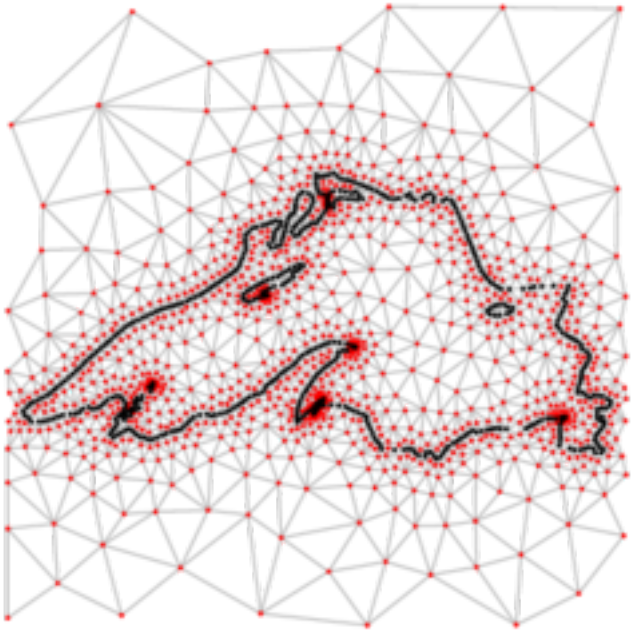
Parallelism in Dynamic Well-Spaced Point Sets

Umut A. Acar[1], Andrew Cotter[2]
Benoît Hudson[3], Duru Türkoğlu[4]



1. Max-Planck Institute for Software Systems
2. Toyota Technological Institute at Chicago
3. Autodesk, Inc.
4. University of Chicago

Dynamic-parallel relationship

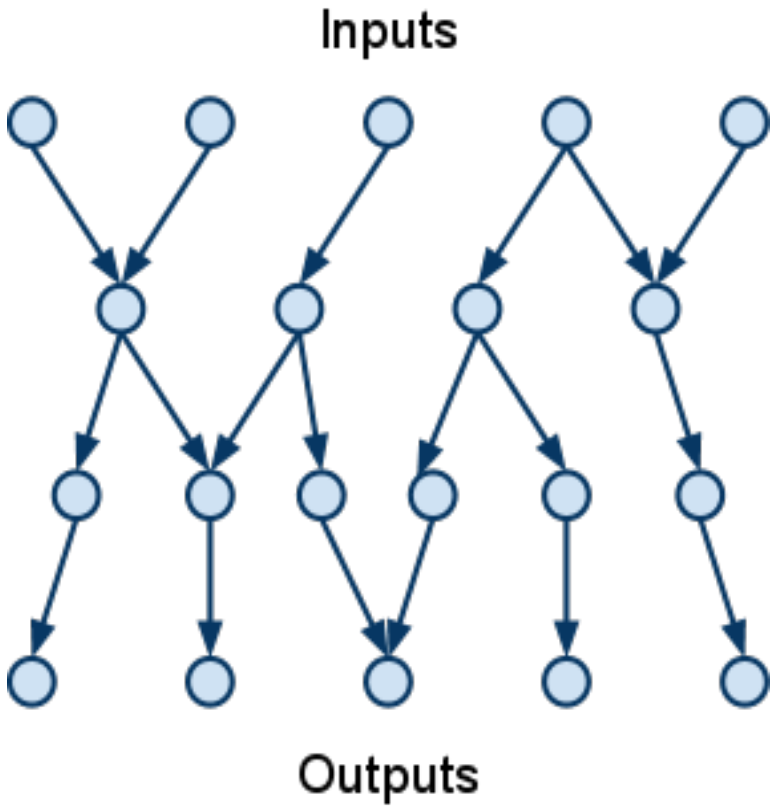
	Work	**Time**
Sequential	$n \cdot \log(n)$	$n \cdot \log(n)$
Parallel	$n \cdot \log(n)$	$\log(n)$
Dynamic (*k* changes)	$k \cdot \log(n)$	$k \cdot \log(n)$
Dynamic & parallel	$k \cdot \log(n)$	$\log(n)$

- Parallel algorithms reduce *time*
- Dynamic algorithms reduce *work*
- We can get the best of both worlds!

Dynamic-parallel relationship

<u>Computation graph</u>

- Nodes are tasks
- Edges are dependencies



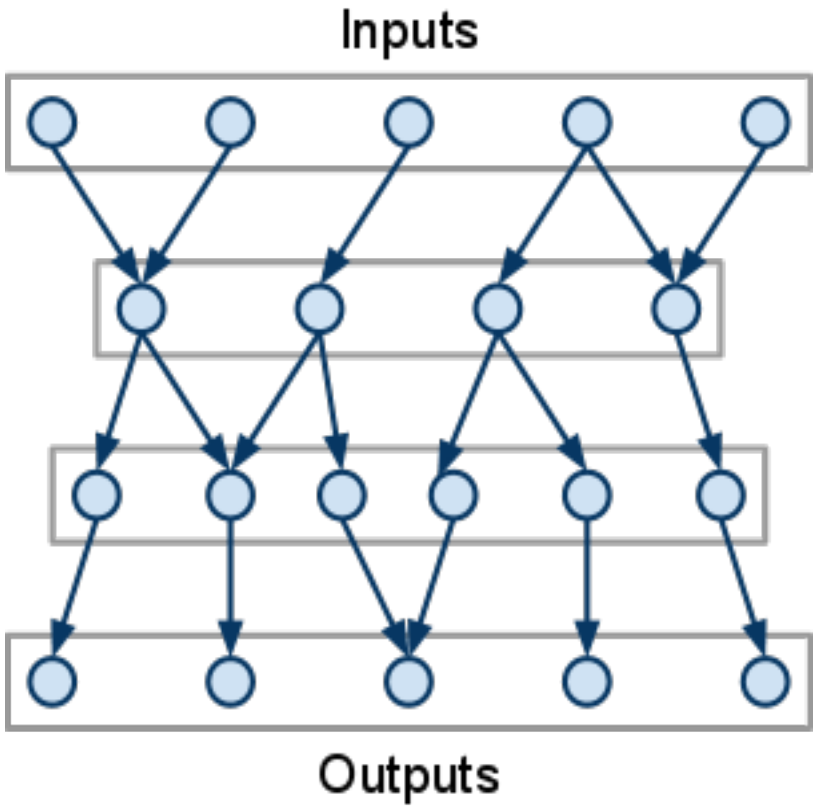
Dynamic-parallel relationship

<u>Computation graph</u>

- Nodes are tasks
- Edges are dependencies

<u>Parallelism</u>

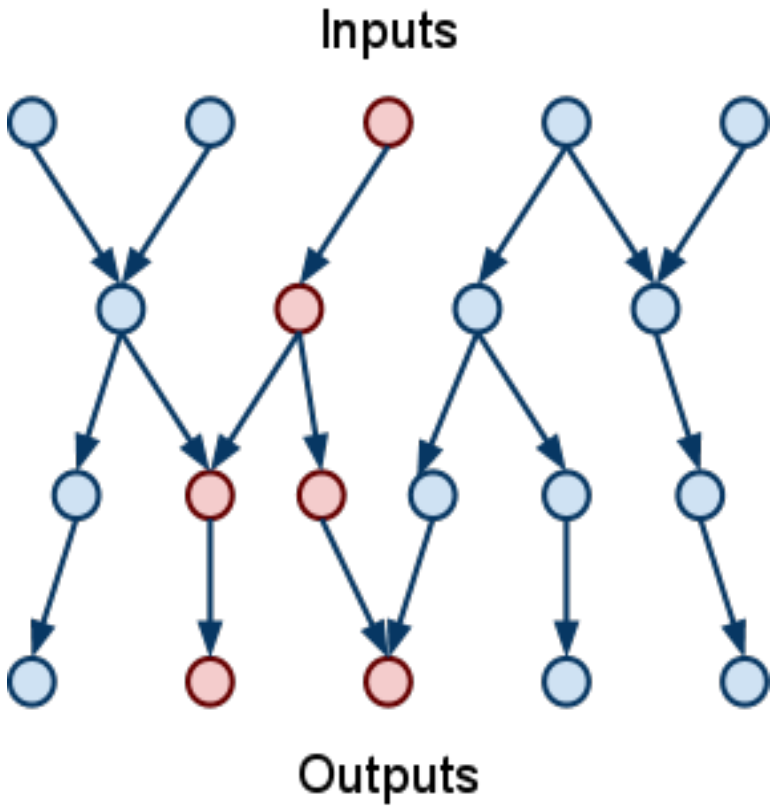
- Execute independent operations in parallel
- Low depth results in good speedups



Dynamic-parallel relationship

<u>Change propagation</u>

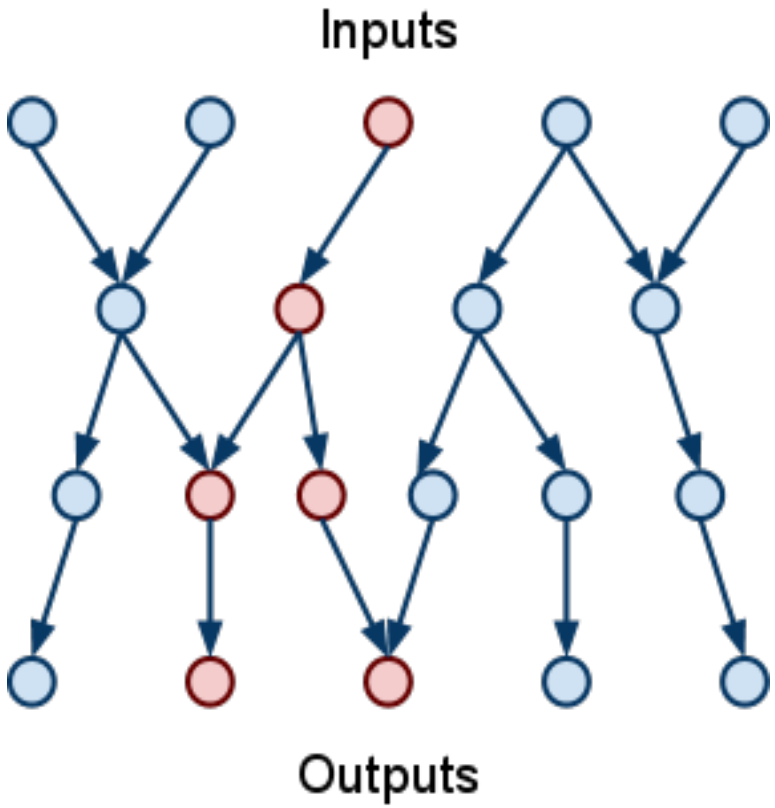
- Re-execute affected nodes when input changes
  - This may change the graph structure
- For good performance:
  - Low depth
  - "Narrow" dependency chains



