new neighbors to snapSet

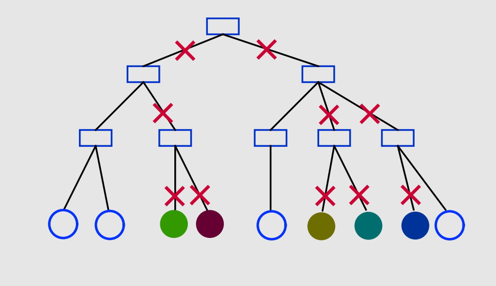
Sweep Process

While seed event stack is not empty:

- pop event
- add pixel containing intersection to each segment's hotRange
- exchange position of segments within active list
- add segments, old, and new neighbors to snapSet
- test if segments intersect new neighbors within column; if so, push event onto stack

Sweep Process

- recalculate internal paths for snapSet segments



Sweep Process

Details omitted:

- starting/stopping segments

Sweep Process

Details omitted:

- starting/stopping segments
- dealing with vertical or zero-length segments

Hot Pixel Process

- examine set of all segments that were involved in seed events

Hot Pixel Process

- examine set of all segments that were involved in seed events
- construct sorted, linked list of distinct hot pixels from hotRange fields

Snap Process

- test every segment in snapSet

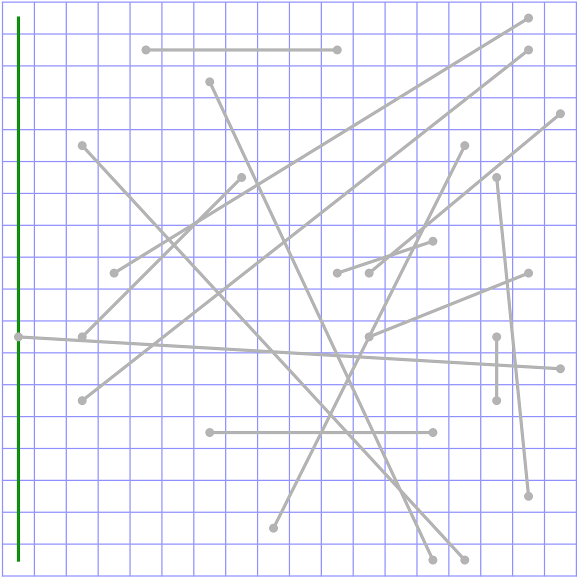
Snap Process

- test every segment in snapSet
- snap segment to list of hot pixels extracted in previous process

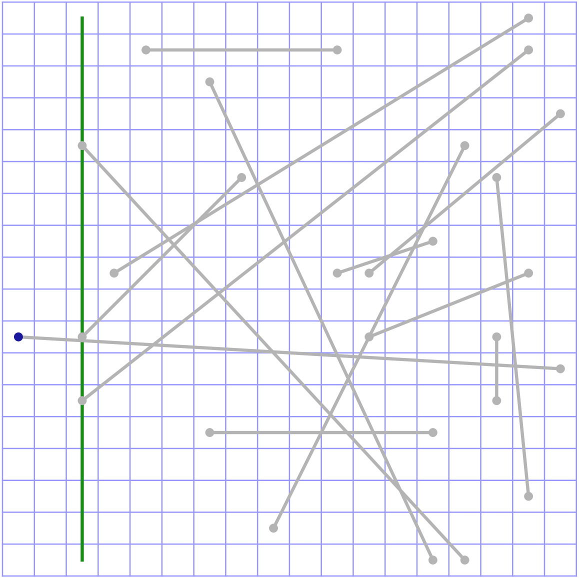
Snap Process

- test every segment in snapSet
- snap segment to list of hot pixels extracted in previous process
- careful use of pointers and sorted list of hot pixels allows this to be done in $O(1)$ time for each snapped fragment

