26^{th} **Symposium on Computational Geometry**

Dynamic Well-Spaced Point Sets

Umut A. Acar[a]

Andrew Cotter[b]

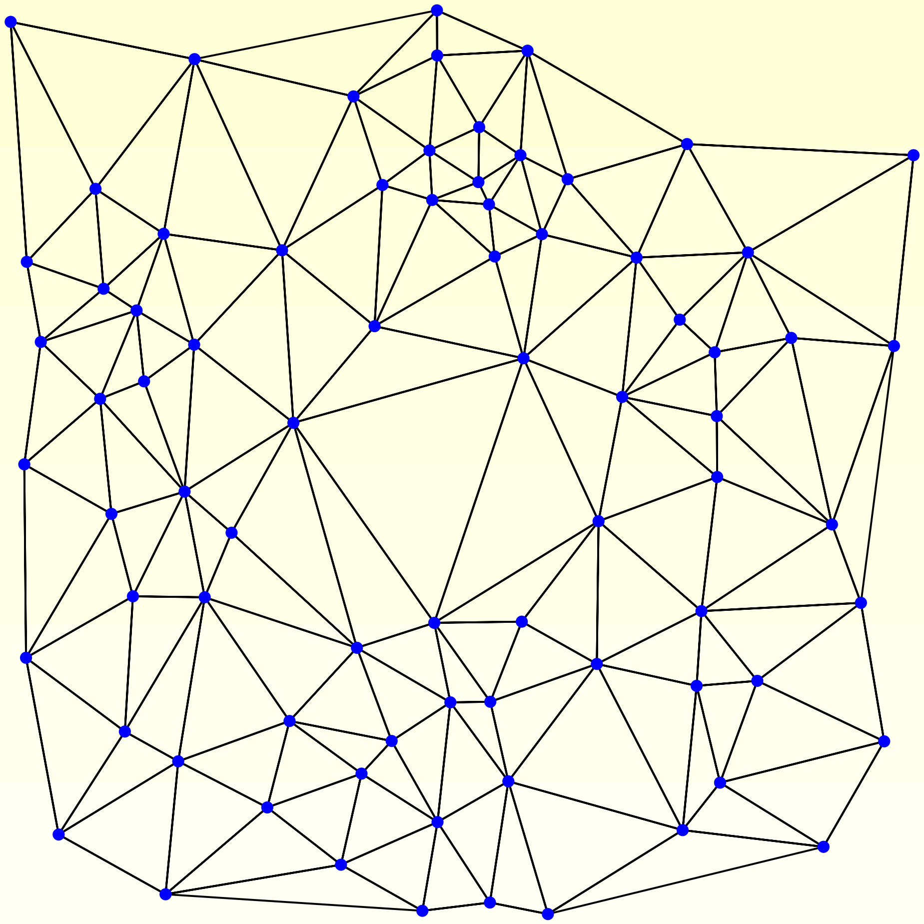
Benoît Hudson[b]

Duru Türkoğlu[c]

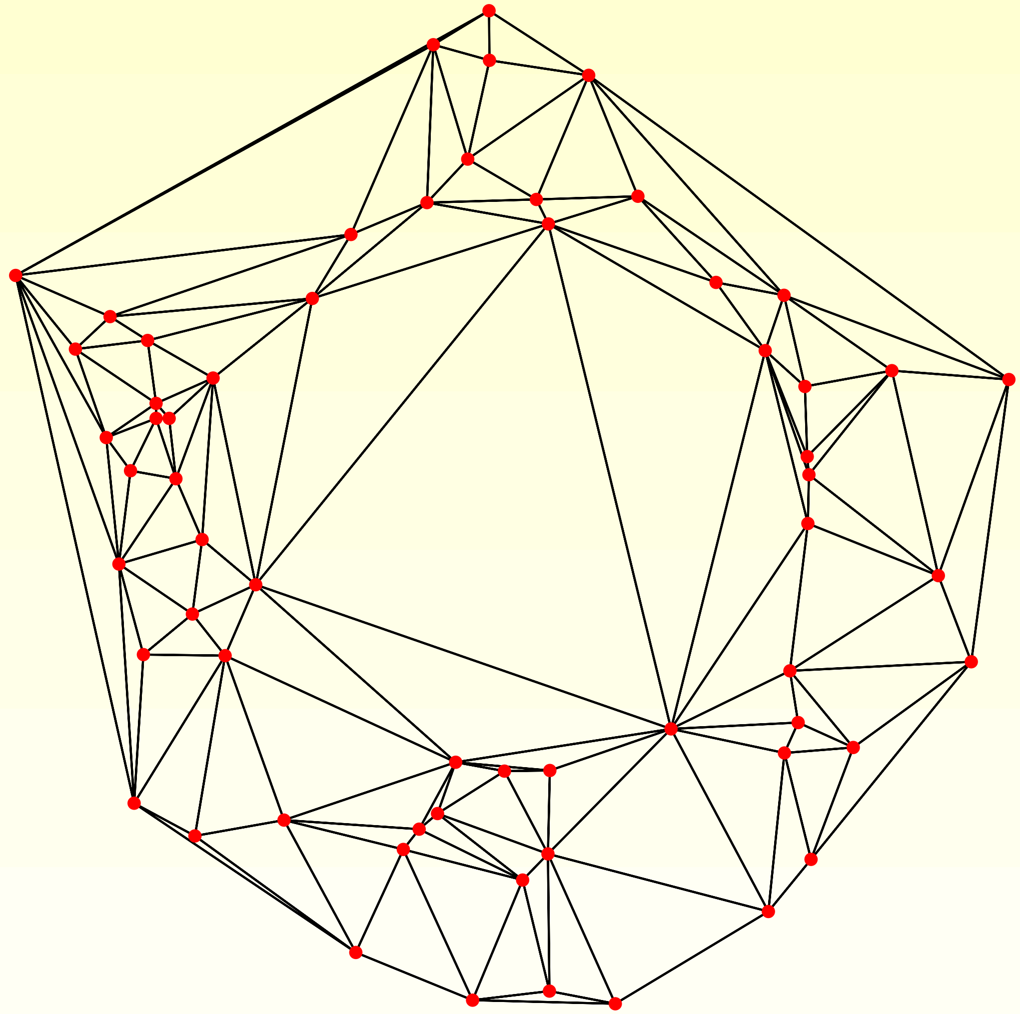
June 4, 2011

[a]Max Planck Institute (SWS)
[b]Toyota Technological Institute at Chicago
[c]University of Chicago