Dynamic-parallel relationship

<u>Change propagation</u>

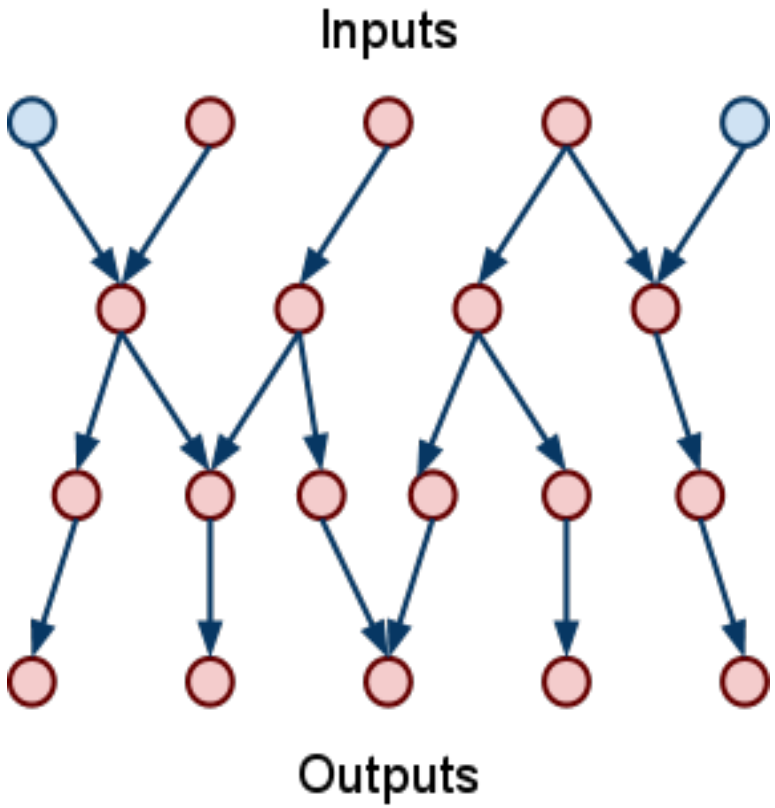
- Ideally:
  - Small changes: little re-execution



Dynamic-parallel relationship

<u>Change propagation</u>

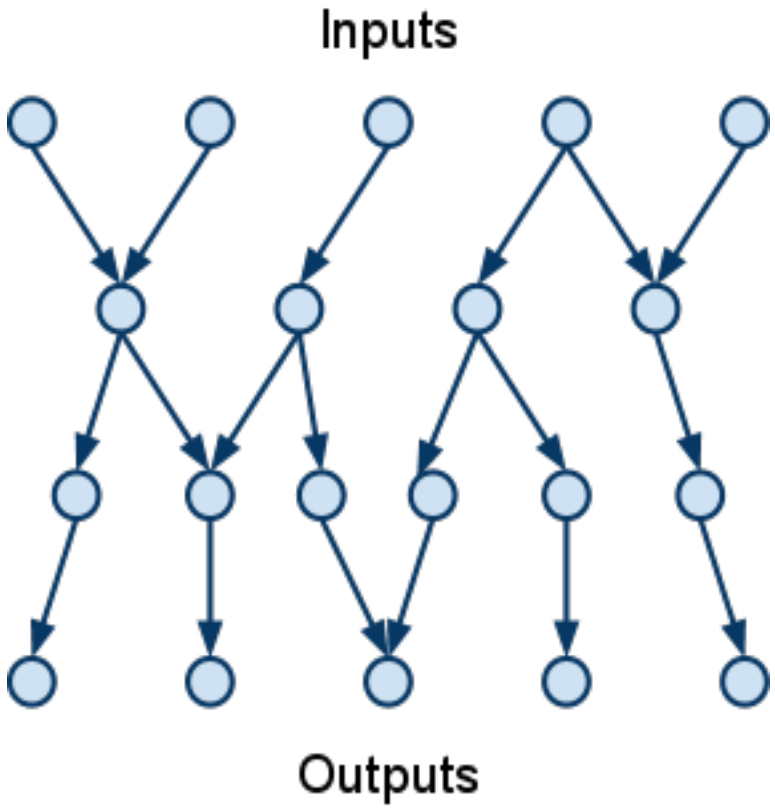
- Ideally:
  - Small changes: little re-execution
  - Large changes: much re-execution



Dynamic-parallel relationship

<u>Dynamic & parallel algorithms</u>

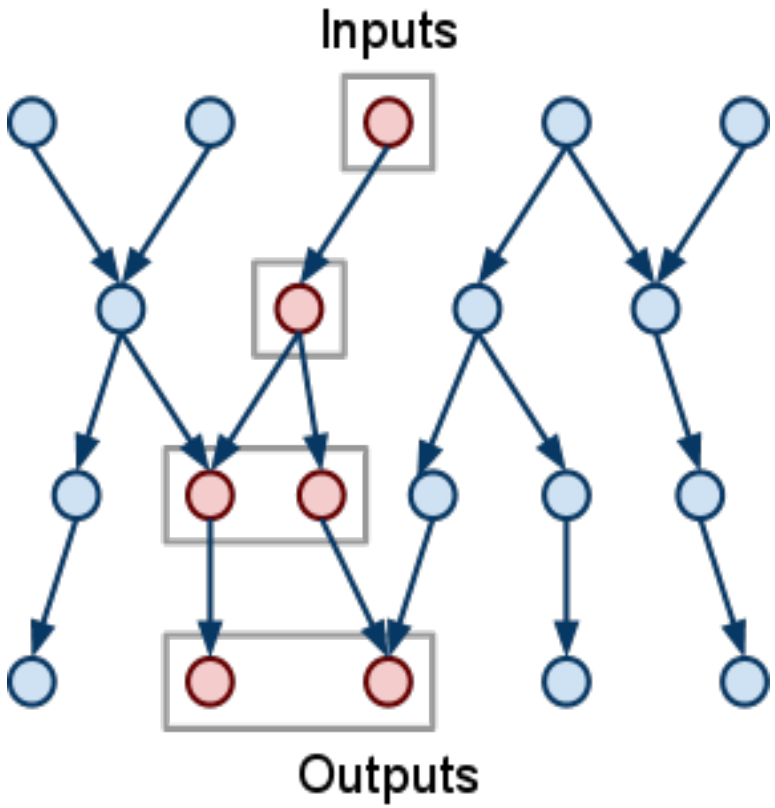
- Perform change propagation in parallel
- Parallel algorithms reduce time
- Dynamic algorithms reduce work
- Dynamic & parallel reduce both



Dynamic-parallel relationship

<u>Dynamic & parallel algorithms</u>

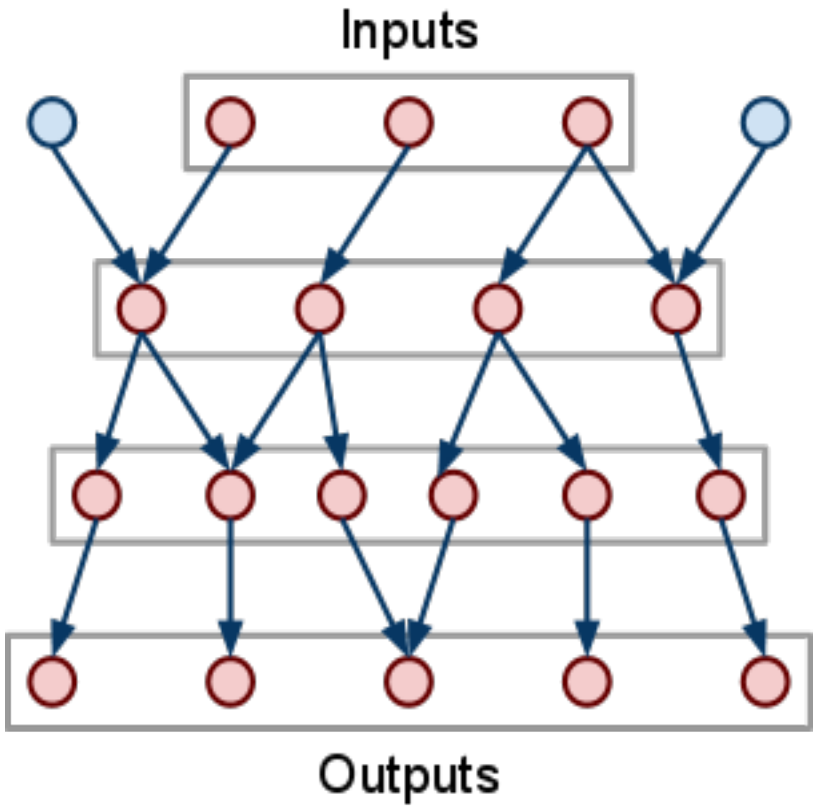
- Small changes result in:
  - Large gains from dynamism
  - Small gains from parallelism