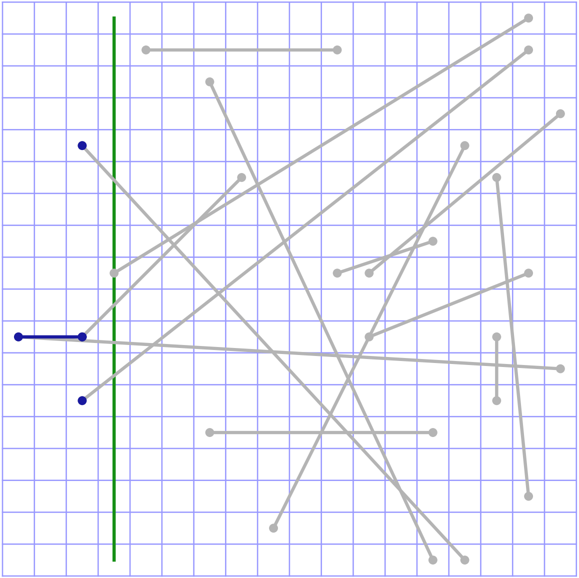
Algorithm One



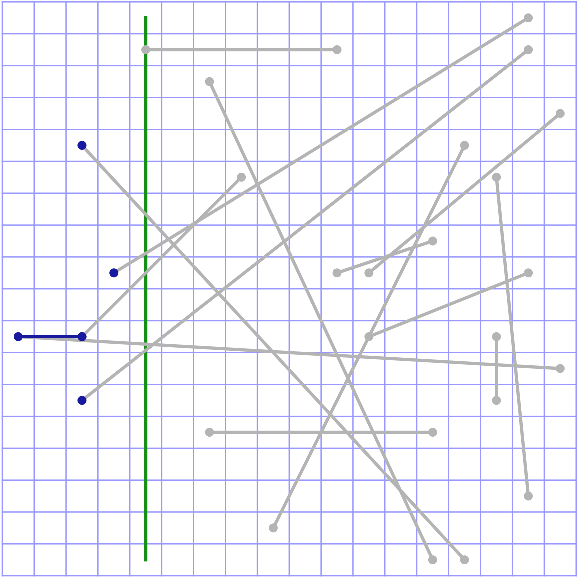
Algorithm One



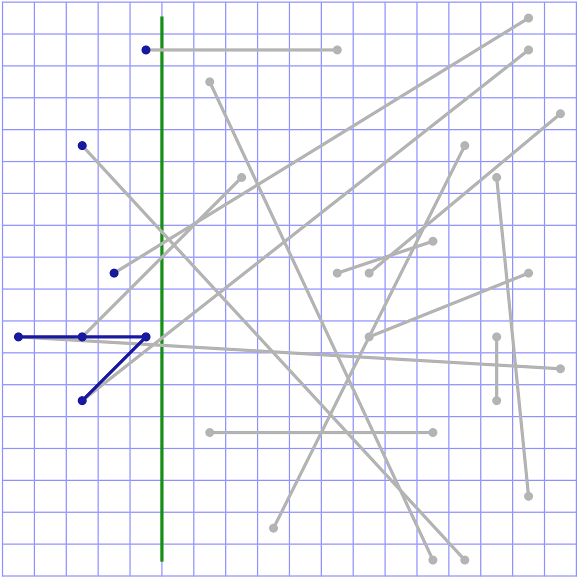
Algorithm One



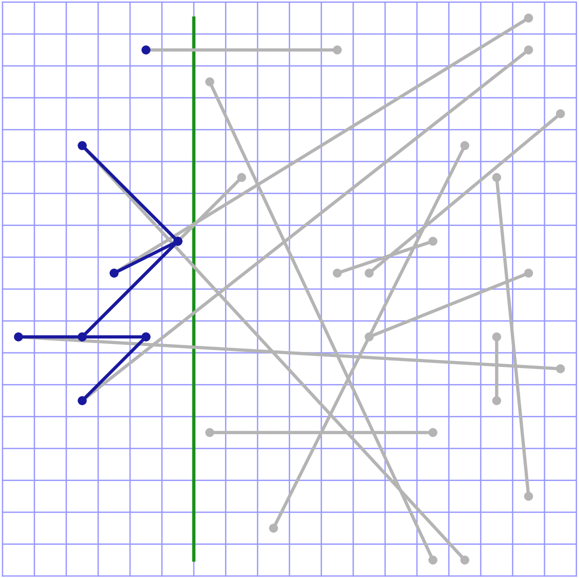
Algorithm One



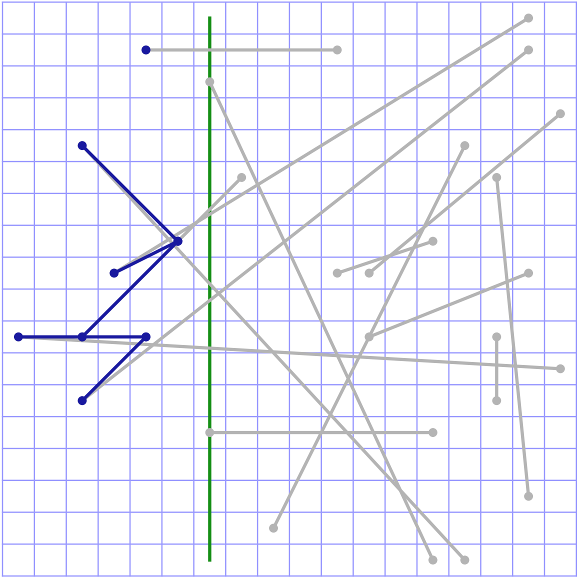
Algorithm One



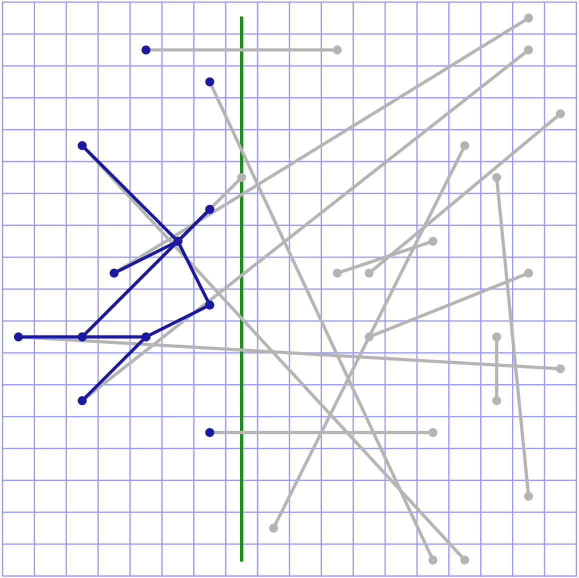
Algorithm One



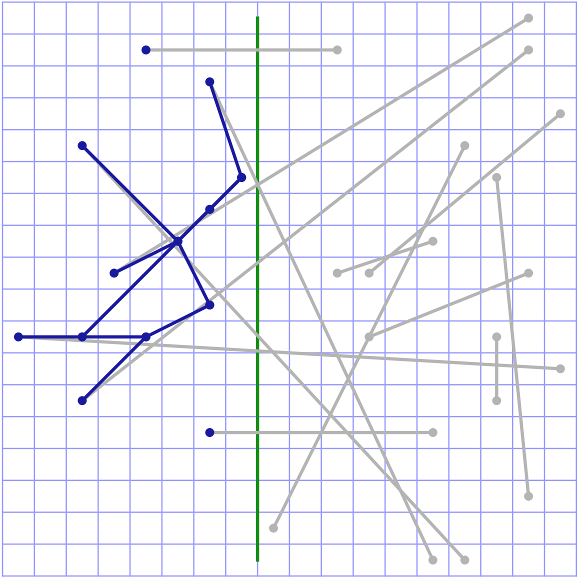
Algorithm One



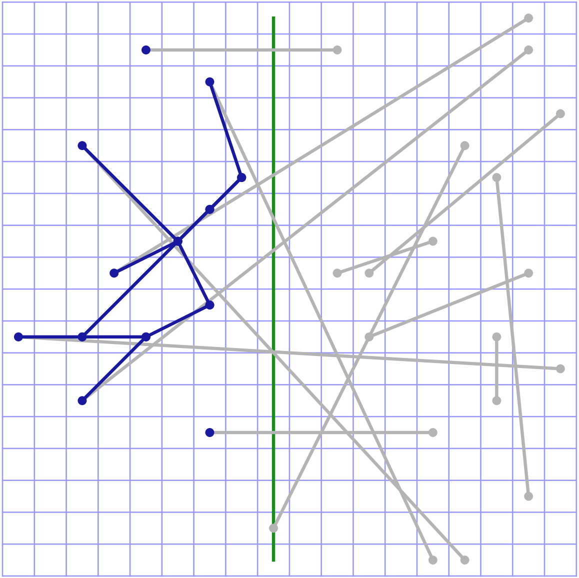
Algorithm One



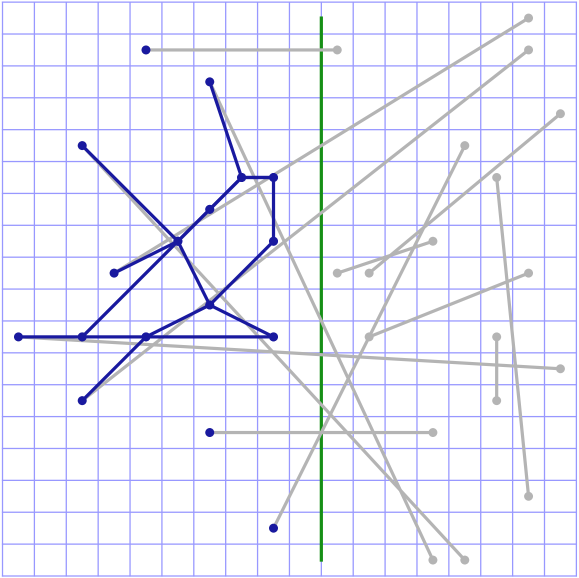
Algorithm One



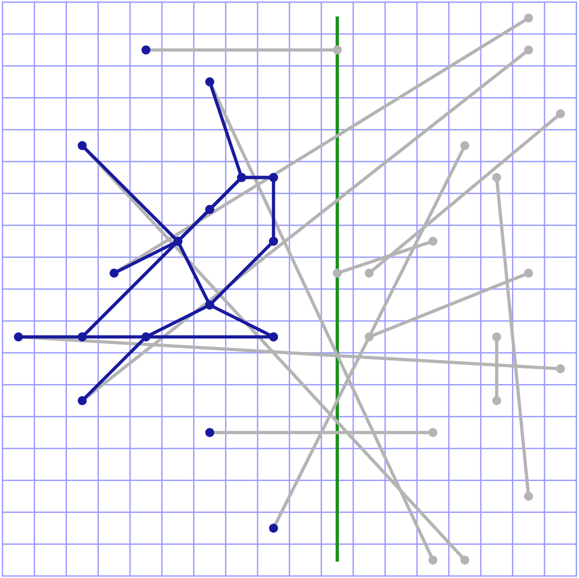
Algorithm One



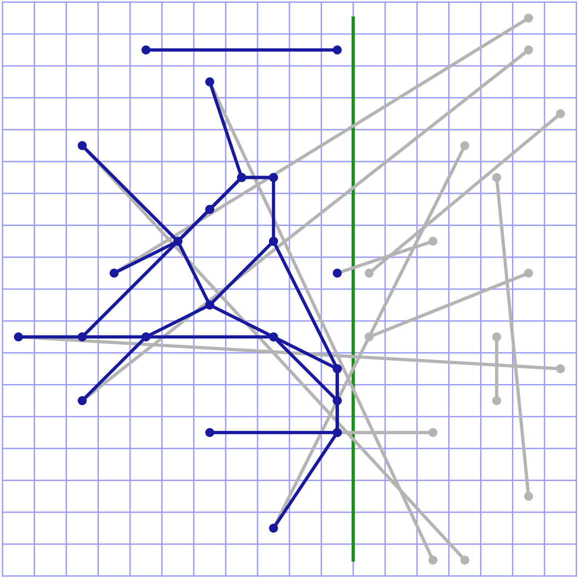
Algorithm One



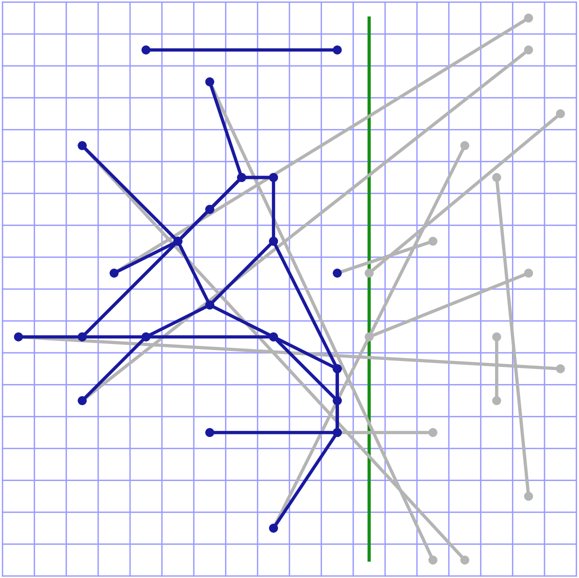
Algorithm One



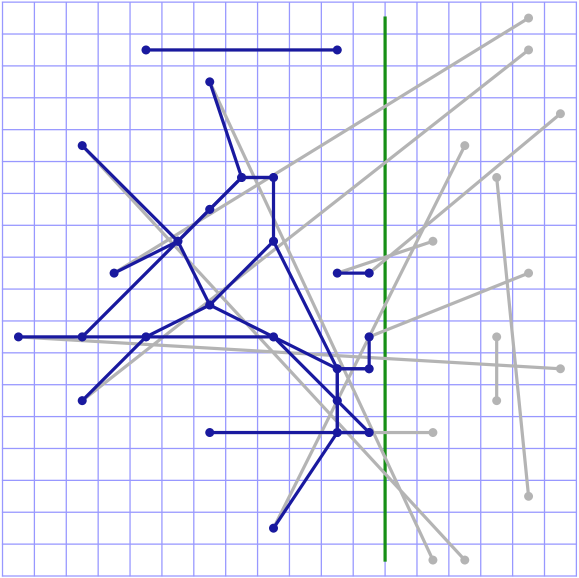
Algorithm One



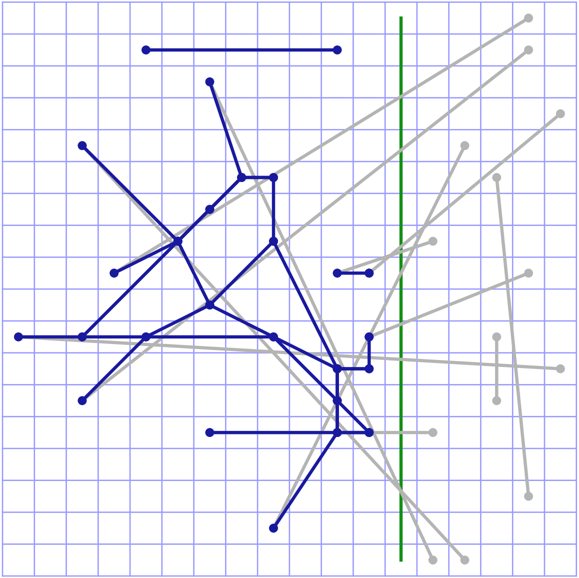
Algorithm One



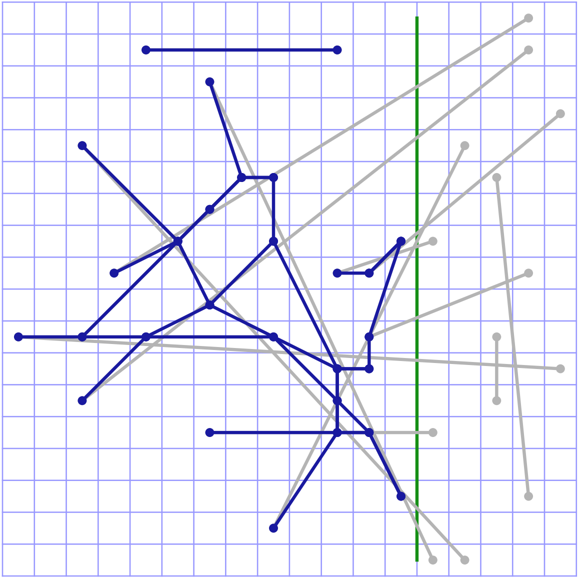
Algorithm One



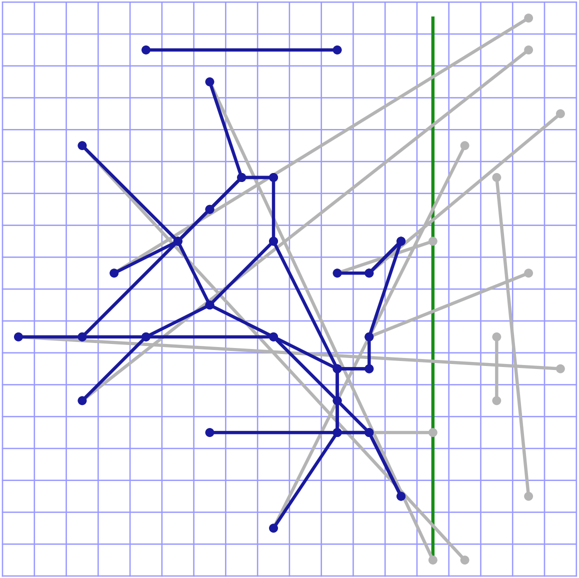
Algorithm One



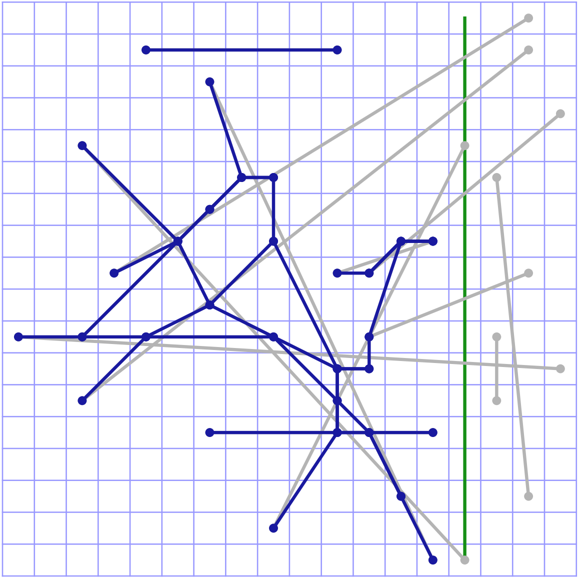
Algorithm One



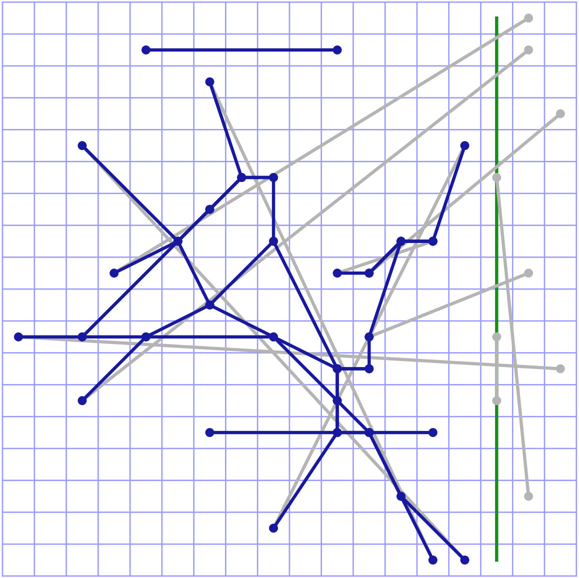
Algorithm One



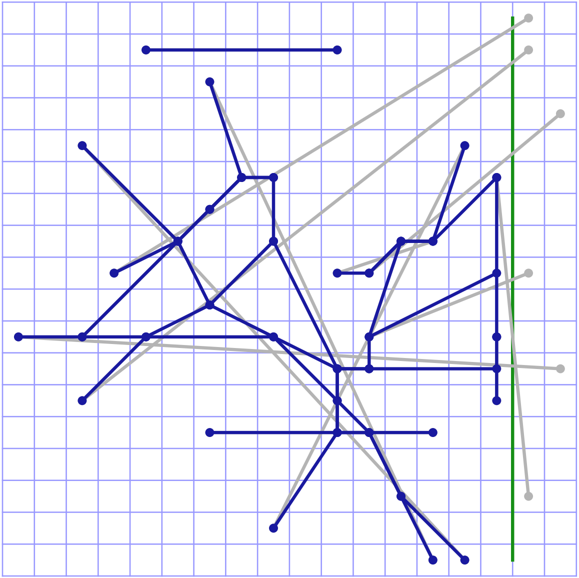
Algorithm One



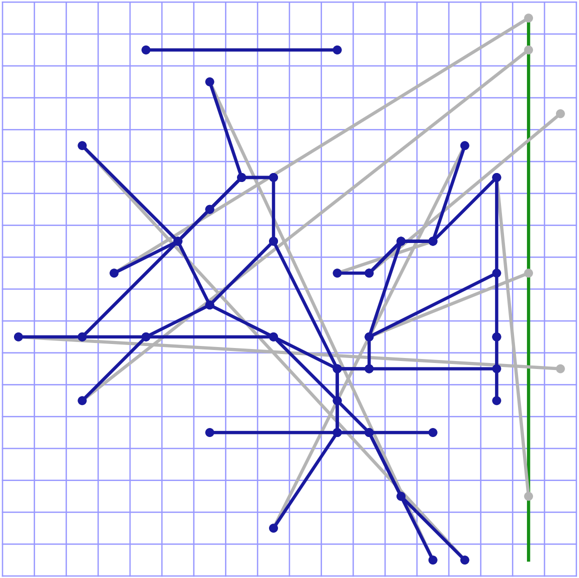
Algorithm One



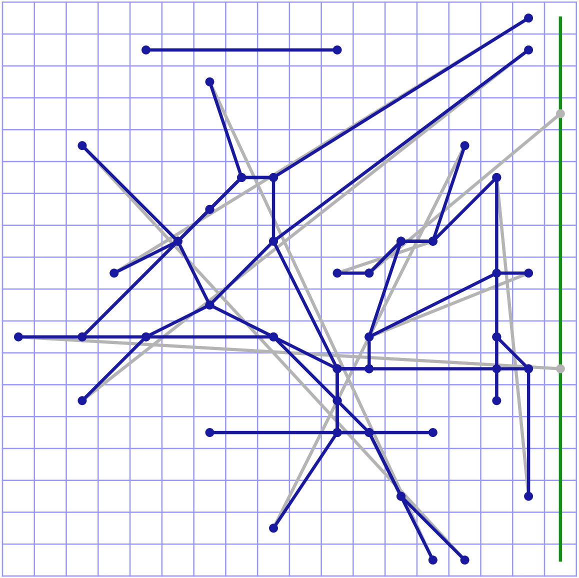
Algorithm One



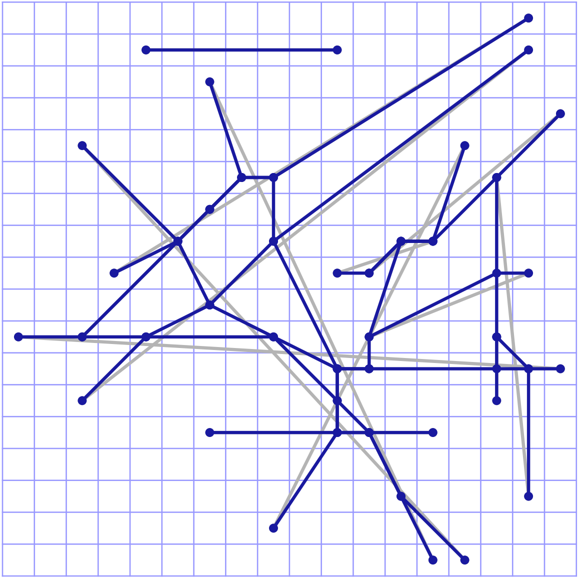
Algorithm One



Algorithm One



Algorithm One



Performance: Sweep Process

- a segment can occur in at most two seed events in a single column

Performance: Sweep Process

- a segment can occur in at most two seed events in a single column
- $O(is(c)\log n)$ cost of extracting seed events, where $is(c)$ is number of segments hot within c

Performance: Sweep Process

- a segment can occur in at most two seed events in a single column
- $O(is(c) \log n)$ cost of extracting seed events, where $is(c)$ is number of segments hot within c
- endpoint events can be included in this term

Performance: Sweep Process

- a segment can occur in at most two seed events in a single column
- $O(is(c)\log n)$ cost of extracting seed events, where $is(c)$ is number of segments hot within c