Delaunay Triangulations, 2D Examples



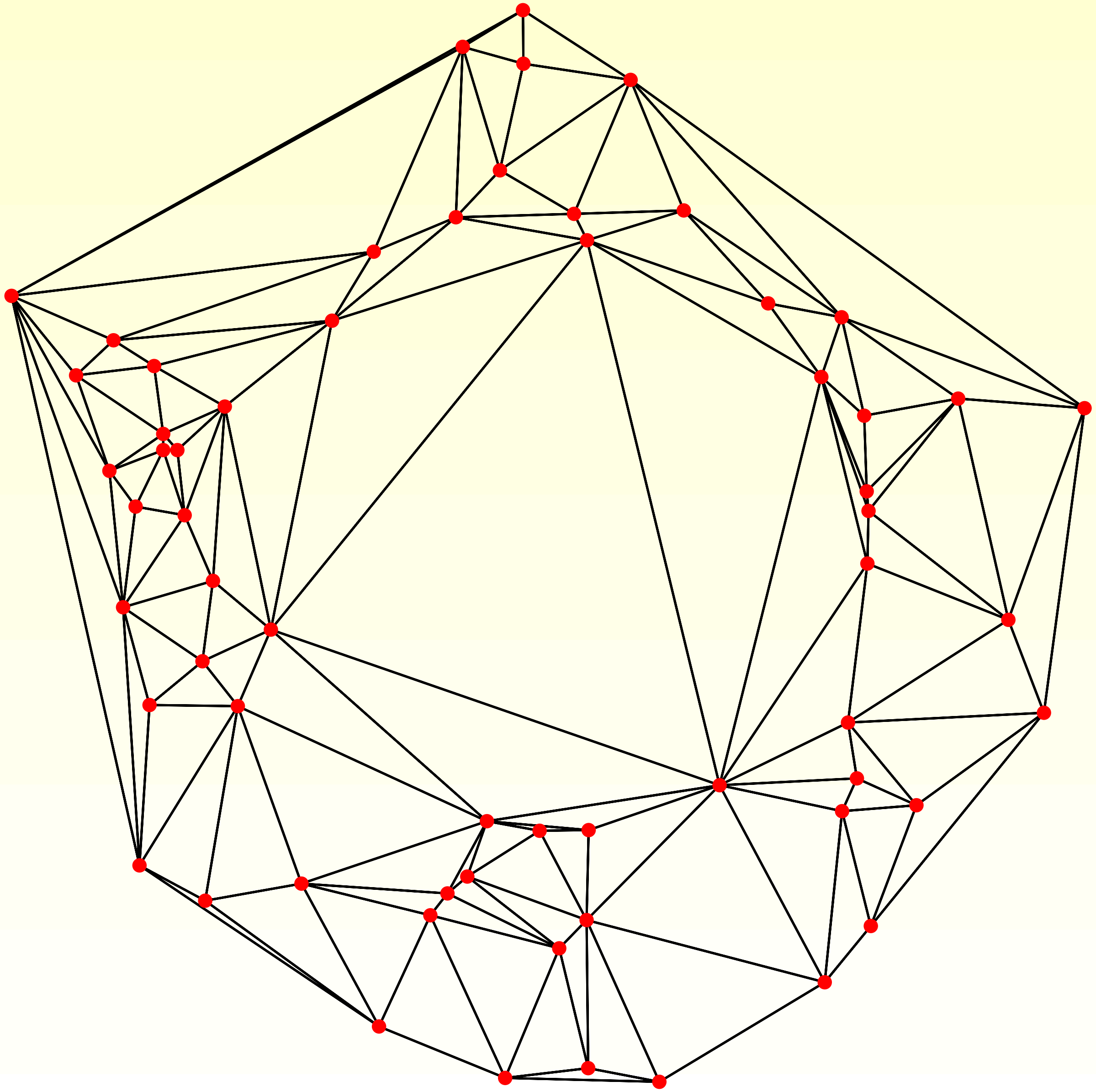
No small angles



Skinny triangles

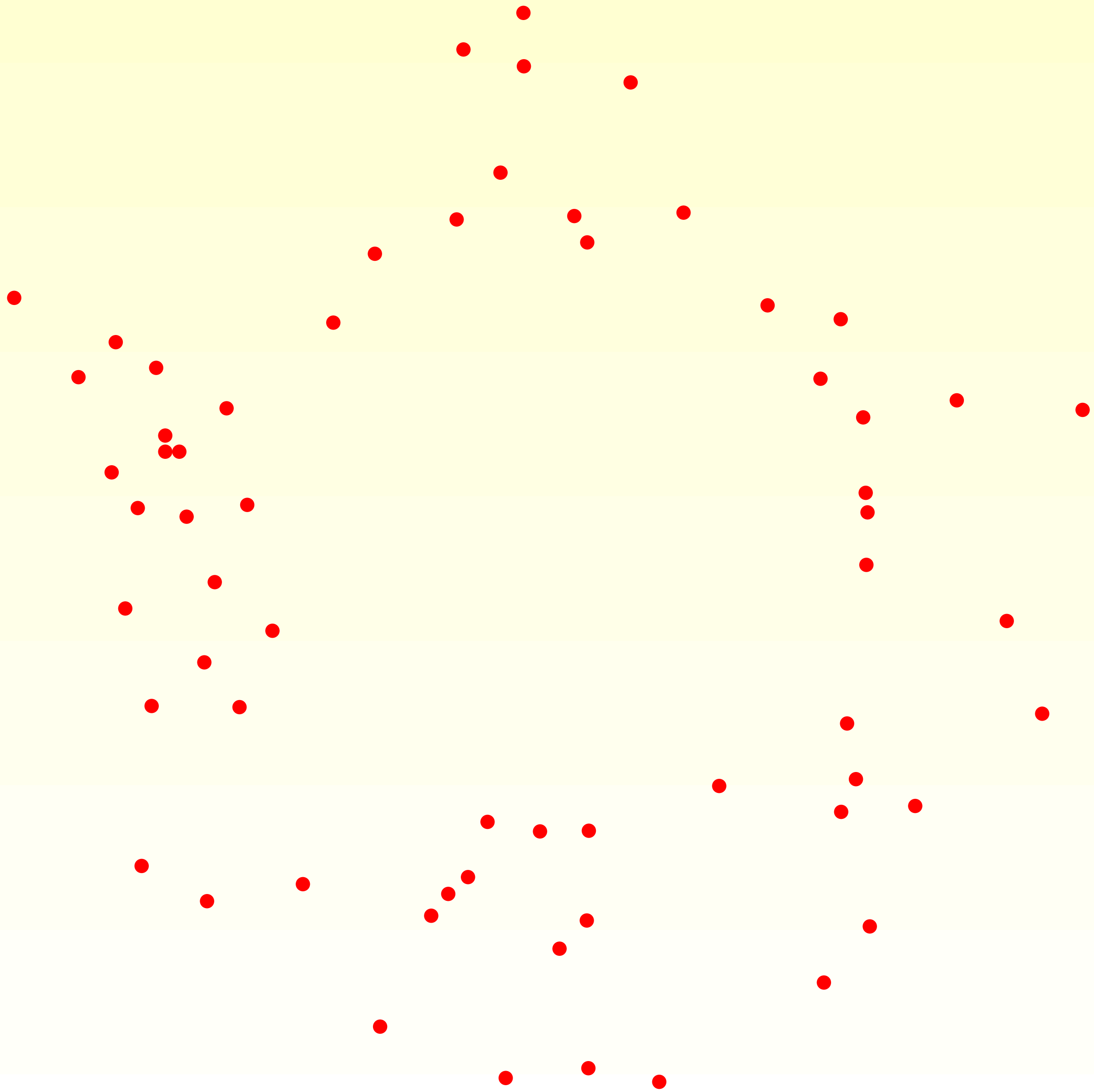
Mesh Refinement