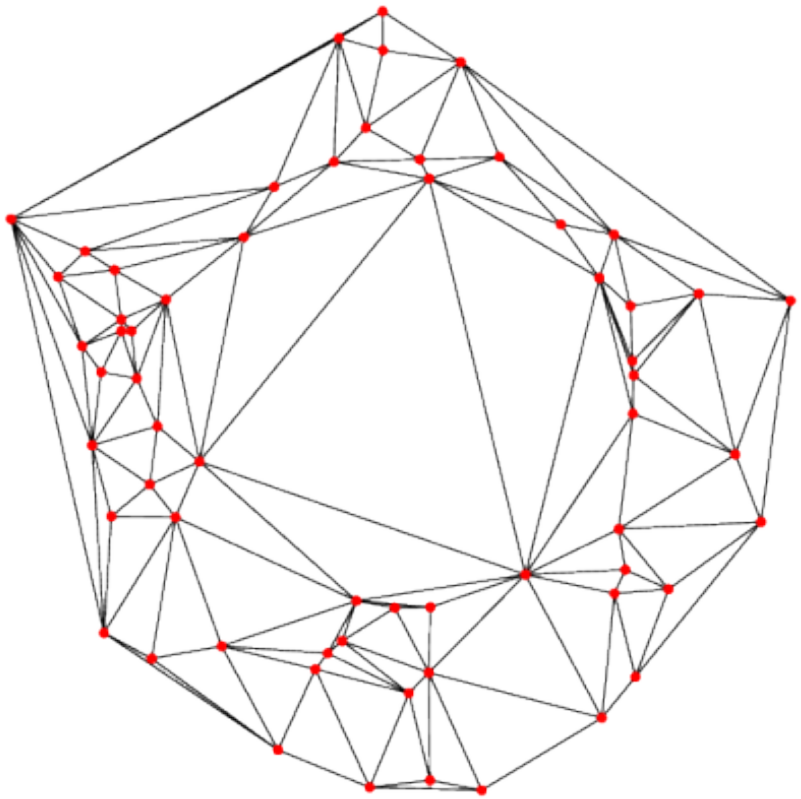
Dynamic-parallel relationship

<u>Dynamic & parallel algorithms</u>

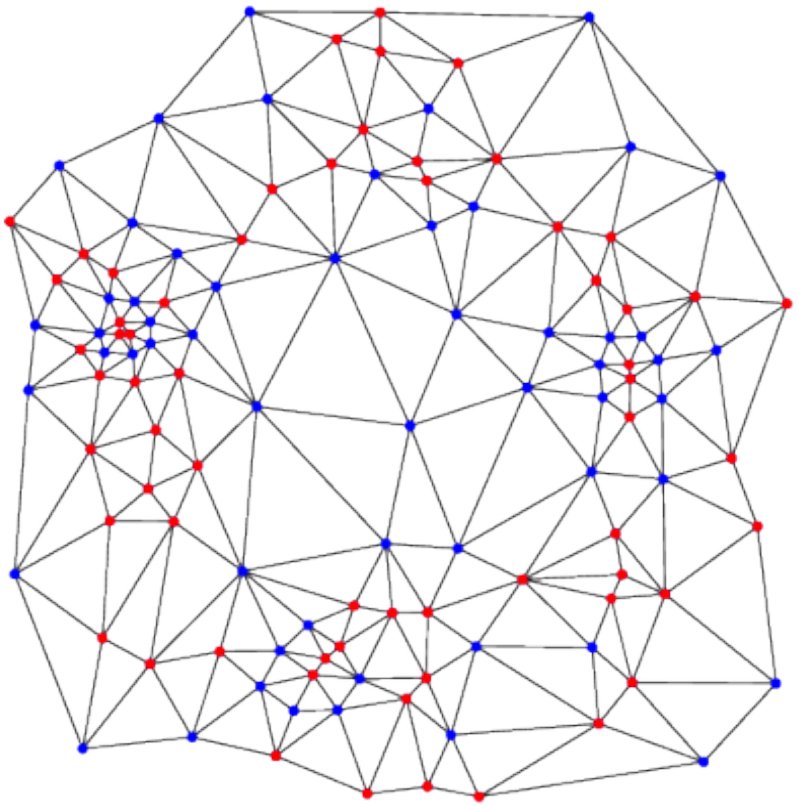
- Small changes result in:
  - Large gains from dynamism
  - Small gains from parallelism
- Large changes result in:
  - Small gains from dynamism
  - Large gains from parallelism