- endpoint events can be included in this term
- using pointers, we can exchange segments within the tree in $O(1)$ time, so cost of exchanges over all columns is $O(|I|)$

Performance: Sweep Process

- a segment can occur in at most two seed events in a single column
- $O(is(c)\log n)$ cost of extracting seed events, where $is(c)$ is number of segments hot within c
- endpoint events can be included in this term
- using pointers, we can exchange segments within the tree in $O(1)$ time, so cost of exchanges over all columns is $O(|I|)$
- recalculating internal paths for each of the $O(is(c))$ segments can be done in $O(is(c)\log n)$ time

Performance: Sweep Process

- a segment can occur in at most two seed events in a single column
- $O(is(c) \log n)$ cost of extracting seed events, where $is(c)$ is number of segments hot within c
- endpoint events can be included in this term
- using pointers, we can exchange segments within the tree in $O(1)$ time, so cost of exchanges over all columns is $O(|I|)$
- recalculating internal paths for each of the $O(is(c))$ segments can be done in $O(is(c) \log n)$ time
- running time for process is thus $O(|I| + \Sigma_c\, is(c) \log n)$

Performance: Hot Pixel Process

- hot pixels are sorted; by theorem 3, there are $O(is(c))$ hot pixels, and since $is(c)$ is $O(n)$, cost is $O(is(c) \lg n)$

Performance: Hot Pixel Process

- hot pixels are sorted; by theorem 3, there are $O(is(c))$ hot pixels, and since $is(c)$ is $O(n)$, cost is $O(is(c) \lg n)$
- running time for this process can be included in that of the sweep process

Performance: Snap Process

- each snapped fragment can be generated in $O(1)$ time

Performance: Snap Process

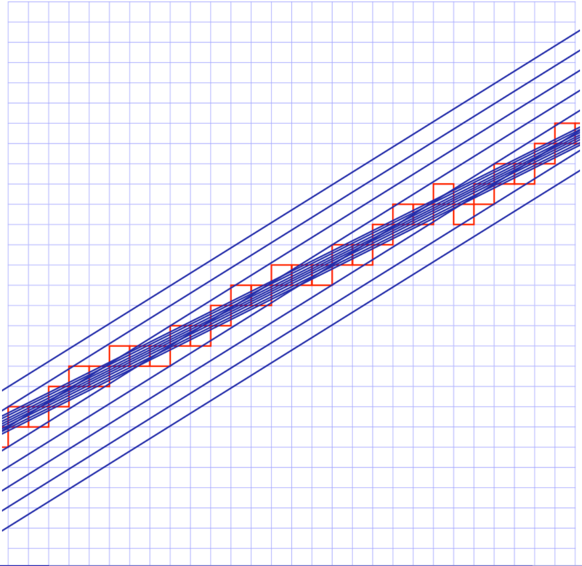
- each snapped fragment can be generated in $O(1)$ time
- I_m^* is set of all snapped fragments

Performance: Snap Process

- each snapped fragment can be generated in $O(1)$ time
- I_m^* is set of all snapped fragments
- running time for all processes is $O(|I| + \Sigma_c\, is(c) \log n + |I_m^*|)$

Algorithm Two: Why?