Input set of points does not have a good triangulation



Mesh Refinement

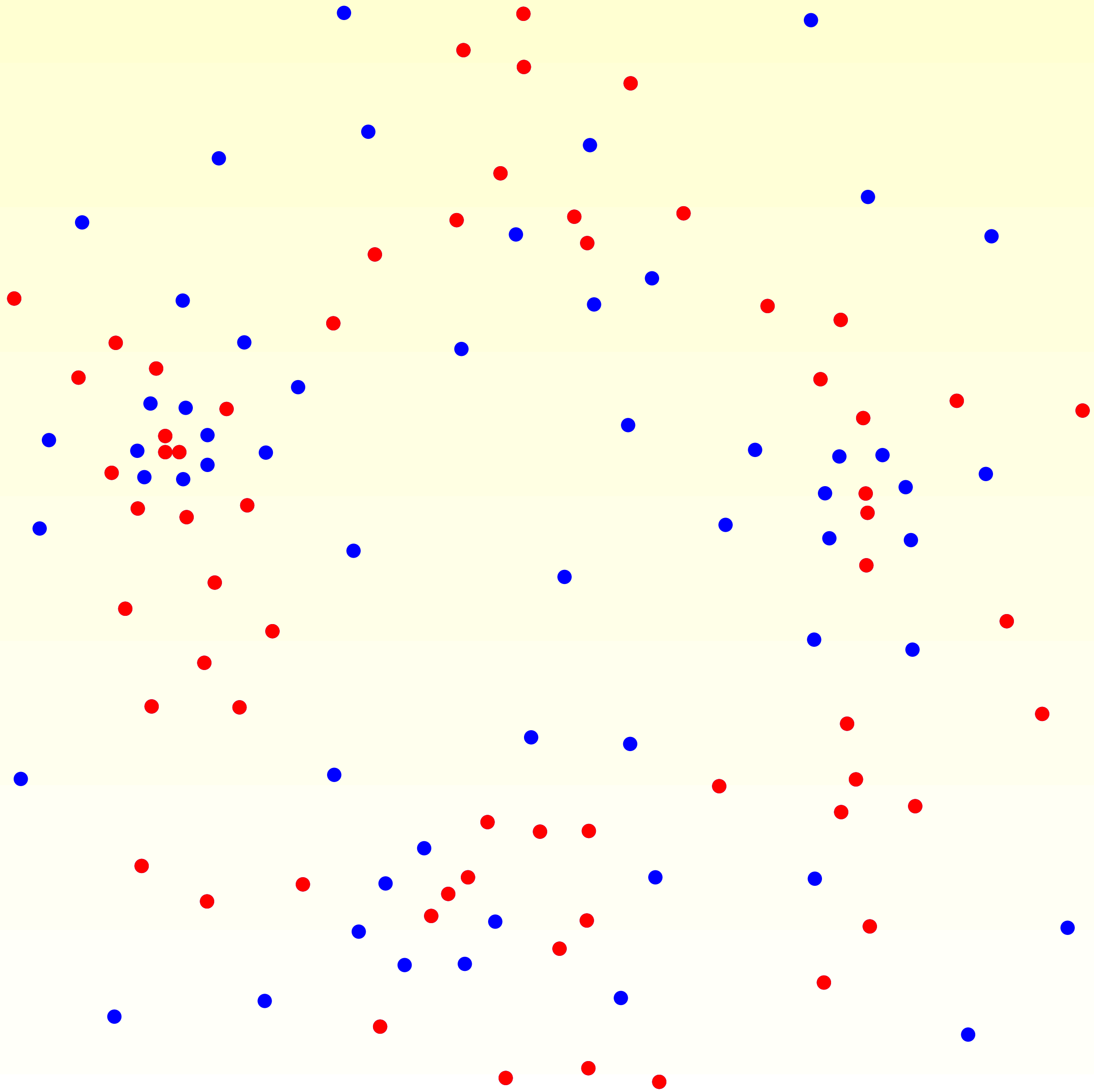
Input set of points does not have a good triangulation