Well-spaced point sets



- "Skinny" triangles
- Small angles

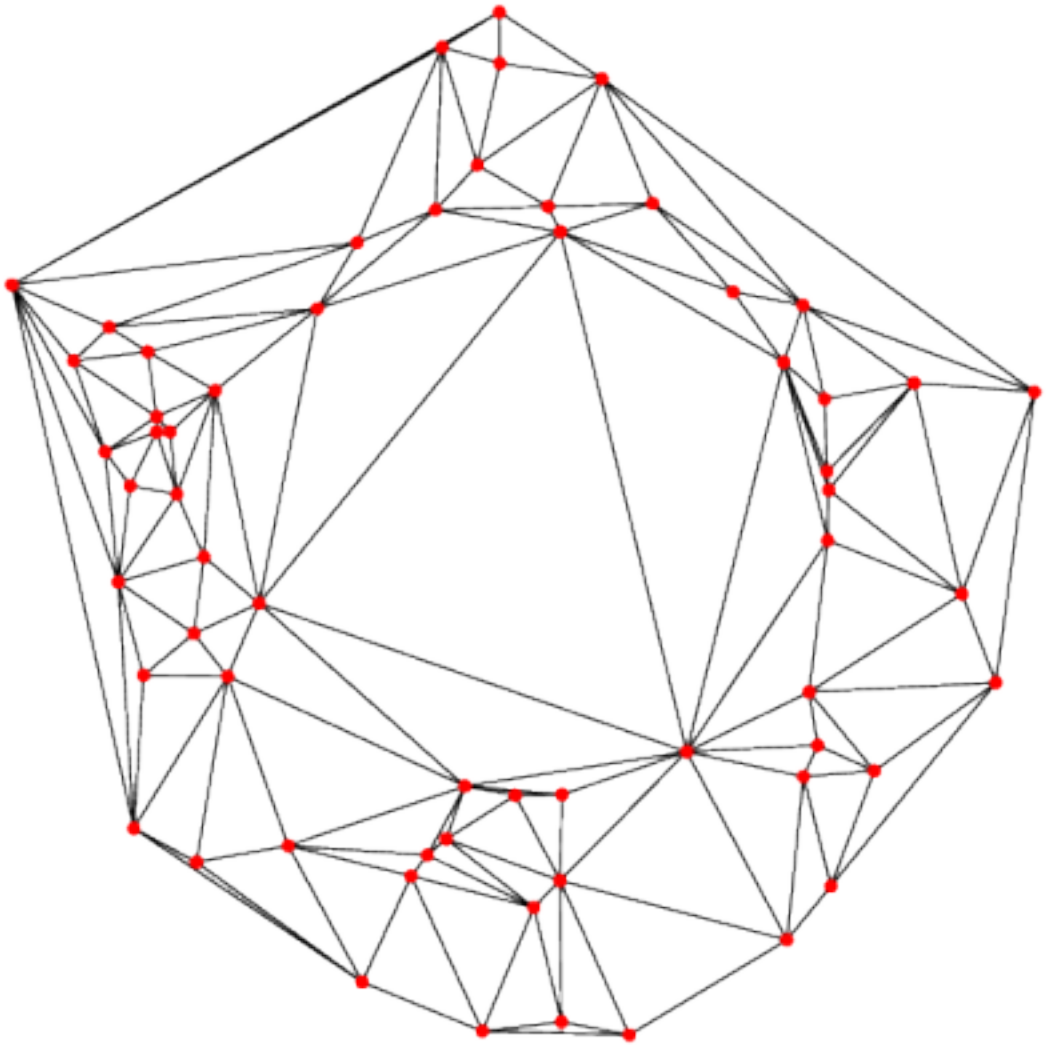


- No skinny triangles
- No small angles

Well-spaced point sets

<u>Input point set</u>

- Delaunay triangulation contains small angles



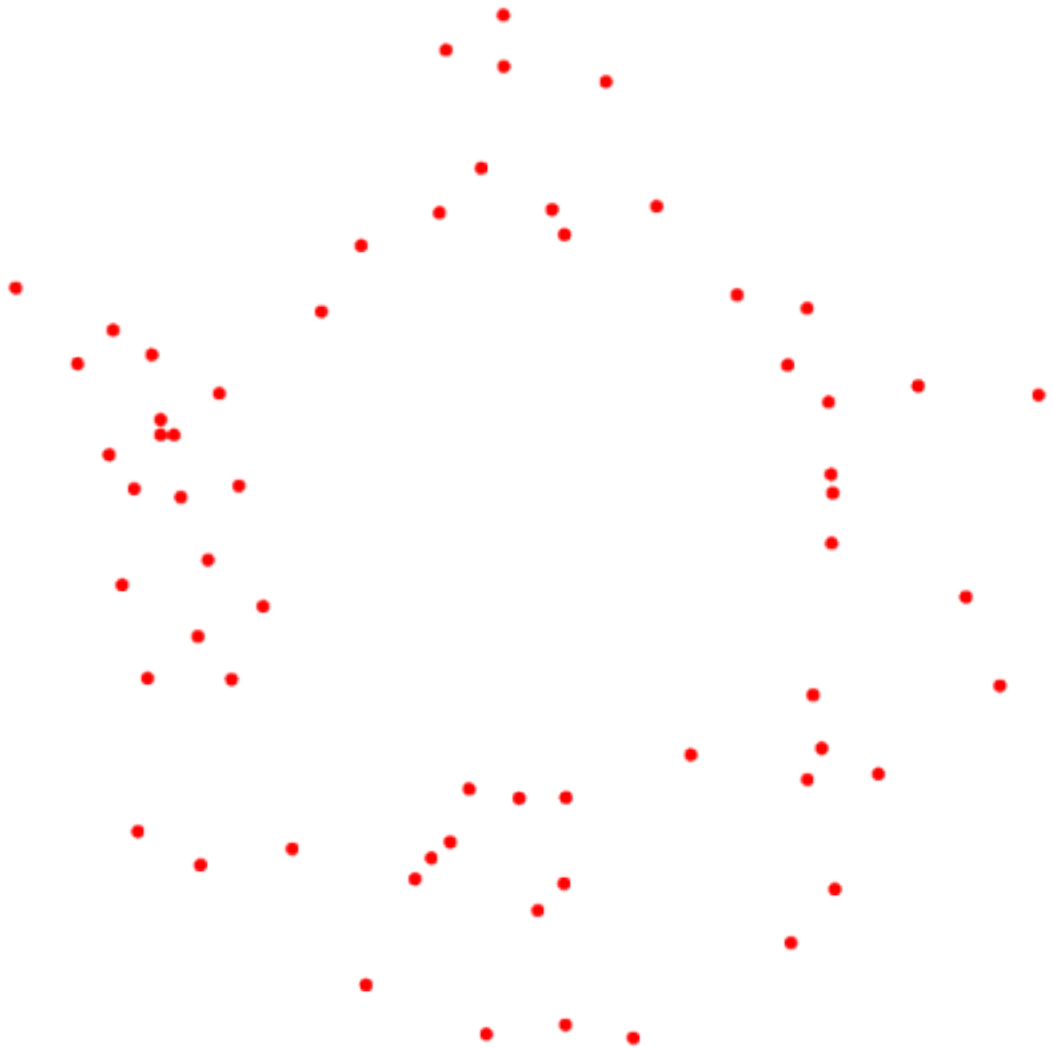
Well-spaced point sets

<u>Input point set</u>

- Delaunay triangulation contains small angles

<u>Procedure</u>

- Starting from input points



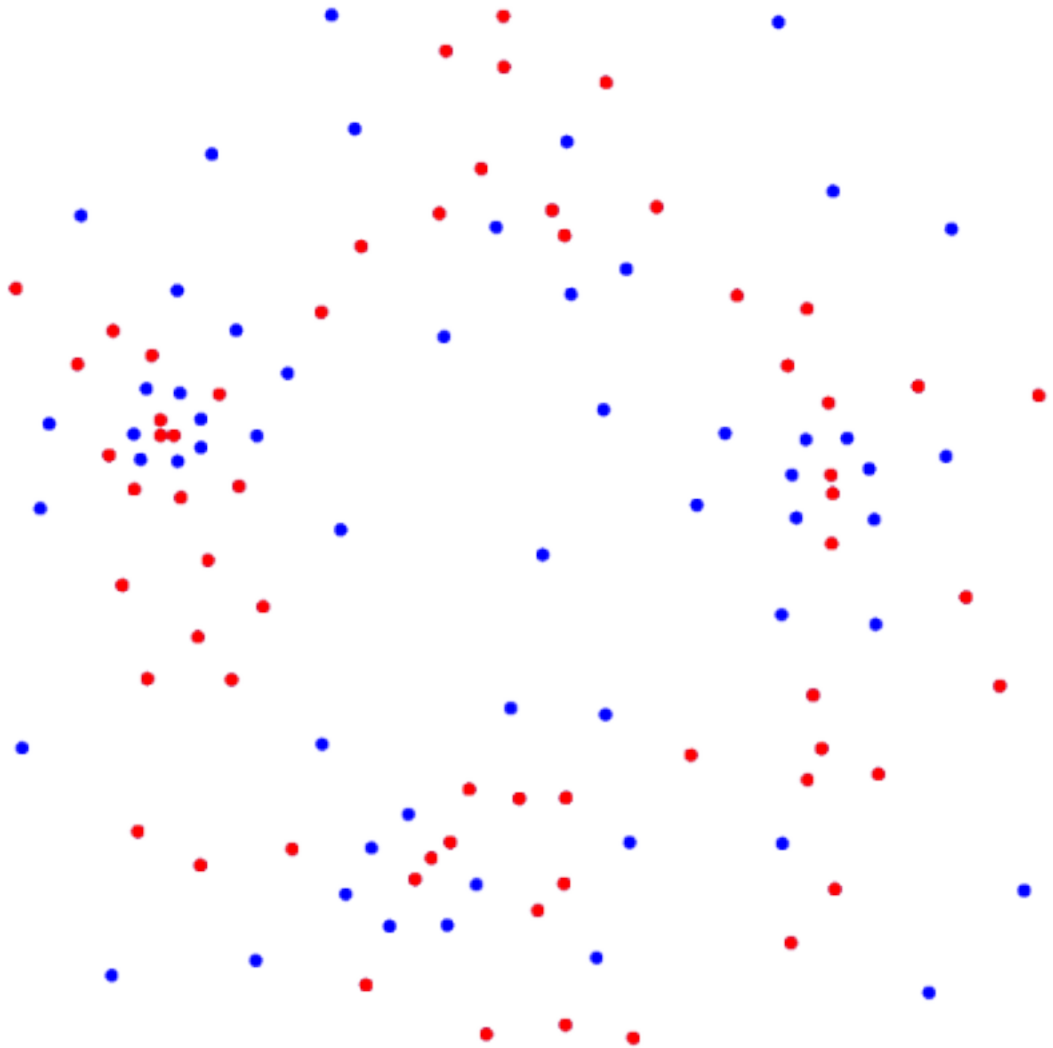
Well-spaced point sets

<u>Input point set</u>

- Delaunay triangulation contains small angles

<u>Procedure</u>

- Starting from input points
- Make point set well-spaced by inserting "Steiner points"



Well-spaced point sets

<u>Input point set</u>

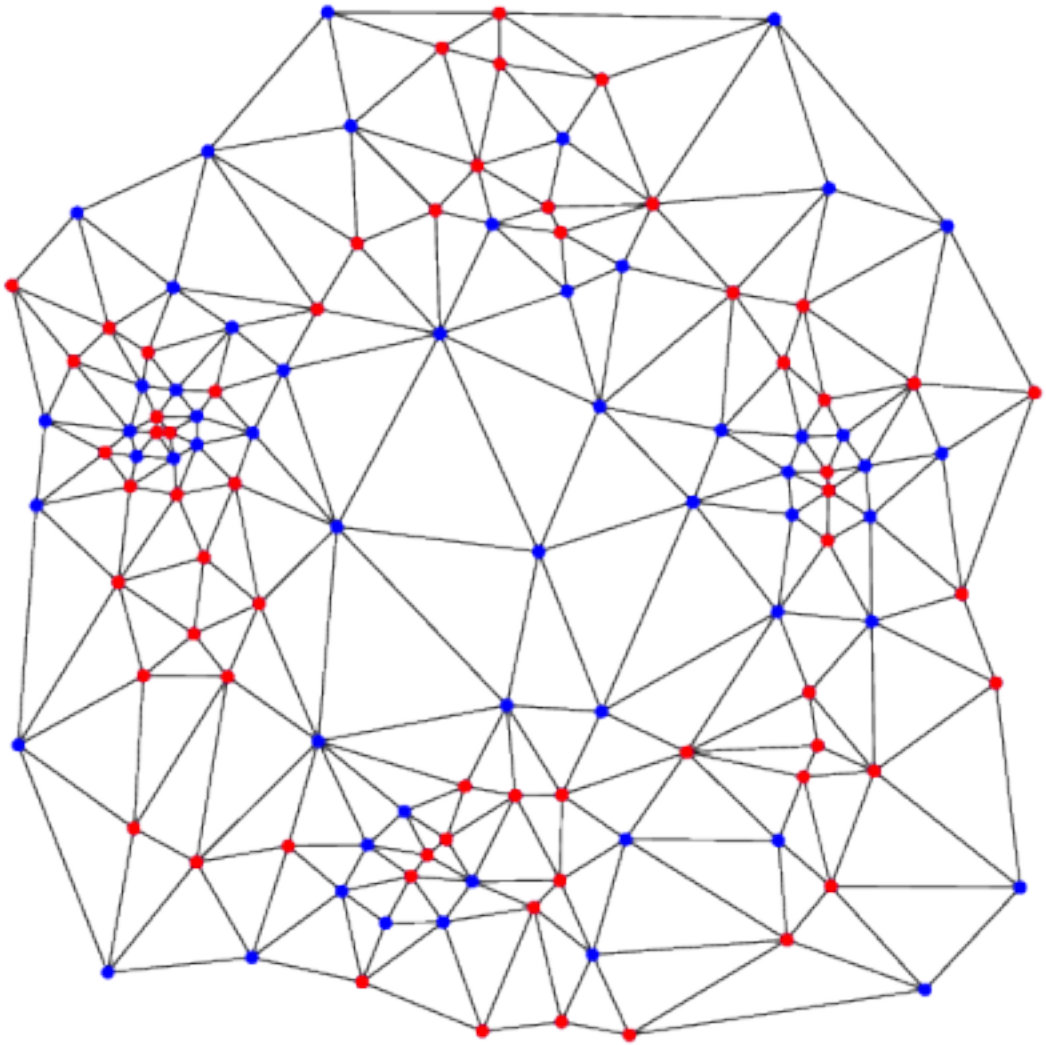
- Delaunay triangulation contains small angles

<u>Procedure</u>

- Starting from input points
- Make point set well-spaced by inserting "Steiner points"