As observed by Halperin & Packer (2002), $|I_m^*|$ can be $\Theta(n^3)$:



Algorithm Two

- use approach of de Berg et al (2007) to group snapped fragments into bundles

Algorithm Two

- use approach of de Berg et al (2007) to group snapped fragments into bundles
- generate an arc for each unique snapped fragment, at a cost of $O(\log n)$ each

Algorithm Two

- use approach of de Berg et al (2007) to group snapped fragments into bundles
- generate an arc for each unique snapped fragment, at a cost of $O(\log n)$ each
- perform initial pass to find hot pixels; this is first algorithm with snap process omitted

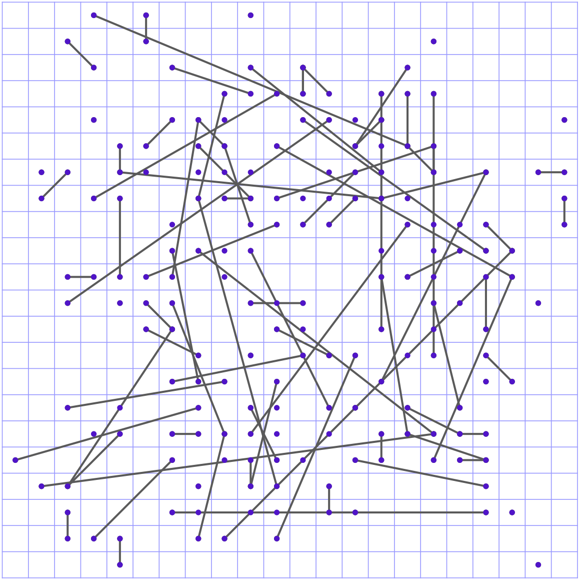
Algorithm Two

- use approach of de Berg et al (2007) to group snapped fragments into bundles
- generate an arc for each unique snapped fragment, at a cost of $O(\log n)$ each
- perform initial pass to find hot pixels; this is first algorithm with snap process omitted
- perform two additional passes to find segment/hot pixel intersections: vertical sweep of 'mostly horizontal' segments, horizontal sweep of 'mostly vertical' segments

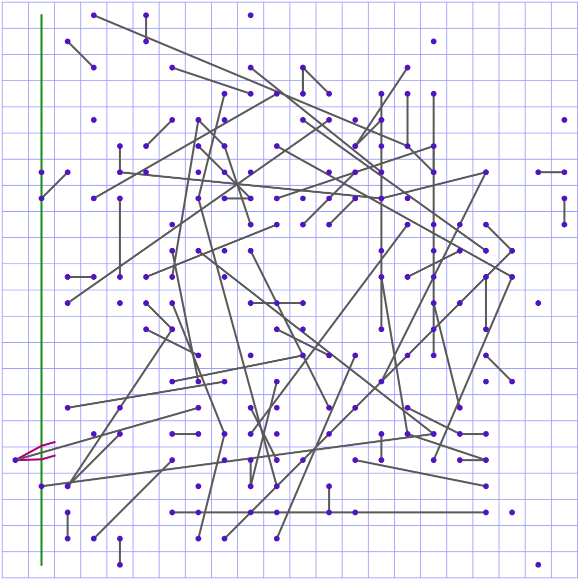
Algorithm Two

- use approach of de Berg et al (2007) to group snapped fragments into bundles
- generate an arc for each unique snapped fragment, at a cost of $O(\log n)$ each
- perform initial pass to find hot pixels; this is first algorithm with snap process omitted
- perform two additional passes to find segment/hot pixel intersections: vertical sweep of 'mostly horizontal' segments, horizontal sweep of 'mostly vertical' segments
- running time is $O(|I| + \Sigma_c\, is(c) \log n + |I^*| \log n)$, where $|I^*|$ is complexity of snapped arrangement