Mesh Refinement

Input set of points does not have a good triangulation

We need to insert Steiner points

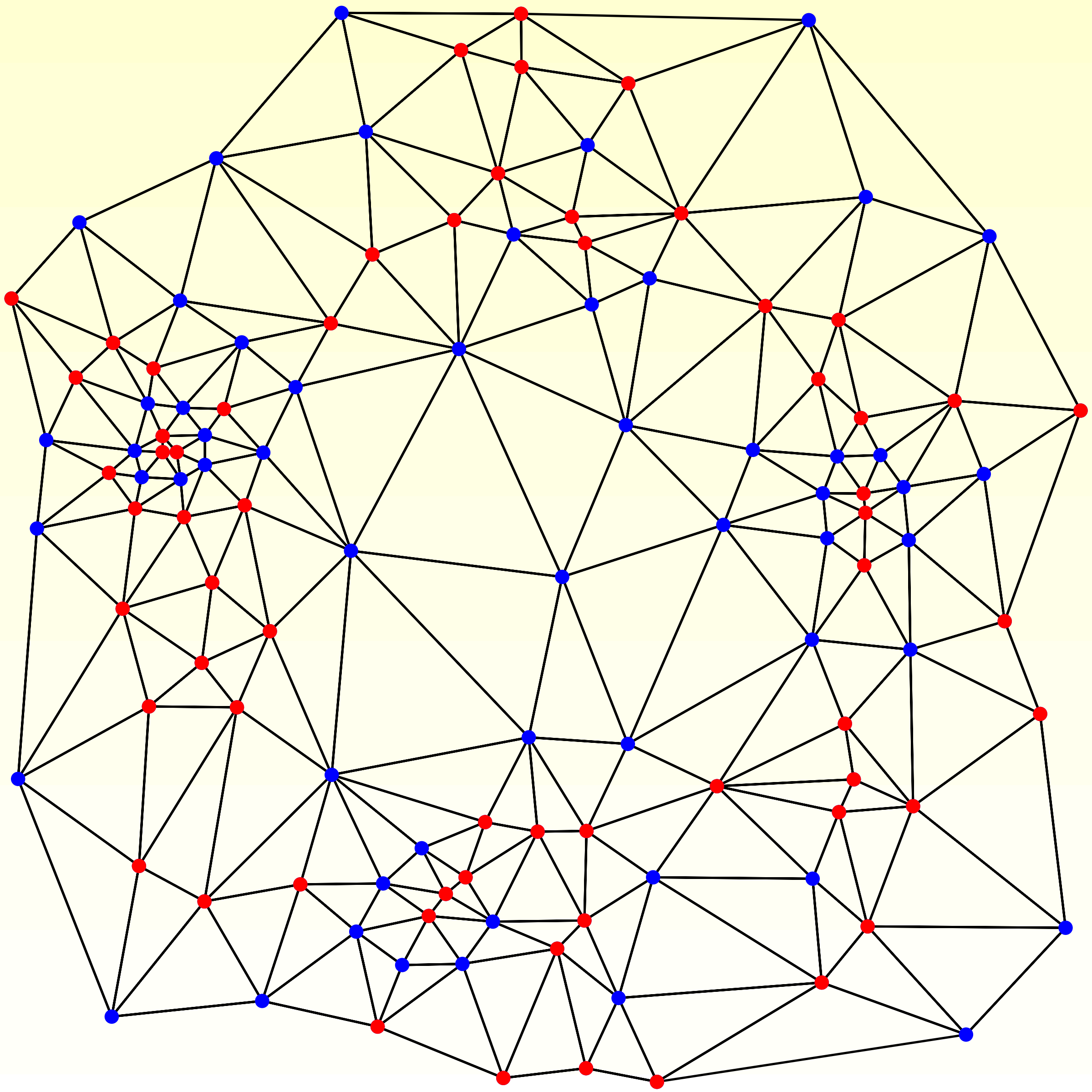


Mesh Refinement

Input set of points does not have a good triangulation

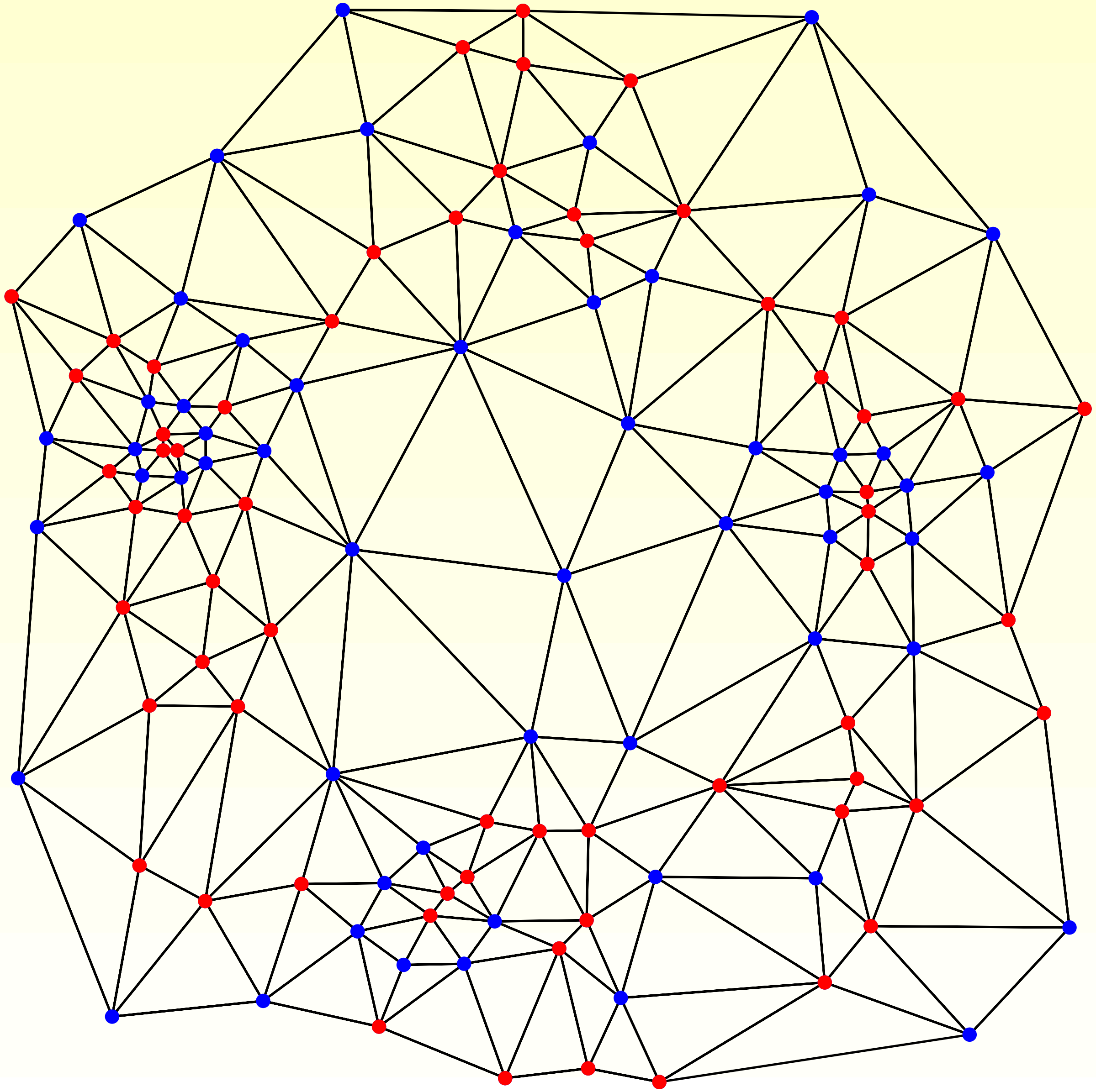
We need to insert Steiner points