<u>Output point set</u>

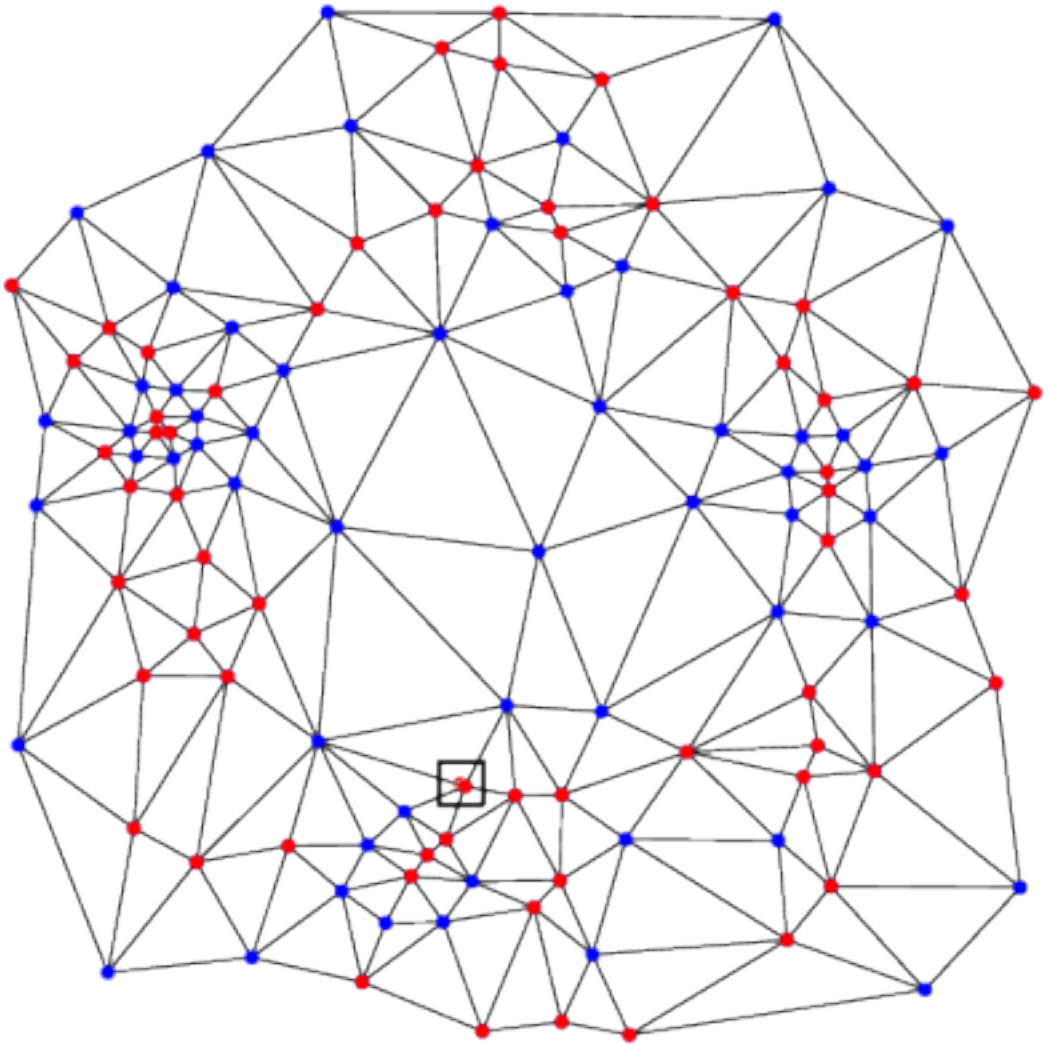
- No small angles



Well-spaced point sets

<u>Dynamic modifications</u>

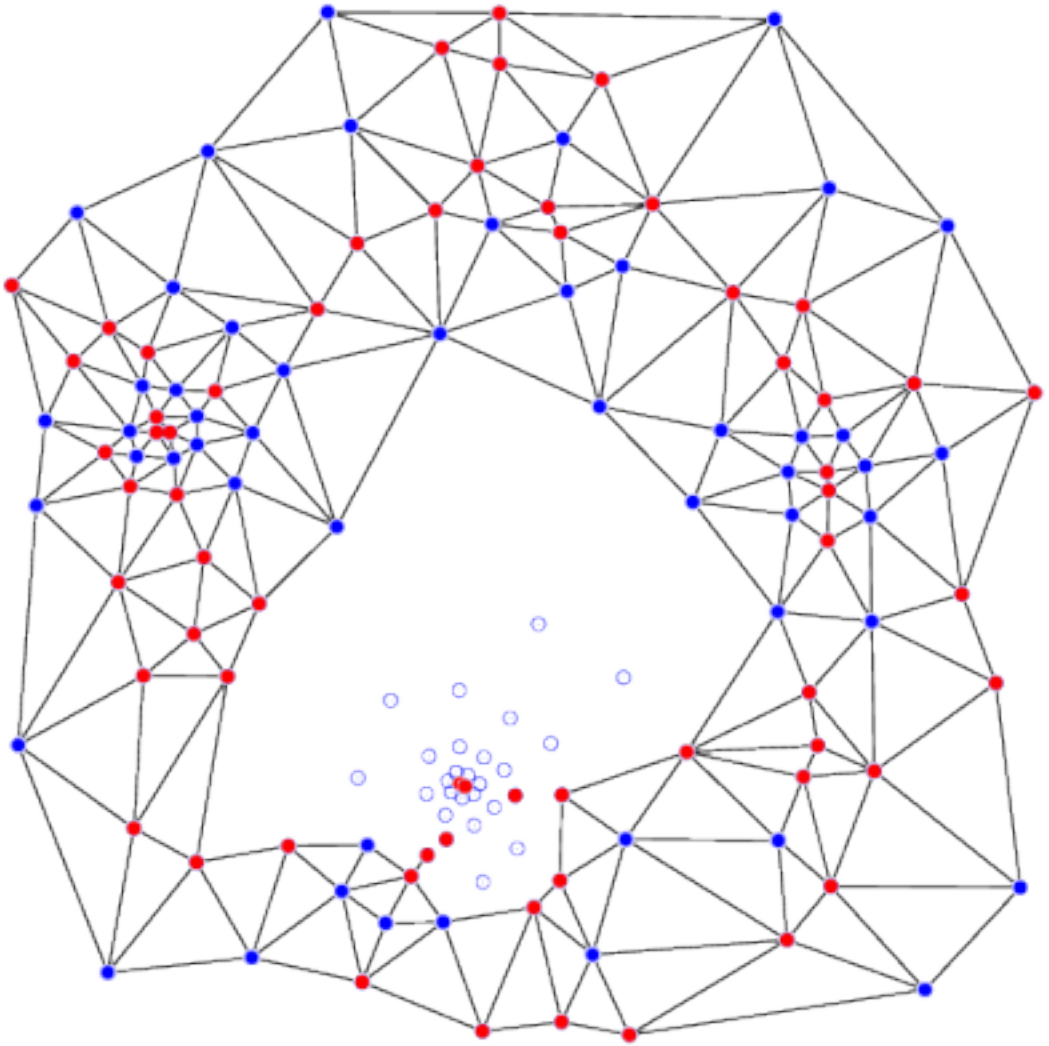
- Insertion and erasure of input points
- Result in *local* changes
- Maintain size-optimality of well-spaced superset



Well-spaced point sets

<u>Dynamic modifications</u>

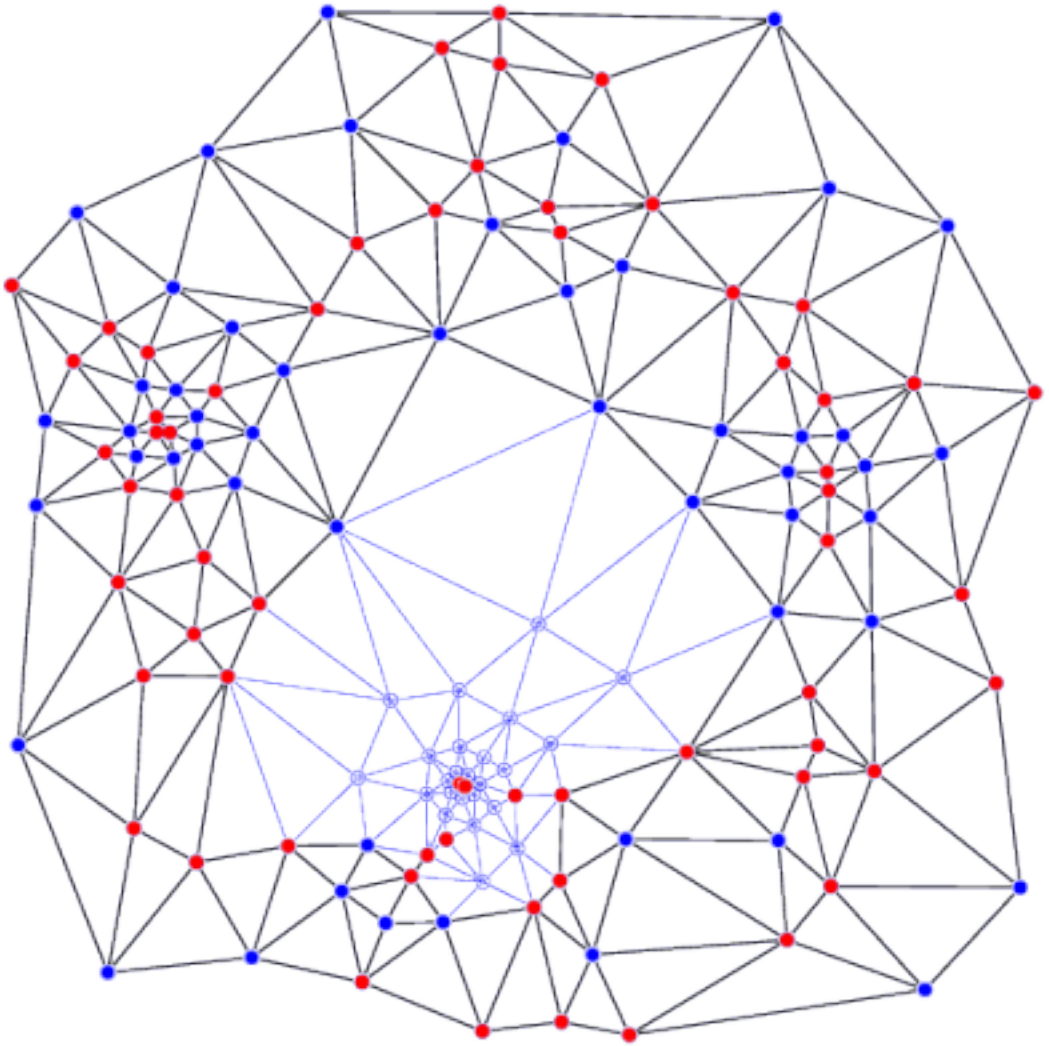
- Insertion and erasure of input points
- Result in *local* changes
- Maintain size-optimality of well-spaced superset