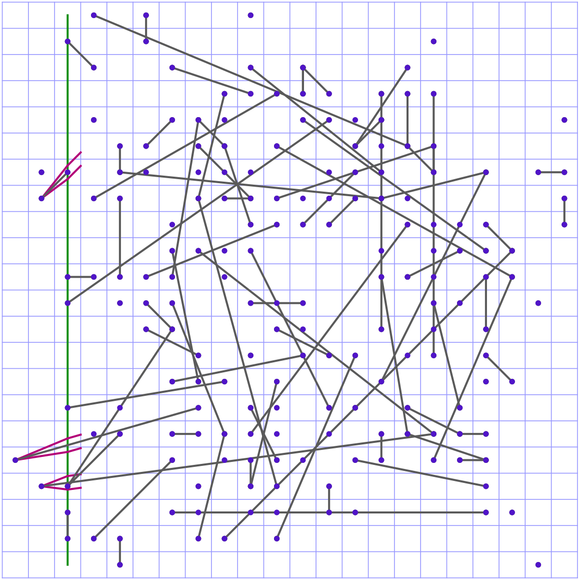
Algorithm Two



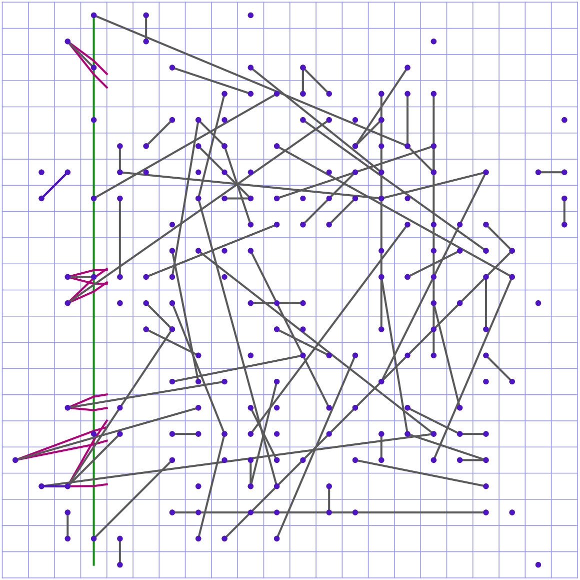
Algorithm Two



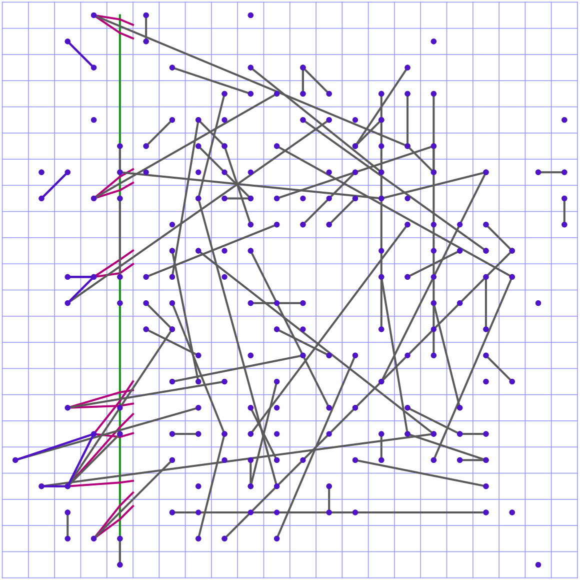
Algorithm Two



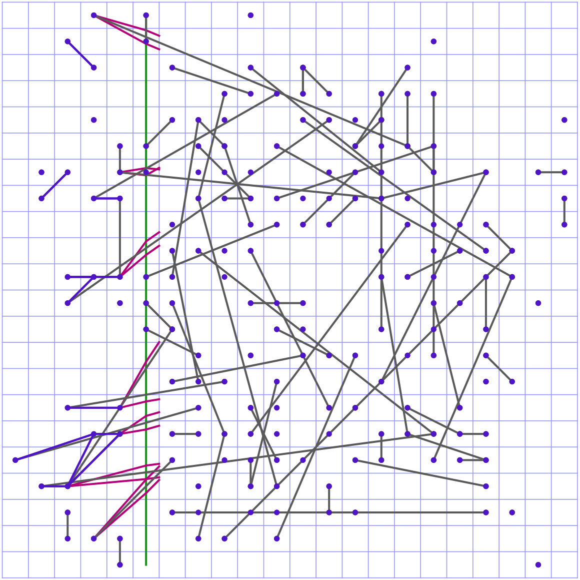
Algorithm Two



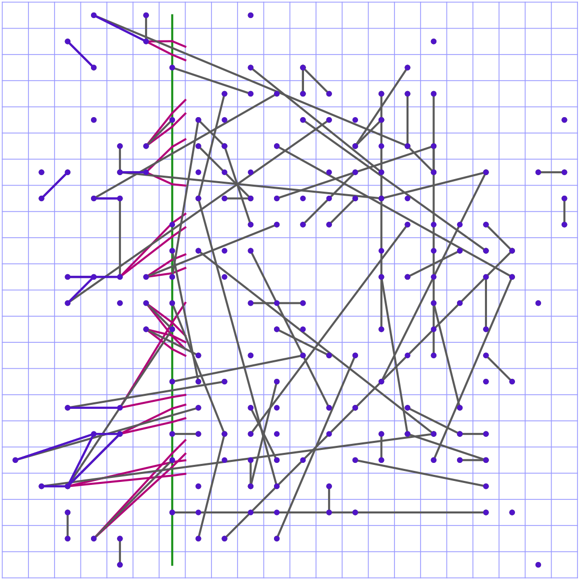
Algorithm Two



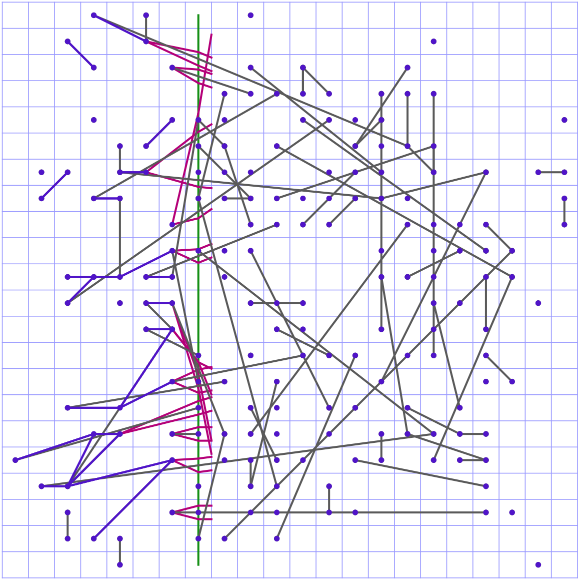
Algorithm Two



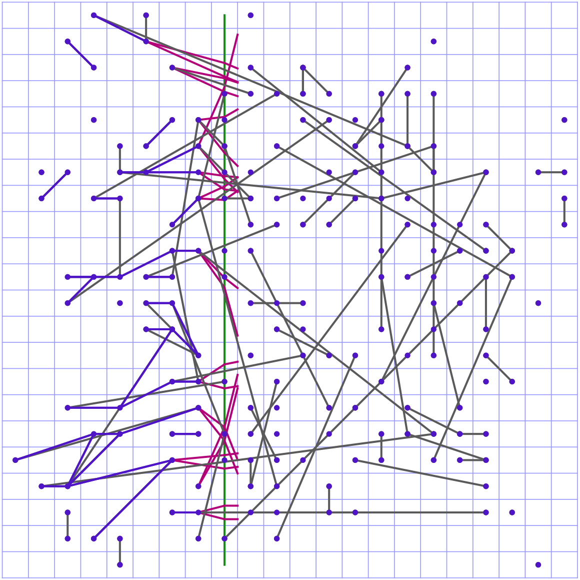
Algorithm Two



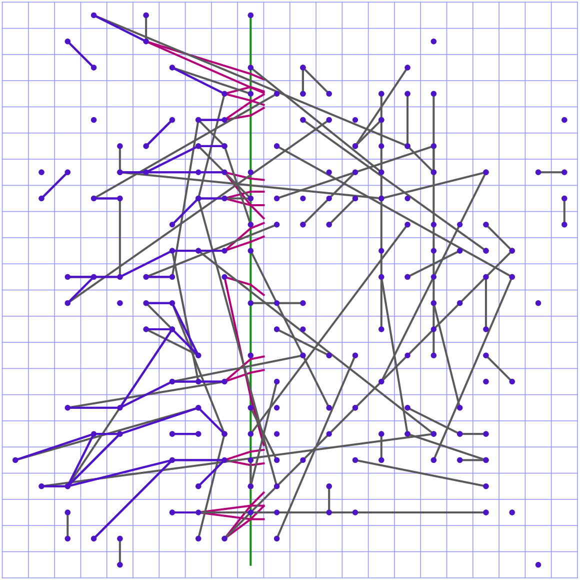
Algorithm Two



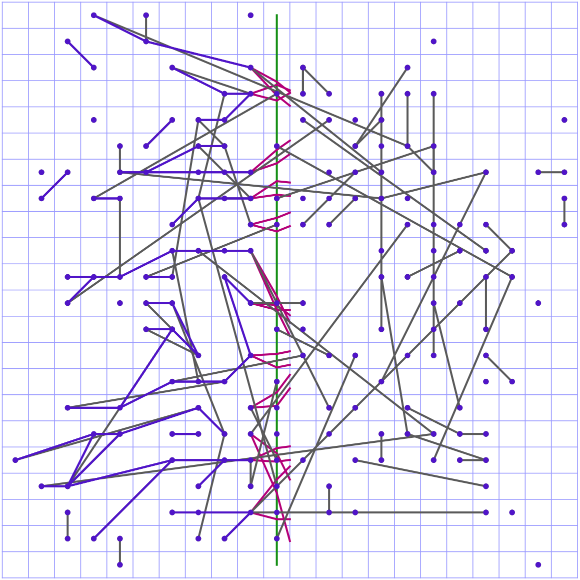
Algorithm Two



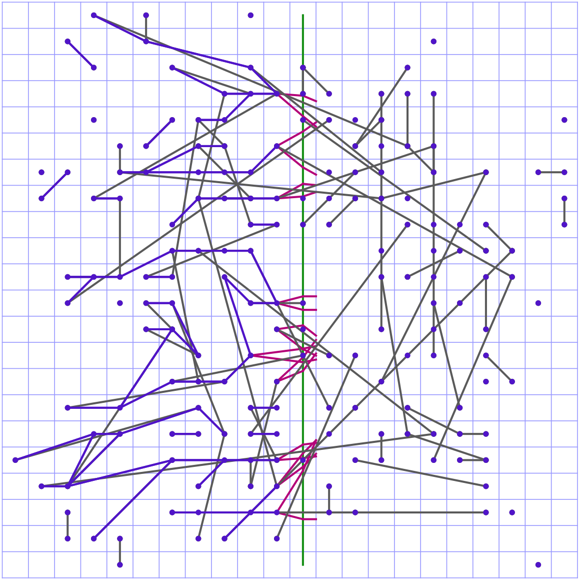
Algorithm Two



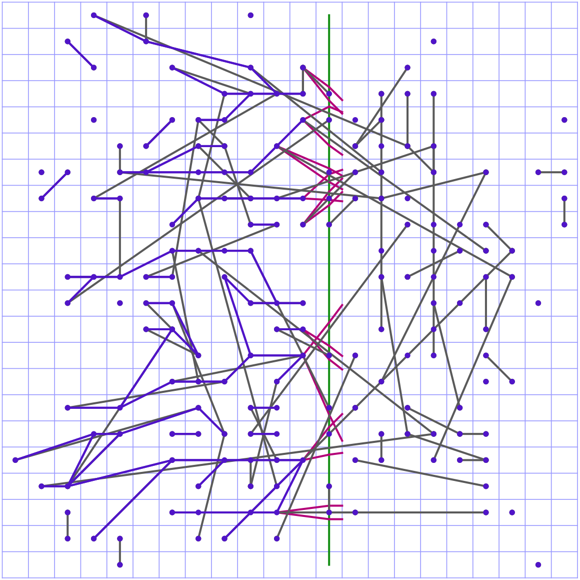
Algorithm Two



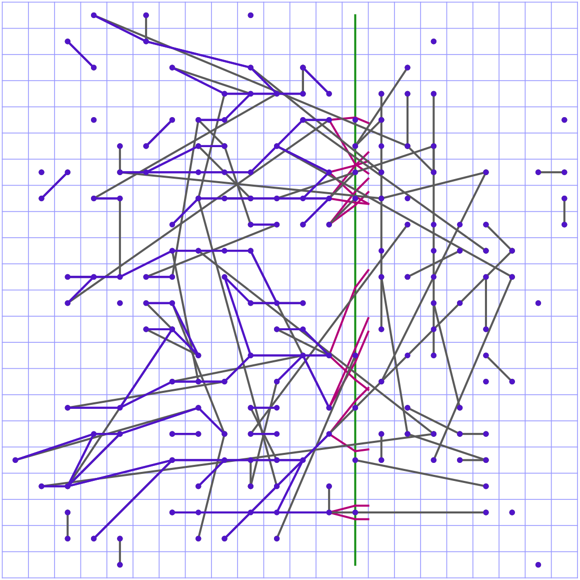
Algorithm Two



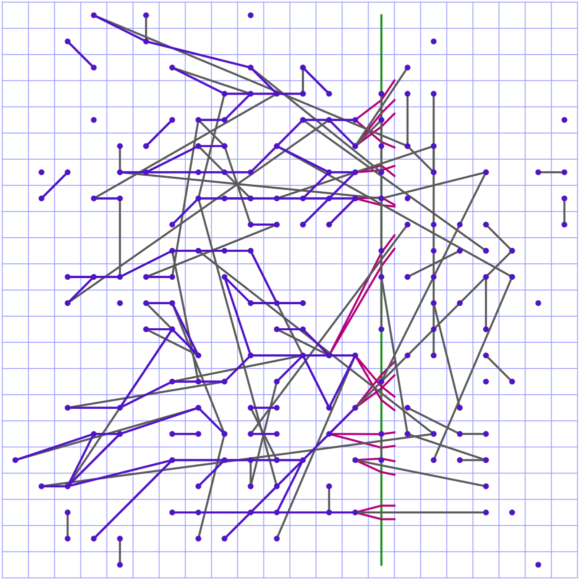
Algorithm Two



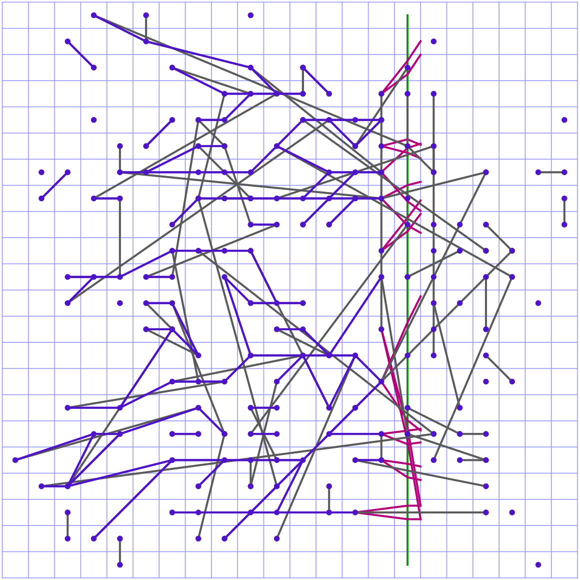
Algorithm Two