The output has a quality Delaunay triangulation, i.e., no small angles



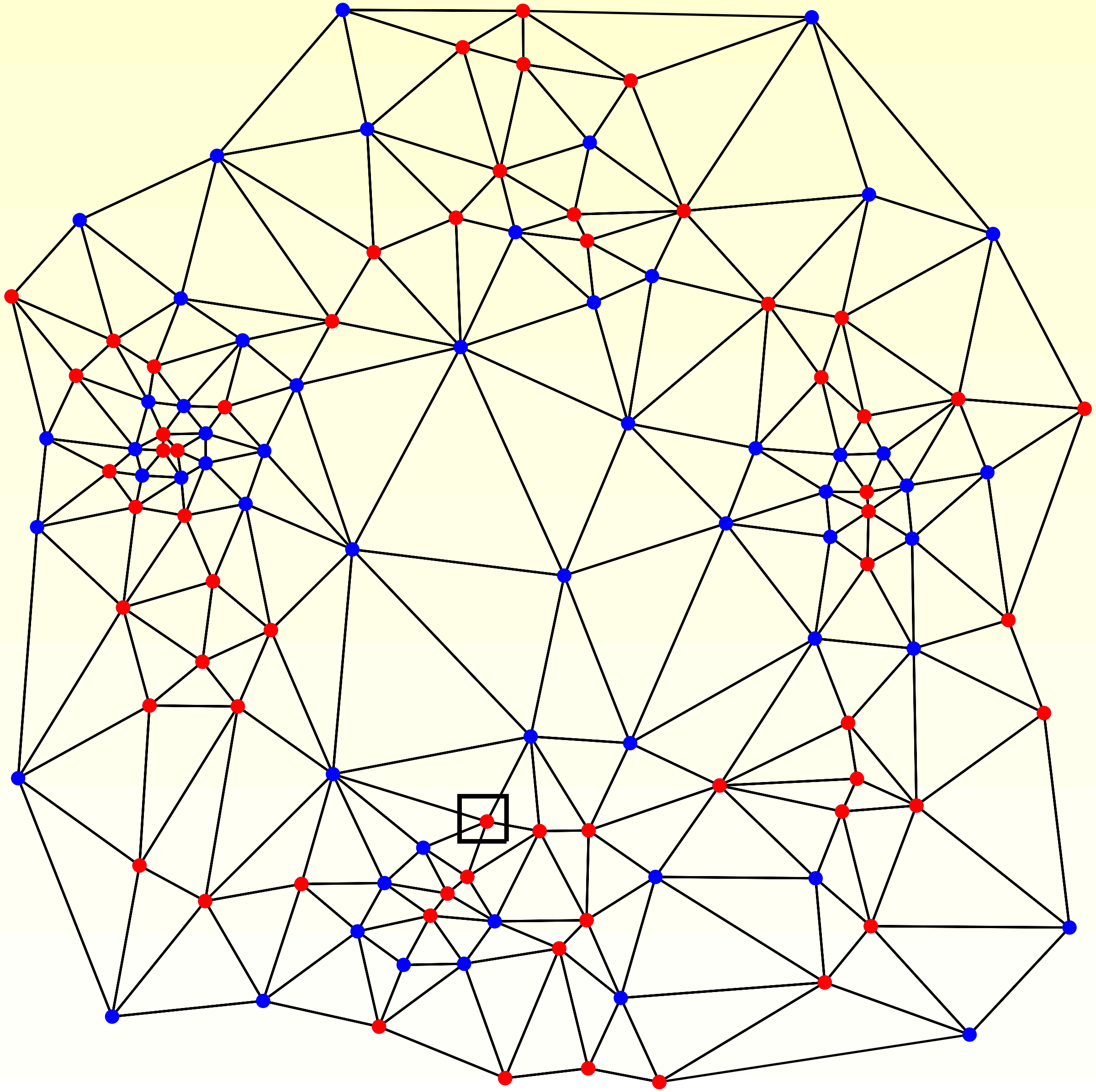
Dynamic Mesh Refinement

Assume somebody wants to change the input set



Dynamic Mesh Refinement

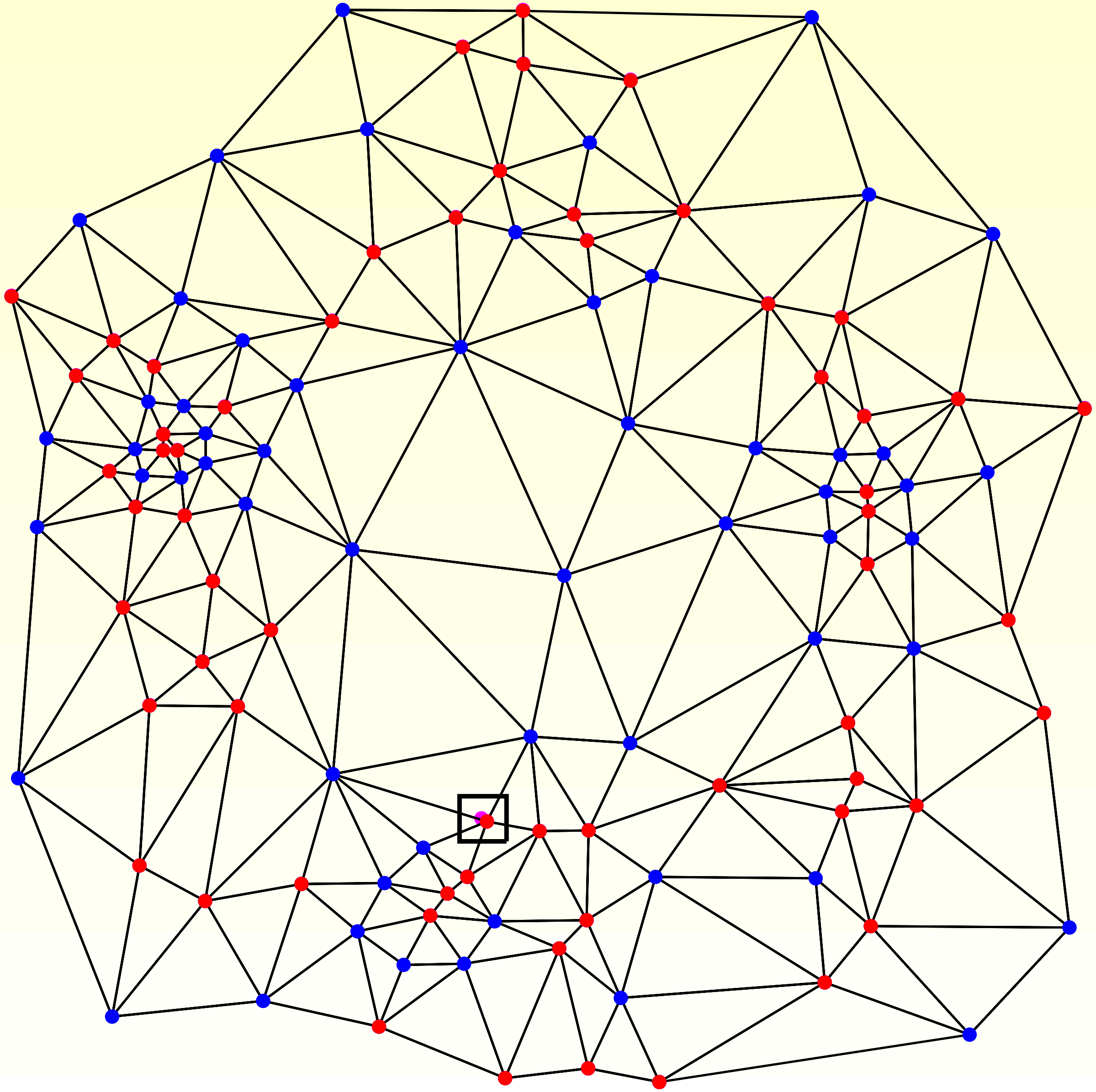
Assume somebody wants to change the input set