Well-spaced point sets

<u>Dynamic modifications</u>

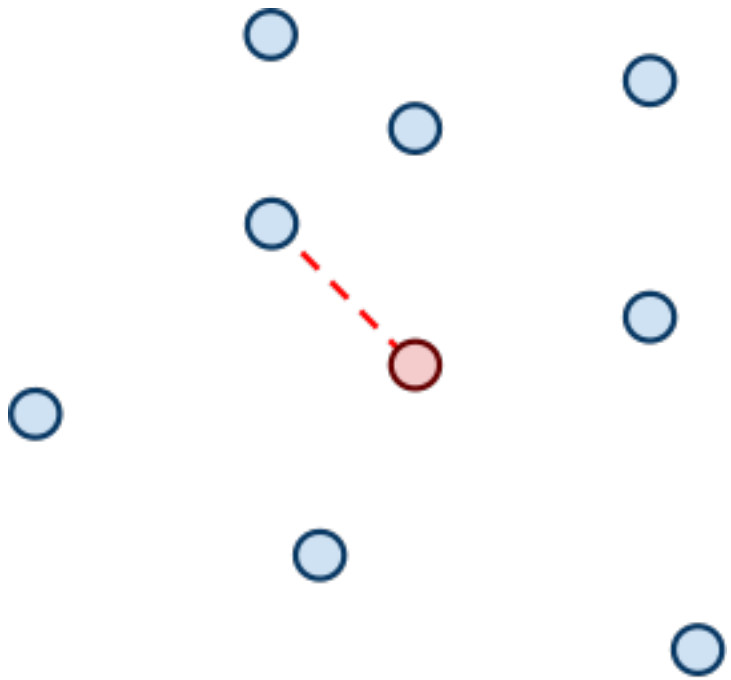
- Insertion and erasure of input points
- Result in *local* changes
- Maintain size-optimality of well-spaced superset



Well-spaced point sets

<u>Well-spacedness</u>

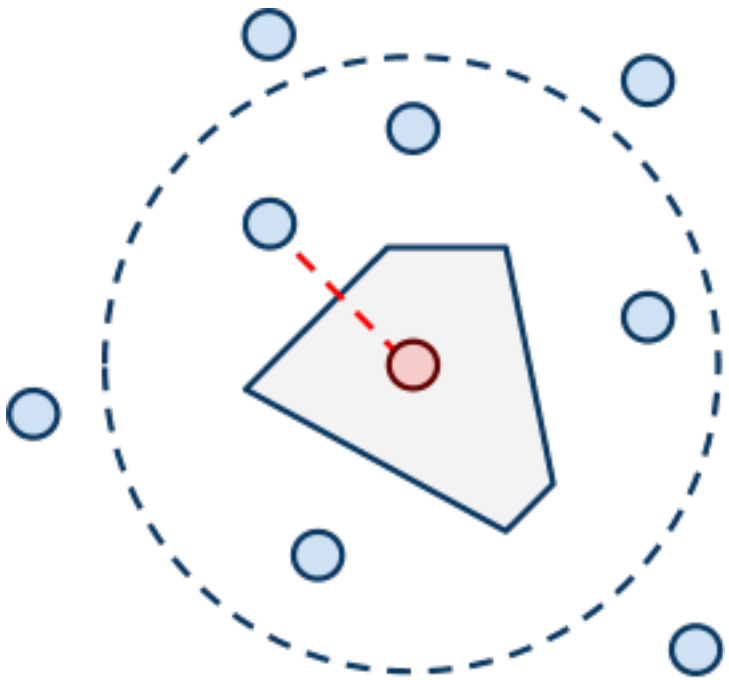
- For a point *v*, let *NN* be the distance to *v*'s nearest neighbor



Well-spaced point sets

<u>Well-spacedness</u>

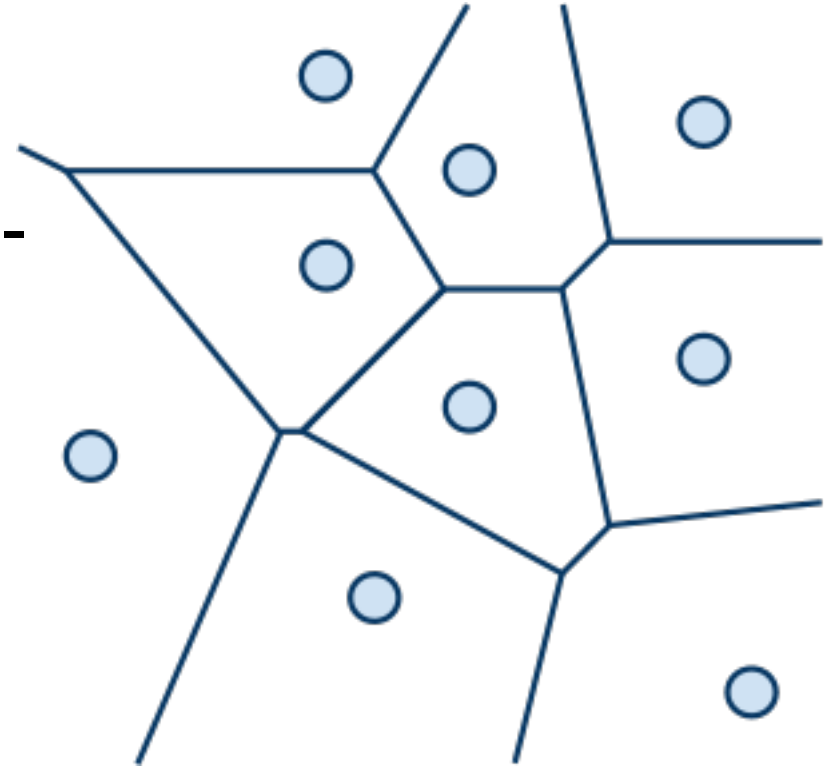
- For a point v, let NN be the distance to v's nearest neighbor
- We call v "ρ-well-spaced" [Talmor '97] if its Voronoi cell is contained in a ball, centered on v, of radius $\rho \cdot NN$
- $\rho > 1$ is a parameter



Well-spaced point sets

<u>Well-spacedness</u>

- A point set is well-spaced if all of its points are well-spaced
- Well-spaced point sets result in high-quality triangulations

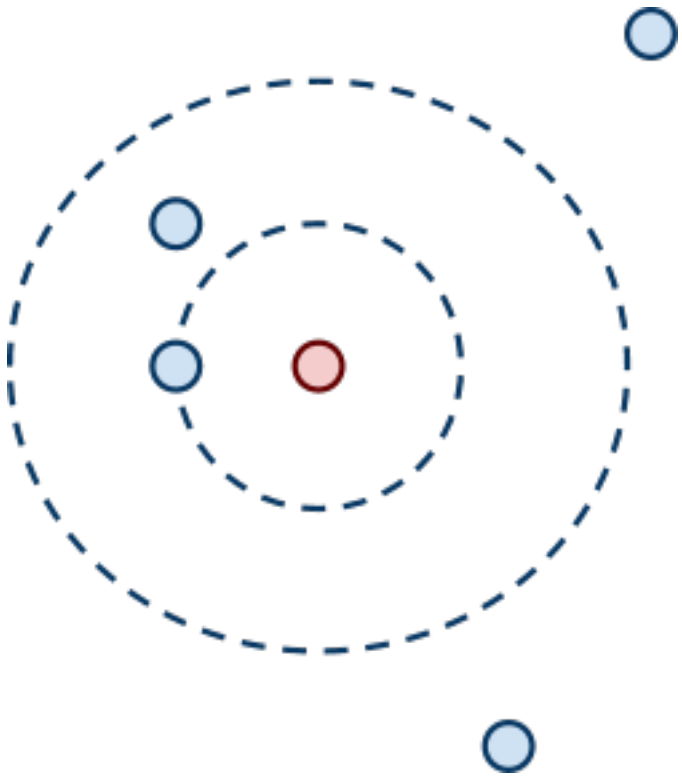


Our algorithm

<u>Fill operation</u>

- Find NN and $\rho \cdot NN$

.