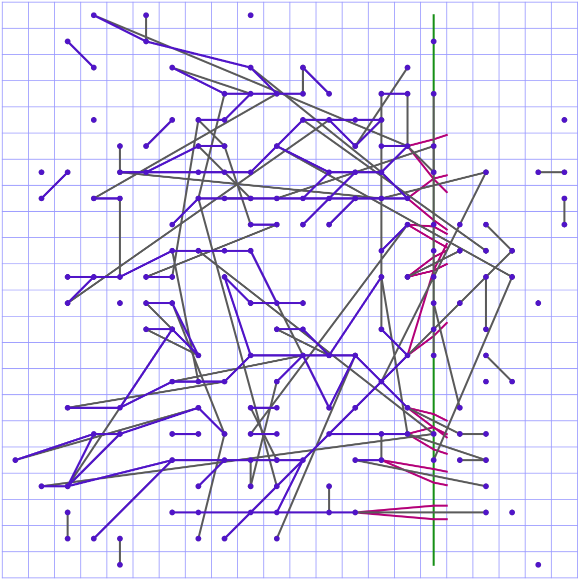
Algorithm Two



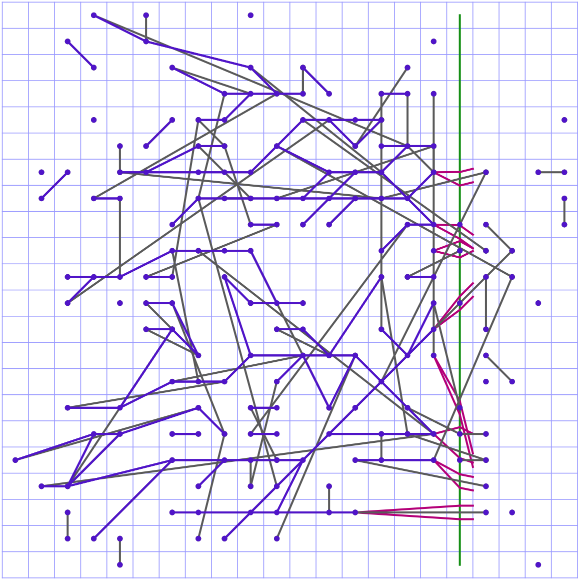
Algorithm Two



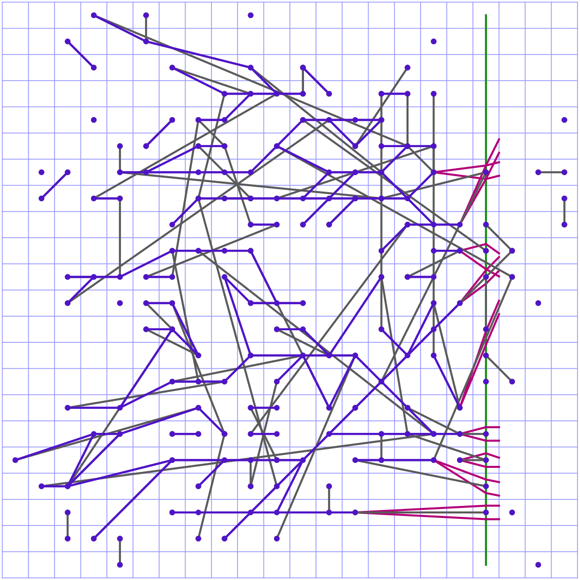
Algorithm Two



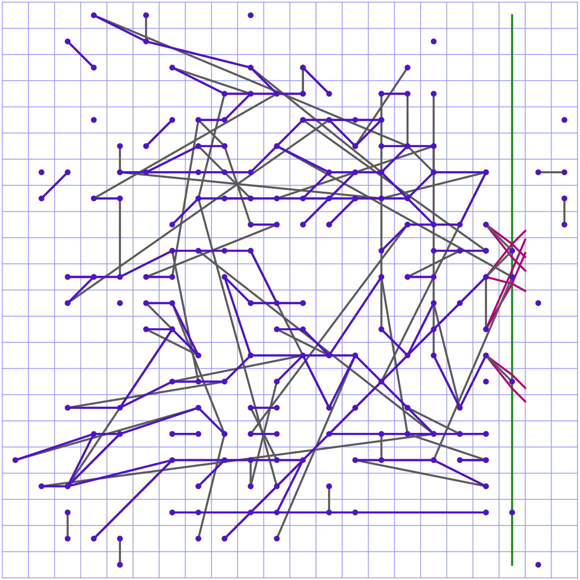
Algorithm Two



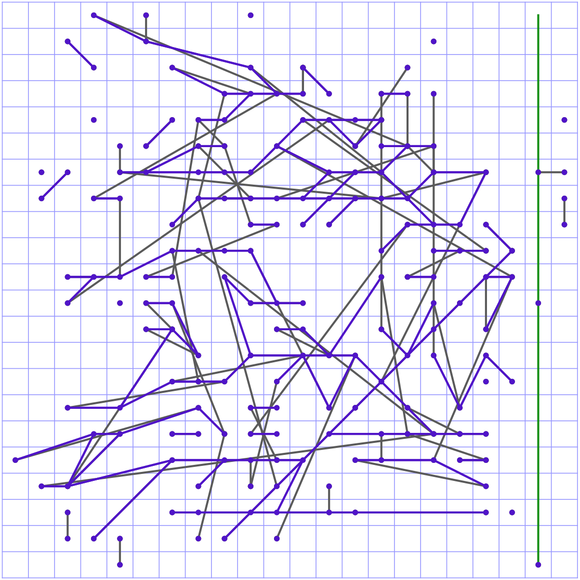
Algorithm Two



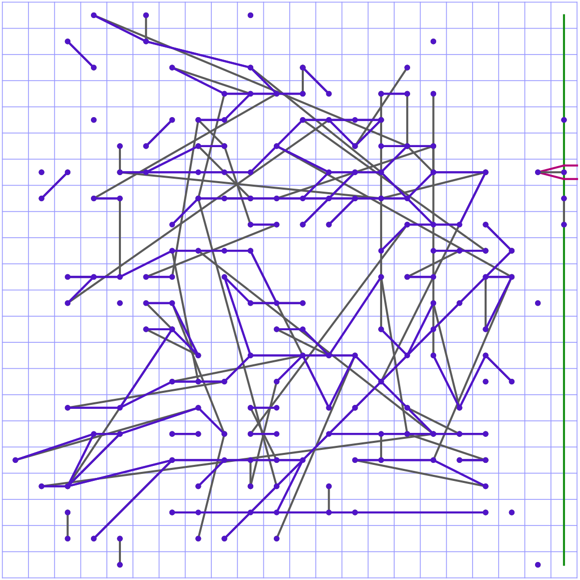
Algorithm Two



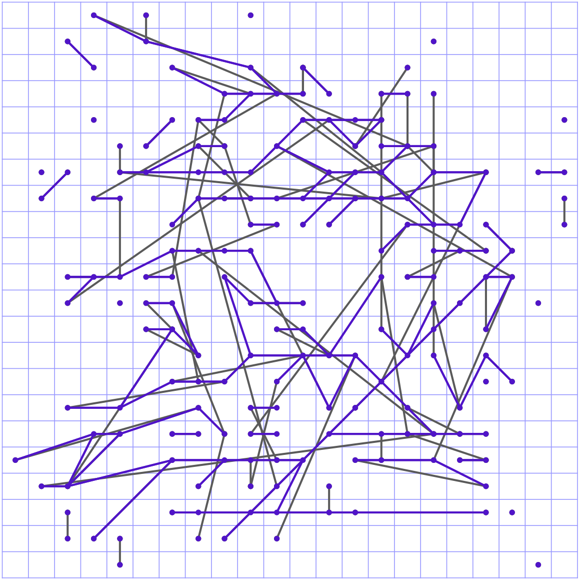
Algorithm Two



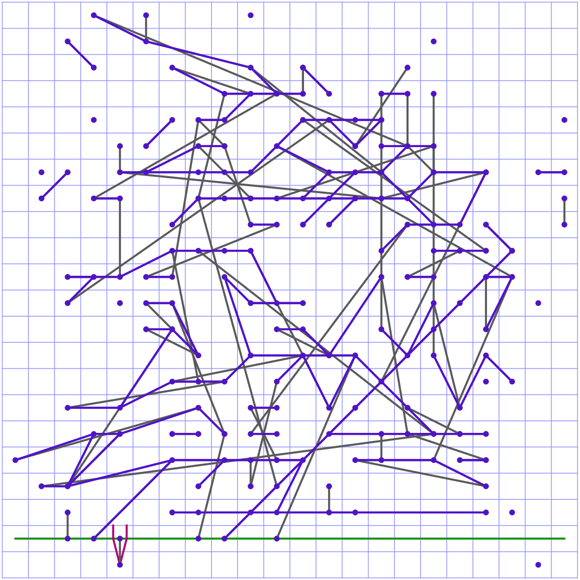
Algorithm Two



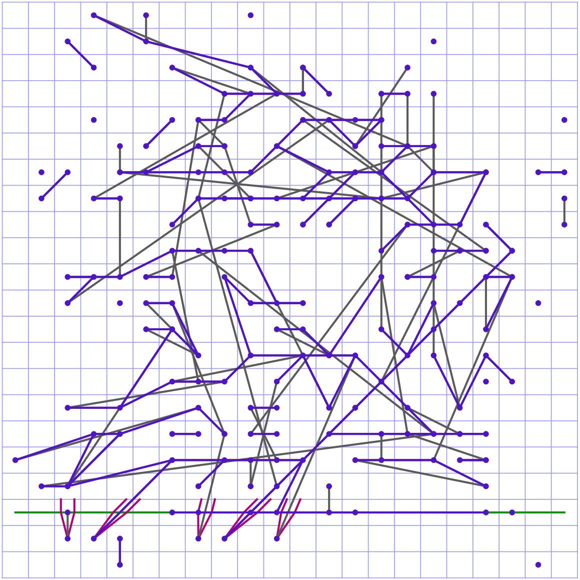
Algorithm Two



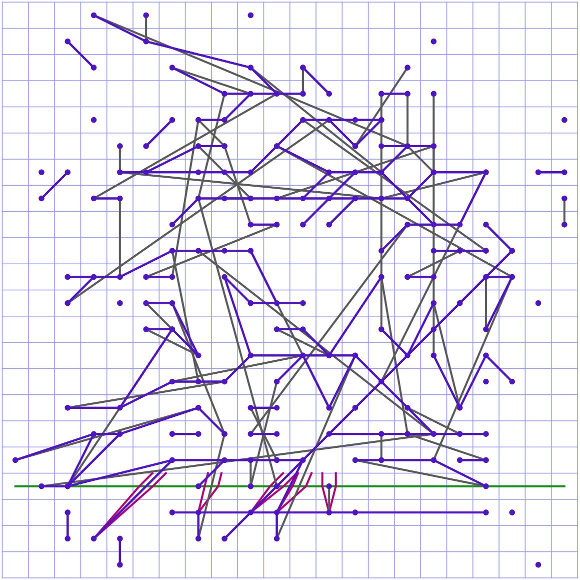
Algorithm Two



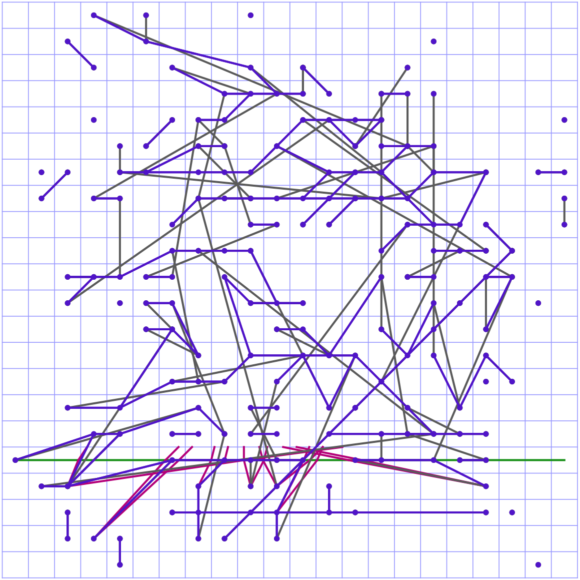
Algorithm Two



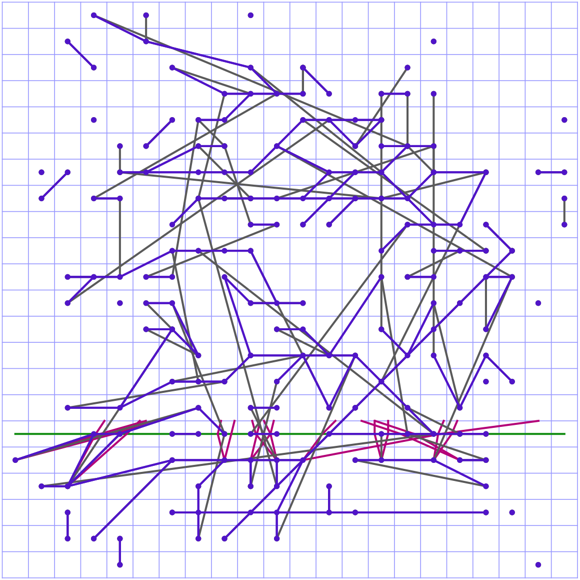
Algorithm Two



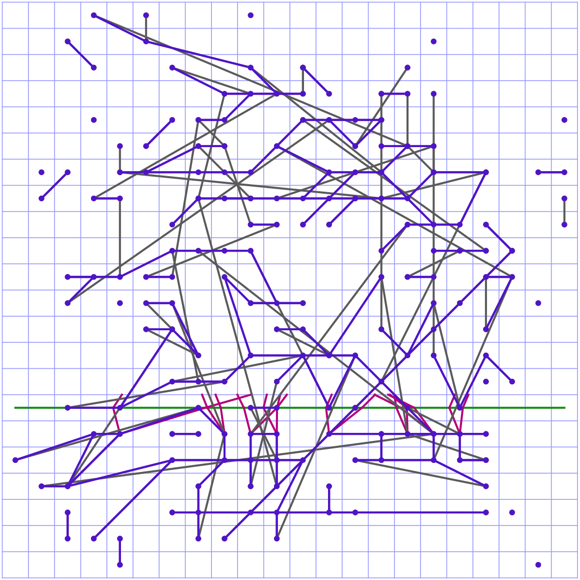
Algorithm Two



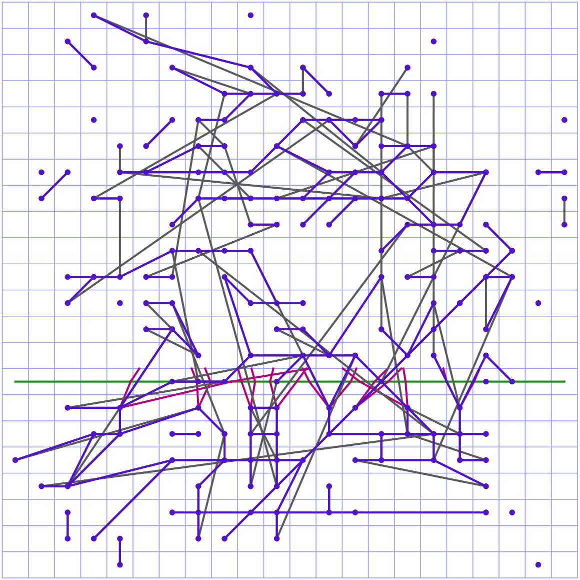
Algorithm Two