Dynamic Mesh Refinement

Assume somebody wants to change the input set

Update the set of Steiner points efficiently, without too many changes

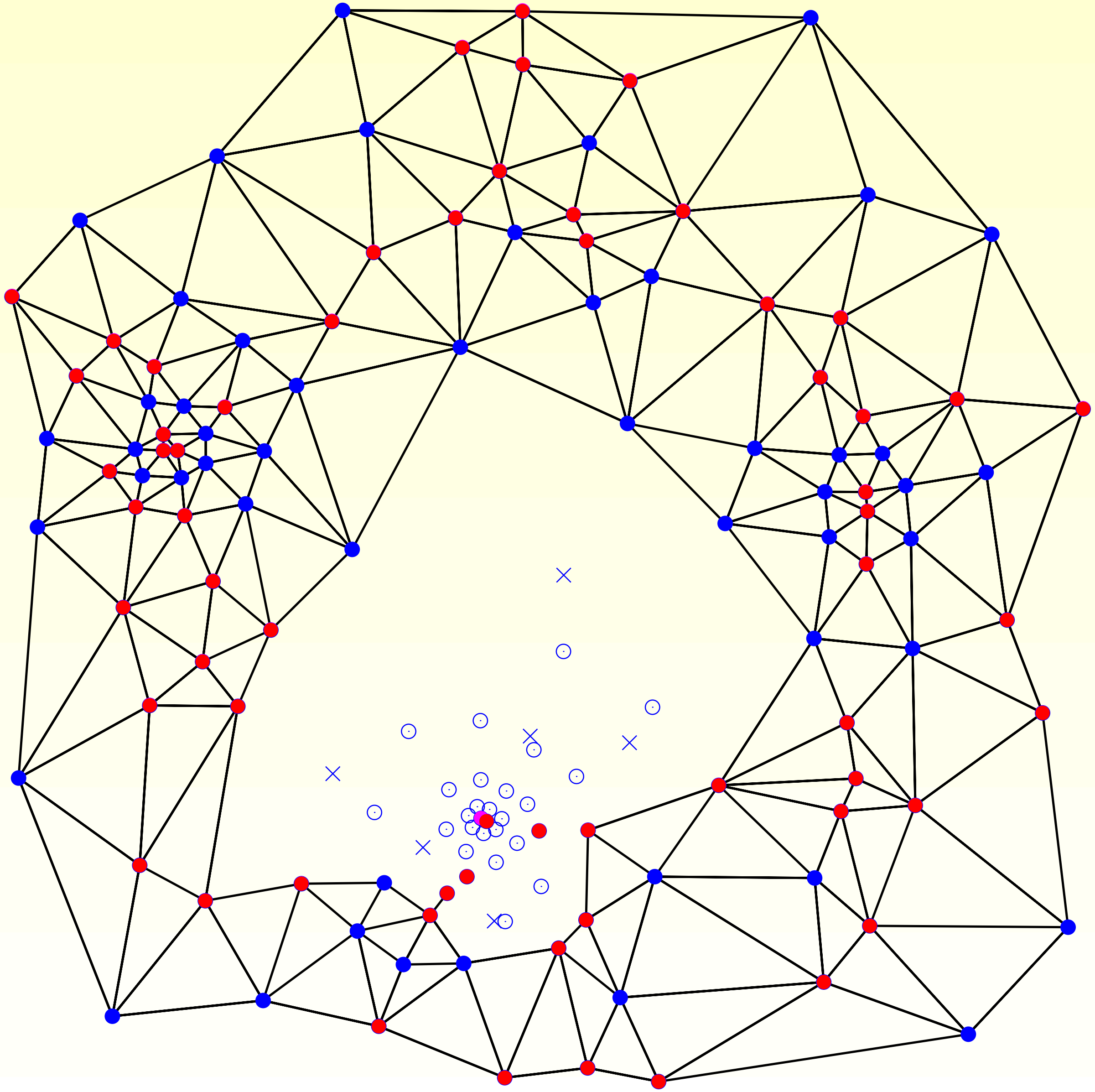


Dynamic Mesh Refinement

Assume somebody wants to change the input set

Update the set of Steiner points efficiently, without too many changes

Sustain output size to be small

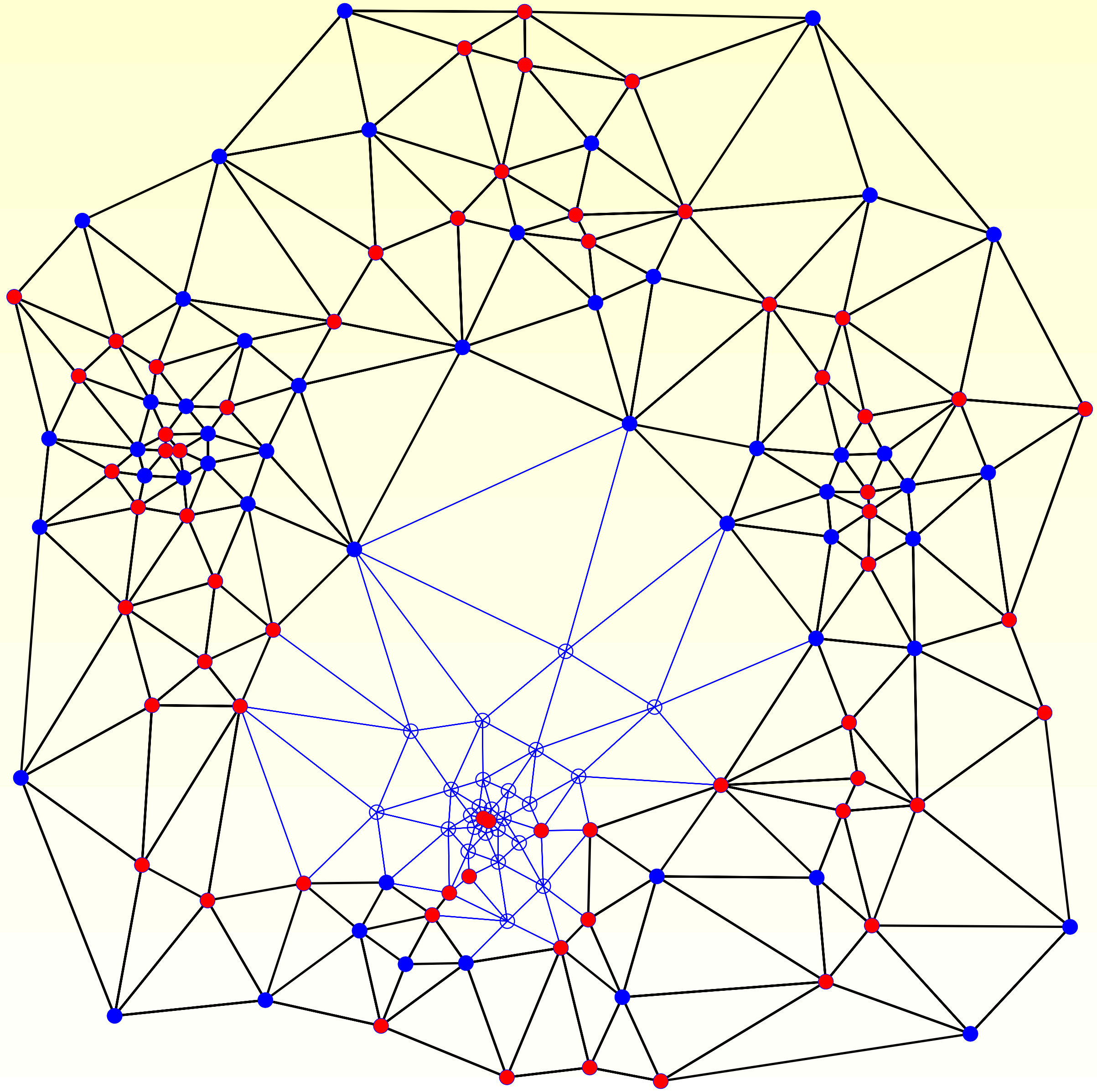


Dynamic Mesh Refinement

Assume somebody wants to change the input set

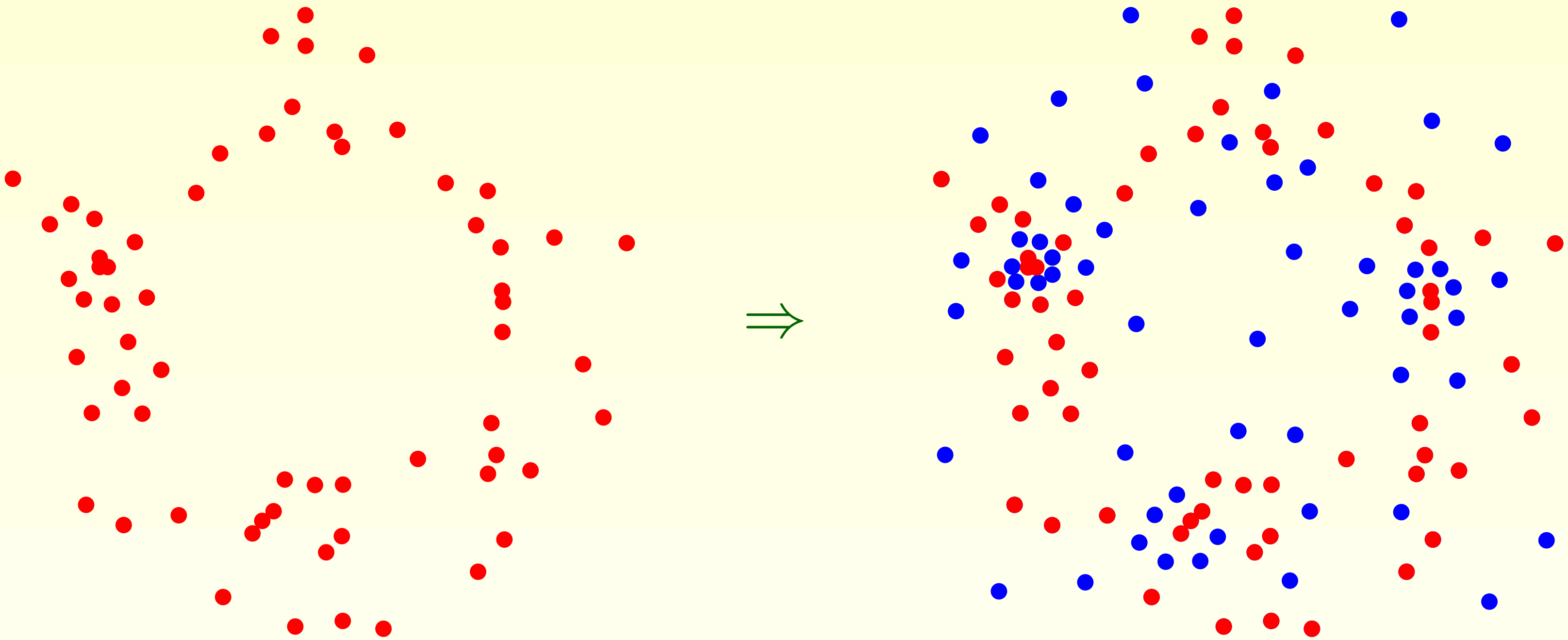
Update the set of Steiner points efficiently, without too many changes

Sustain output size to be small



Well-Spaced Point Set Problem

Compute a ρ-well-spaced superset of a given input



Maintain a ρ-well-spaced superset as input changes

Objectives: Efficiency and Size-Optimality

Related Work

- Chew '89 (First non-heuristic construction algorithm)
- Ruppert '95 (Size-optimal algorithm in 2D)
- Spielman, Teng, Üngör '02 (Parallel algorithm in 2D & 3D)
- Har-Peled, Üngör '05 (Efficient algorithm in 2D)
- Hudson, Miller, Phillips '06 (Efficient in arbitrary D)
- Hudson, T '08 (Precursor of this work, arbitrary D)

Existing Approaches for the Dynamic Problem

1) Update is efficient but updated output is not size-optimal

2) Worst case update time is as bad as running from scratch

Our Results

Construction Algorithm

- Given N and $\rho > 1$, output $M \supset N$ is ρ-well-spaced
- Output is **size-optimal** w.r.t. N, i.e. $|M| = O(|M_{OPT}|)$
- Runs in $O(n \log \Delta)$ time, $\Delta = \frac{\text{diameter}}{\text{smallest distance}}$