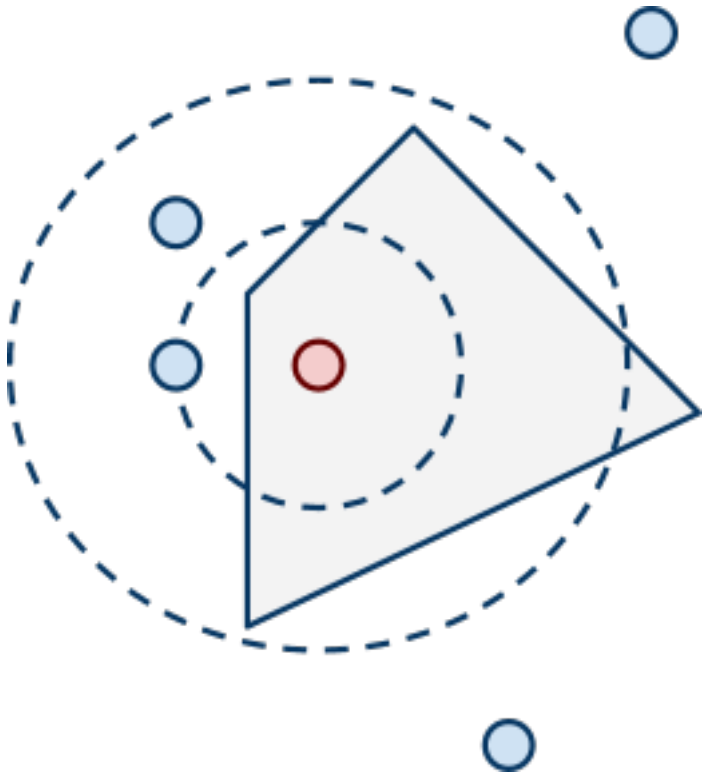


Our algorithm

<u>Fill operation</u>

- Find *NN* and $\rho \cdot NN$
- Find *v*'s Voronoi cell

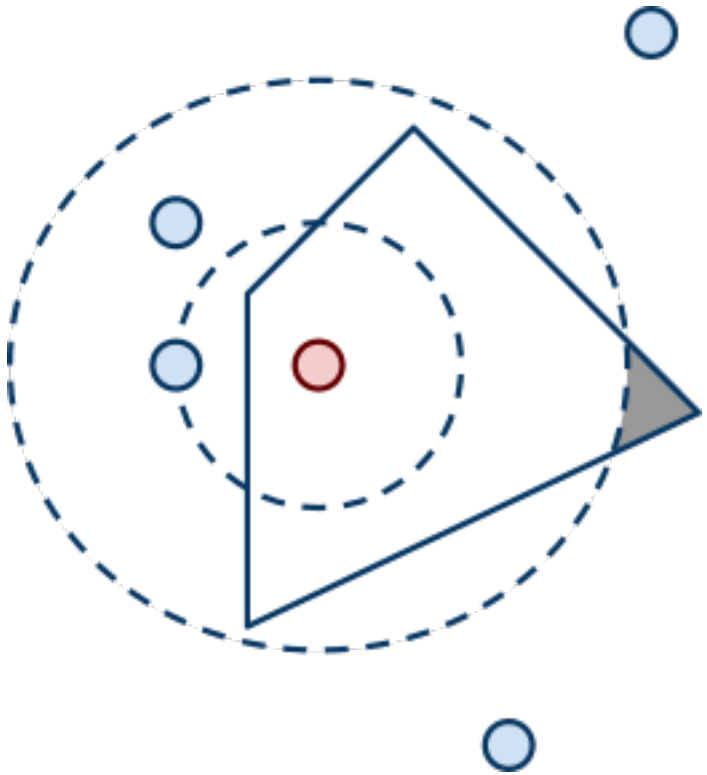
.



Our algorithm

<u>Fill operation</u>

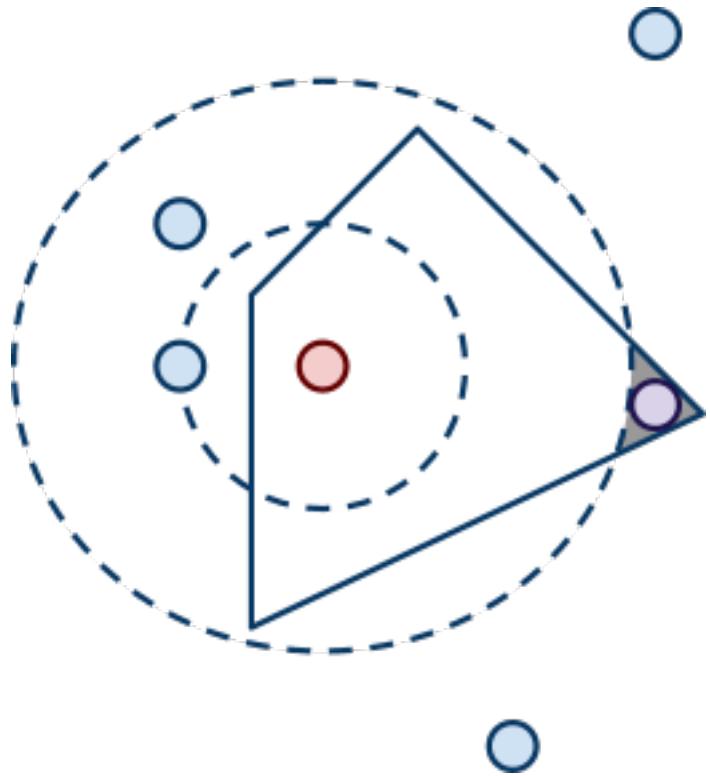
- Find *NN* and $\rho \cdot NN$
- Find *v*'s Voronoi cell
- Define the "picking region"



Our algorithm

<u>Fill operation</u>

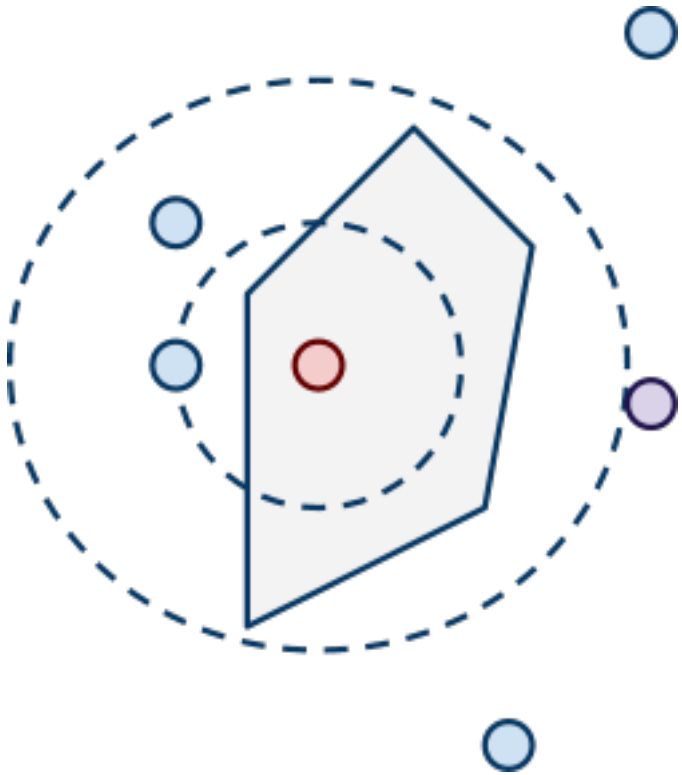
- Find *NN* and $\rho \cdot NN$
- Find *v*'s Voronoi cell
- Define the "picking region"
- Choose a Steiner point from the picking region



Our algorithm

<u>Fill operation</u>

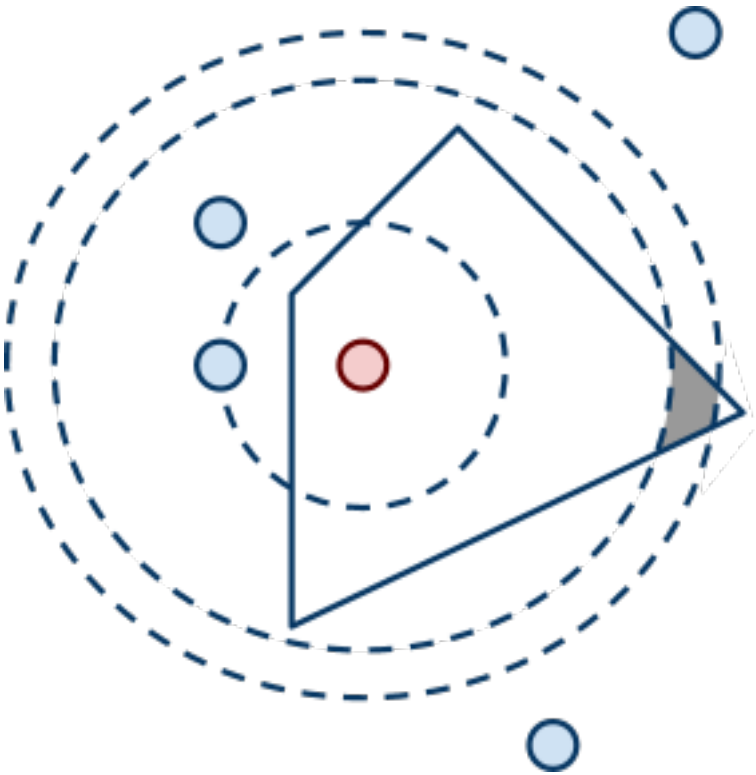
- Find *NN* and $\rho \cdot NN$
- Find *v*'s Voronoi cell
- Define the "picking region"
- Choose a Steiner point from the picking region
- Update the Voronoi cell
- Repeat if necessary



Our algorithm

<u>Improvement to fill operation</u>

- Restrict the picking region to a radius of $\beta \cdot NN$ for $\beta > \rho$
- This eliminates superfluous dependencies



Our algorithm

<u>Scheduling</u>

- The fill operation *directly* enforces well-spacedness
- Could repeatedly "fill" points until the point set is well-spaced
- Overall goals are to:
  - Insert a small number of Steiner points
  - Have a "good" computation graph
- To ensure these, fill operations must be scheduled carefully