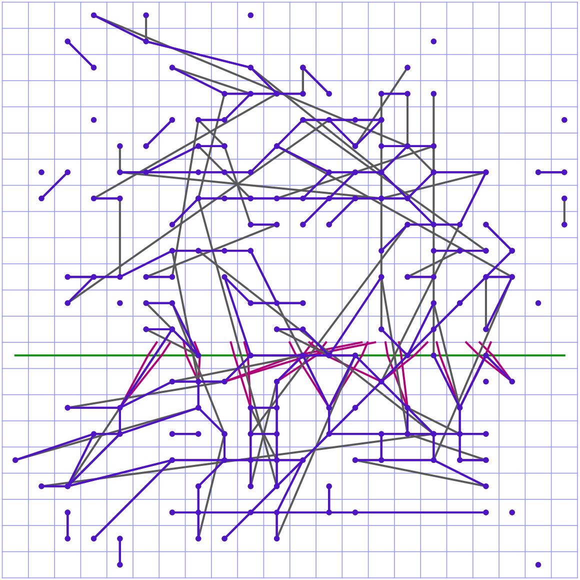
Algorithm Two



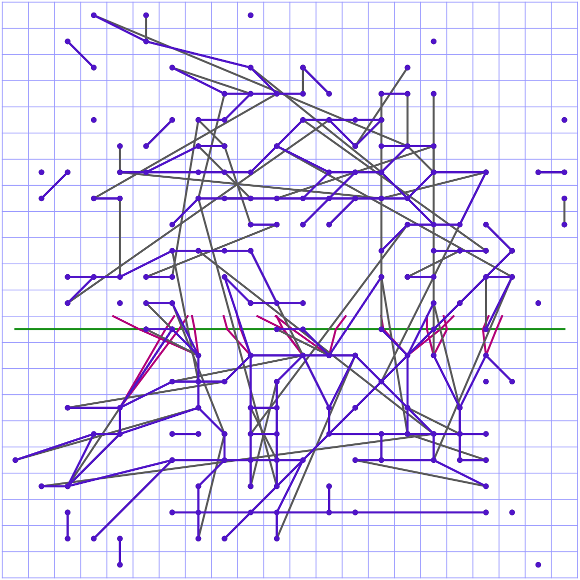
Algorithm Two



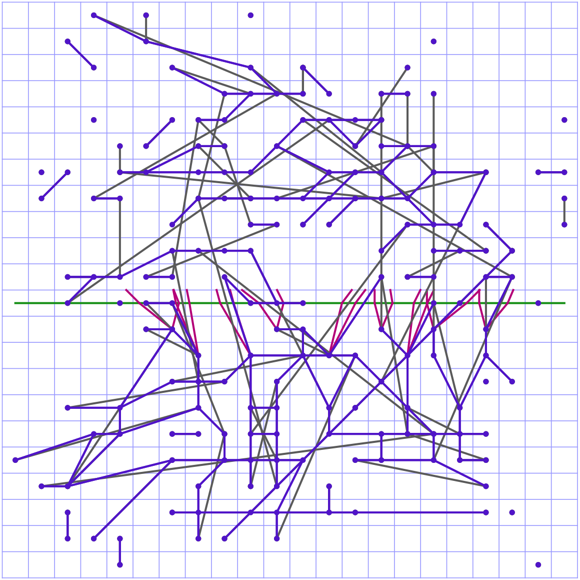
Algorithm Two



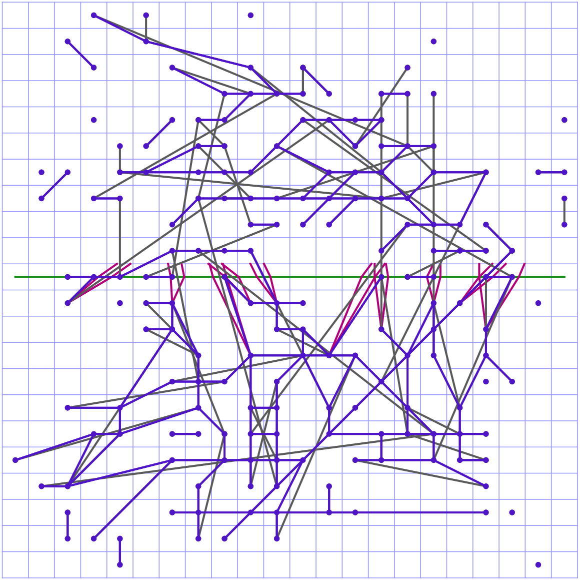
Algorithm Two



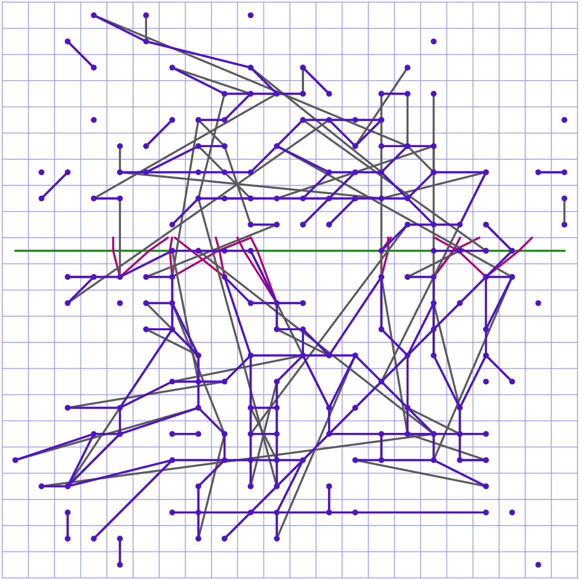
Algorithm Two



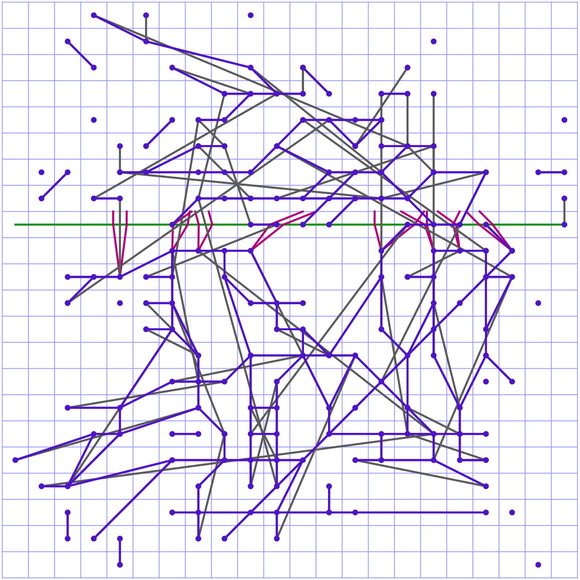
Algorithm Two



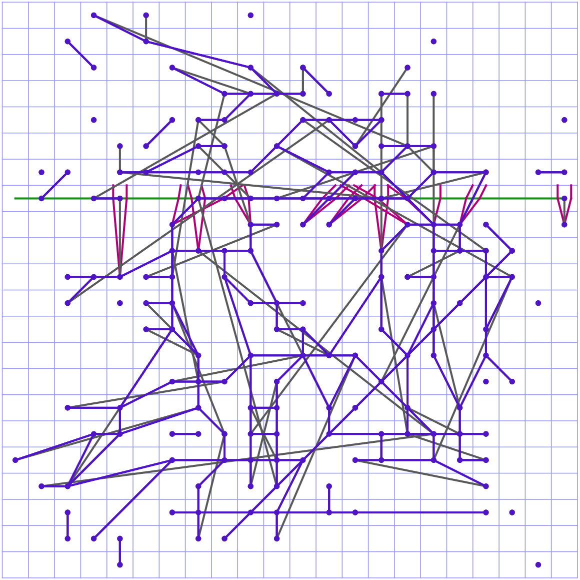
Algorithm Two



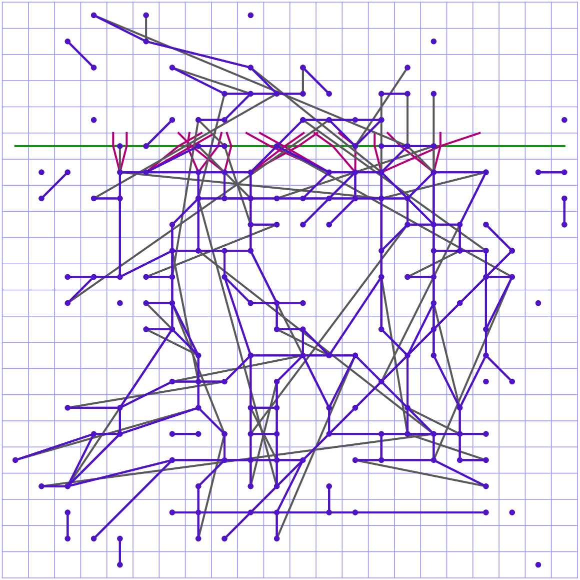
Algorithm Two



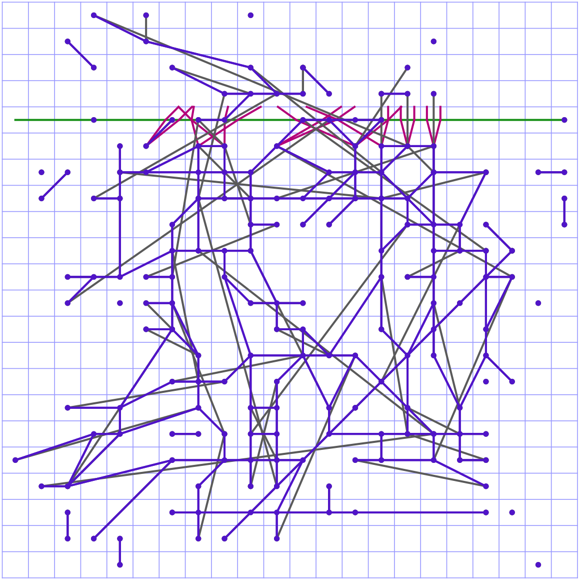
Algorithm Two



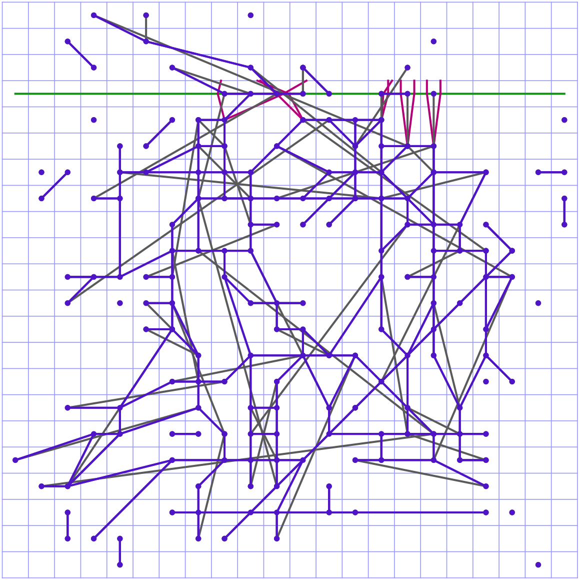
Algorithm Two



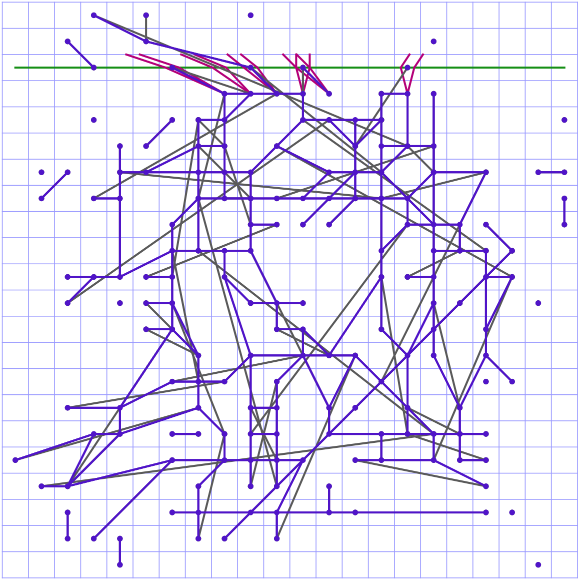
Algorithm Two



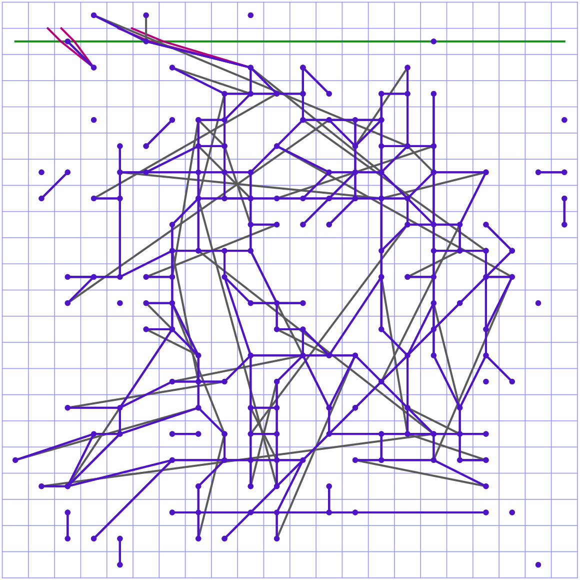
Algorithm Two



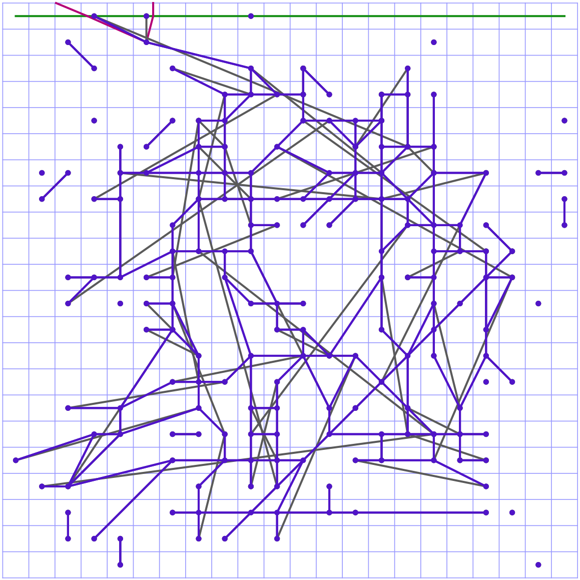
Algorithm Two



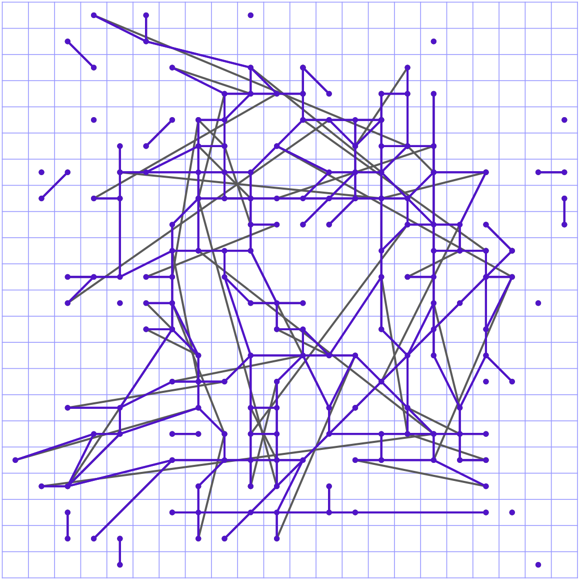
Algorithm Two