Our algorithm

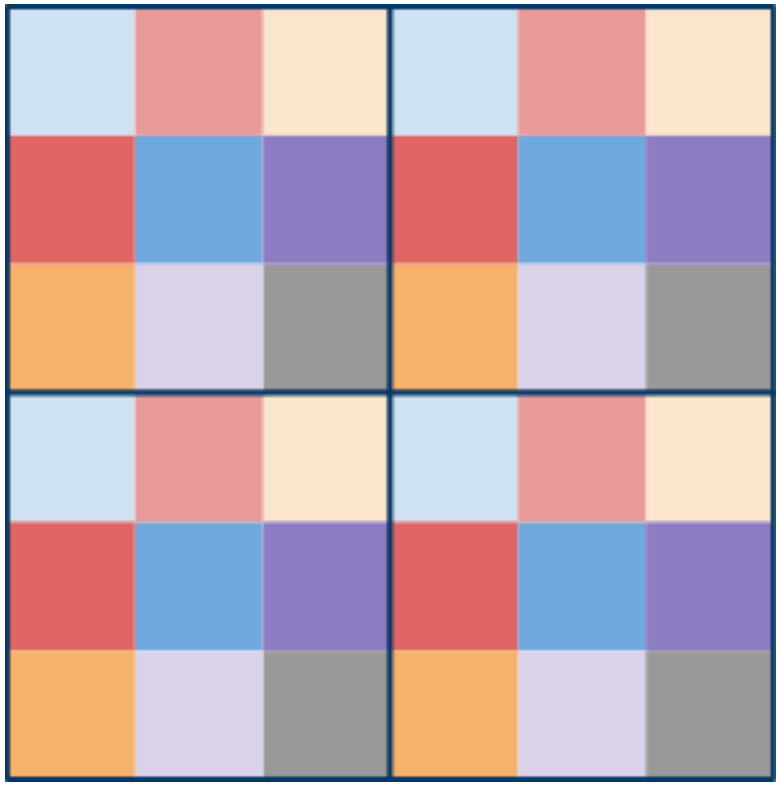
<u>Ranks</u>

- Define the "rank" of *v* as the (integer) base-*ρ* logarithm of *NN*
- Fill the points in rank order
  - When Steiner points are inserted, schedule them, and reschedule their neighbors
- Each point will be filled exactly once
- Output will be well-spaced and size-optimal
- The computation graph will *not* be "good"
  - Neither low depth nor narrowness

Our algorithm

<u>Coloring</u>

- Orders operations *within* a rank
- Repeating grid covers entire space
- Color of *v* is color of grid cell containing *v*



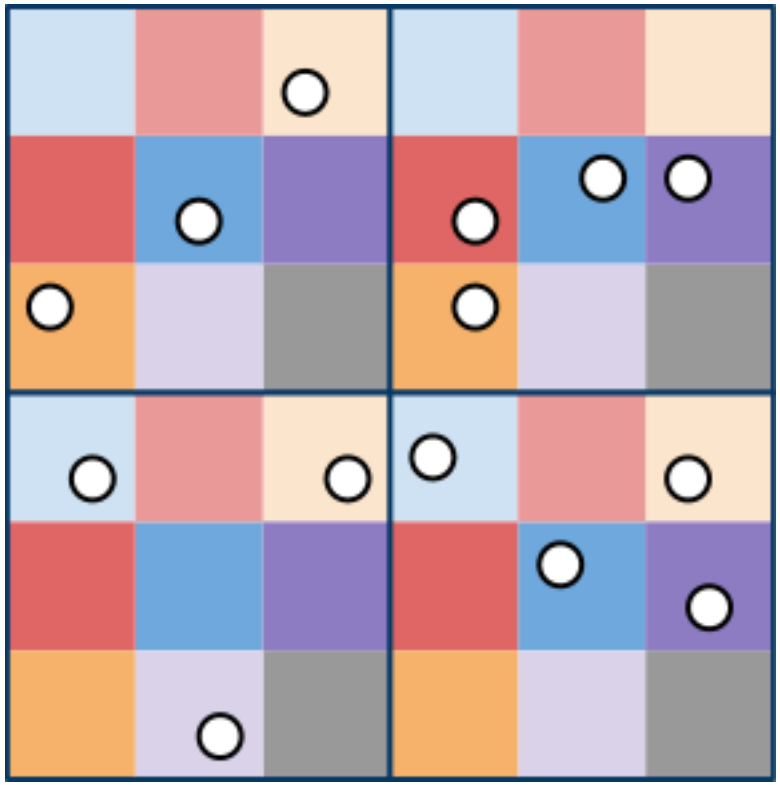
Our algorithm

<u>Coloring</u>

- Orders operations *within* a rank
- Repeating grid covers entire space
- Color of *v* is color of grid cell containing *v*

<u>Properties</u>

- No grid cell contains multiple points



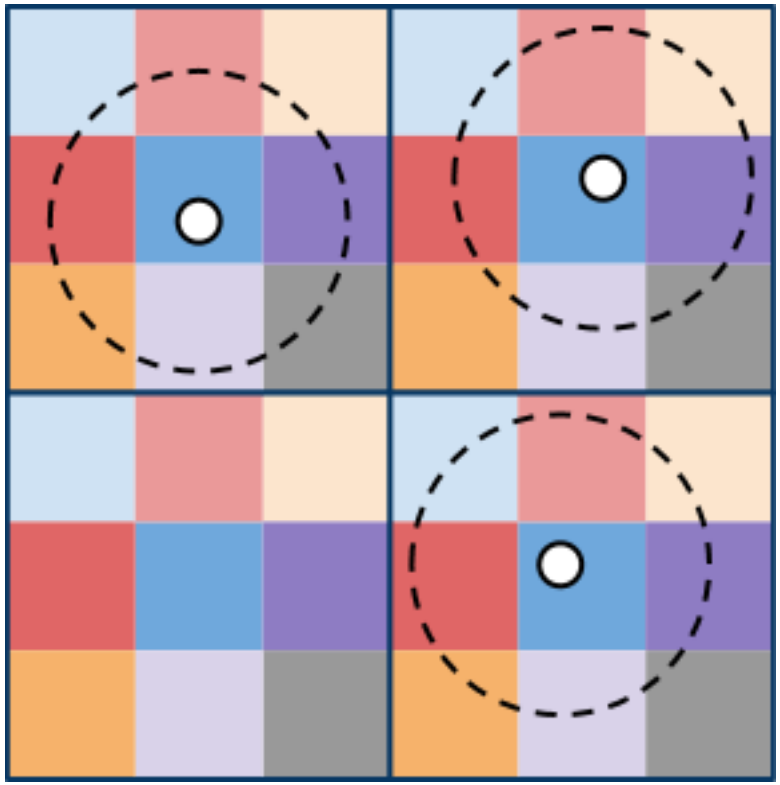
Our algorithm

<u>Coloring</u>

- Orders operations *within* a rank
- Repeating grid covers entire space
- Color of *v* is color of grid cell containing *v*

<u>Properties</u>

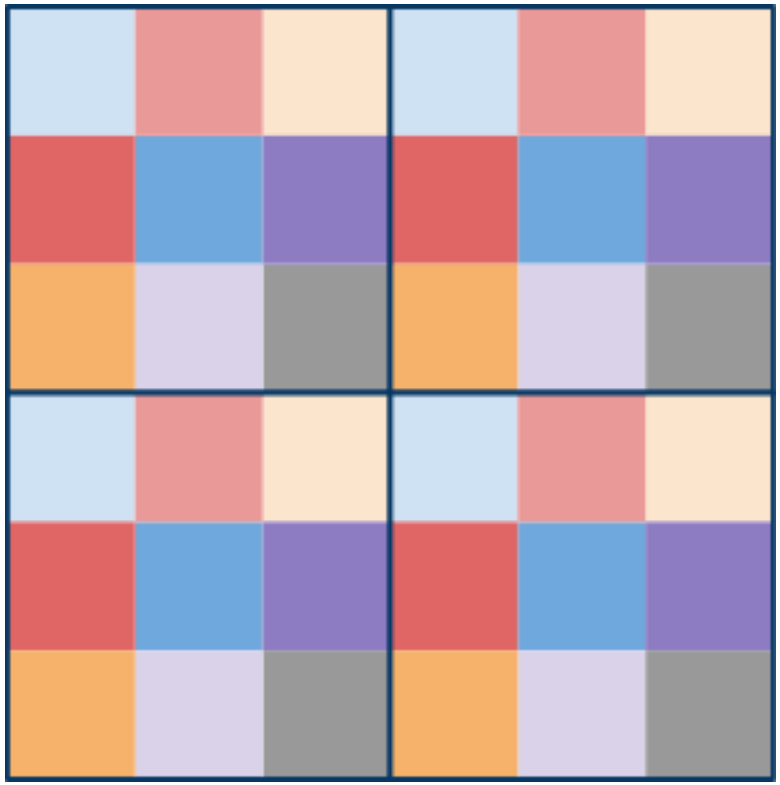
- No grid cell contains multiple points
- Fill operations scheduled at the same color are independent



Our algorithm

<u>Coloring parameters</u>

- Number of colors
- Side-length of grid squares
- Both depend only on ρ and β
- Number of colors is *constant*



Our algorithm

<u>Scheduling by rank and color</u>

- Order fill operations first by rank, then by color
  - Constant number of colors
  - Logarithmically many ranks
- Good dynamic and parallel performance
  - Parallel runtime is $\log(n)$
  - Work is $k \cdot \log(n)$, for k dynamic changes

Experiments

Dataset	***n***	***k***	**1 core**	**2 core**	**4 core**
New Zealand (2D)	18595	463	49×	74×	98×
Lake Superior (2D)	33487	296	102×	150×	189×
San Francisco (2D)	85910	322	196×	299×	408×
Bunny (3D)	35947	657	9.3×	16×	26×
Armadillo (3D)	172974	4242	14×	25×	42×

- Speedups are relative to recomputing from scratch
- *n* is dataset size, *k* is changeset size

Related work

<u>Well-spaced point sets</u>

- Chew '89 (first non-heuristic construction algorithm)
- Ruppert '95 (size-optimal in 2D)
- Har-Peled, Üngör '05 (efficient in 2D)
- Hudson, Miller, Phillips '06 (efficient in any D)
- Hudson, Miller, Phillips '07 (parallel version of above)

<u>Direct influences</u>

- Spielman, Teng, Üngör '02 (parallel)
- Acar, Cotter, Hudson, Türkoğlu '10 (dynamic)

Summary

<u>Our findings</u>

- An appealing connection between dynamic and parallel algorithms
- An efficient parallel change-propagation algorithm for the well-spaced point set problem
- Results demonstrating that dynamic and parallel algorithms interact well in practice

Summary

<u>In our paper</u>

- Description of parallel dynamic algorithm targeting EREW PRAM model
- Proofs of size-optimality and performance guarantees
- Description of implementation on real-world multi-core hardware
- More detailed experimental evaluation

Thank you