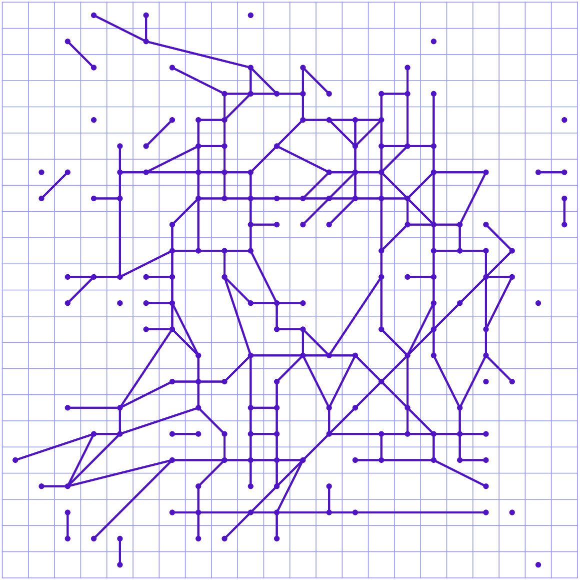
Algorithm Two



Algorithm Two



Algorithm Two



Conclusion

Both of our algorithms:

- use simple integer arithmetic

Conclusion

Both of our algorithms:

- use simple integer arithmetic
- are robust

Conclusion

Both of our algorithms:

- use simple integer arithmetic
- are robust
- are practical

Conclusion

Both of our algorithms:

- use simple integer arithmetic
- are robust
- are practical
- are linear in $|I|$

Conclusion

Both of our algorithms:

- use simple integer arithmetic
- are robust
- are practical
- are linear in $|I|$
- are based on interactions within columns of pixels ($is(c)$) instead of individual pixels ($is(h)$ or $ed(h)$)

Conclusion

An applet demonstrating both algorithms is available at

`http://www.sfu.ca/~jpsember/snap.html`