Dynamic Update Algorithm

- Given an insertion/deletion $(N \to N')$, modifies the output
- Modified output M' is **size-optimal** w.r.t. N'
- Update in $O(\log \Delta)$ time $\Rightarrow |M' \ominus M| = O(\log \Delta)$
- Worst case lower bound requires $|M' \ominus M| = \Omega(\log \Delta)$

Our Solution — Change Propagation

Stable Algorithm: Executions with similar inputs should produce similar outputs/intermediate data

$$
\begin{array}{ccc}
{\color{green}N} & \xrightarrow{\varepsilon\text{-modification}} & {\color{green}N'} \\
{\color{red}\Downarrow} & \longrightarrow & {\color{red}\Downarrow} \\
{\color{red}M} & \xrightarrow{\varepsilon\text{-change}} & {\color{red}M'}
\end{array}
$$

Our Solution — Change Propagation

Stable Algorithm: Executions with similar inputs should produce similar outputs/intermediate data

$$
\begin{array}{ccc}
N & \stackrel{\varepsilon\text{-modification}}{\longrightarrow} & N' \\
\Downarrow & \longrightarrow & \Downarrow \\
M & \stackrel{\varepsilon\text{-change}}{\longrightarrow} & M'
\end{array}
$$

Offline Algorithm $\Rightarrow$ Quality

Our Solution — Change Propagation

Stable Algorithm: Executions with similar inputs should produce similar outputs/intermediate data

$$
\begin{array}{ccc}
N & \xrightarrow{\varepsilon\text{-modification}} & N' \\
\Downarrow & \longrightarrow & \Downarrow \\
M & \xrightarrow{\varepsilon\text{-change}} & M'
\end{array}
$$

$$\textbf{Offline Algorithm} \Rightarrow \textbf{Quality}$$

$$+$$

$$\textbf{Stability} \Rightarrow \begin{cases} \textbf{Executions are similar} \\ M' \ominus M \textbf{ is small} \end{cases}$$

Our Solution — Change Propagation

Stable Algorithm: Executions with similar inputs should produce similar outputs/intermediate data

$$\begin{array}{ccc} N & \xrightarrow{\varepsilon\text{-modification}} & N' \\ \Downarrow & \longrightarrow & \Downarrow \\ M & \xrightarrow{\varepsilon\text{-change}} & M' \end{array}$$

Offline Algorithm $\Rightarrow$ **Quality**

$+$

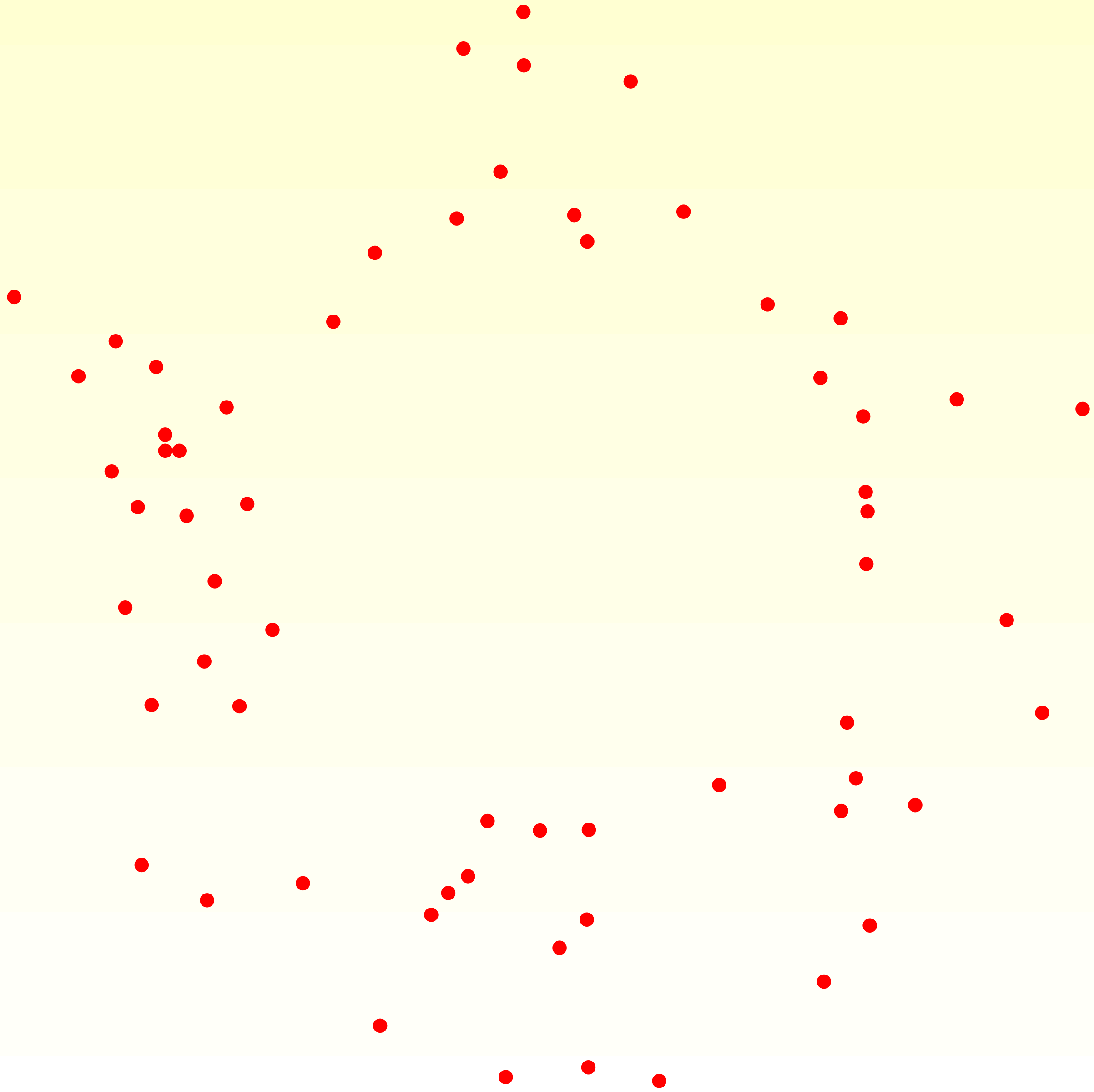
$$\textbf{Stability} \Rightarrow \begin{cases} \textbf{Executions are similar} \\ M' \ominus M \textbf{ is small} \end{cases}$$

$+$

Dynamizing Stable Algorithm $\Rightarrow$ **Efficiency**

Stable Refinement Algorithm

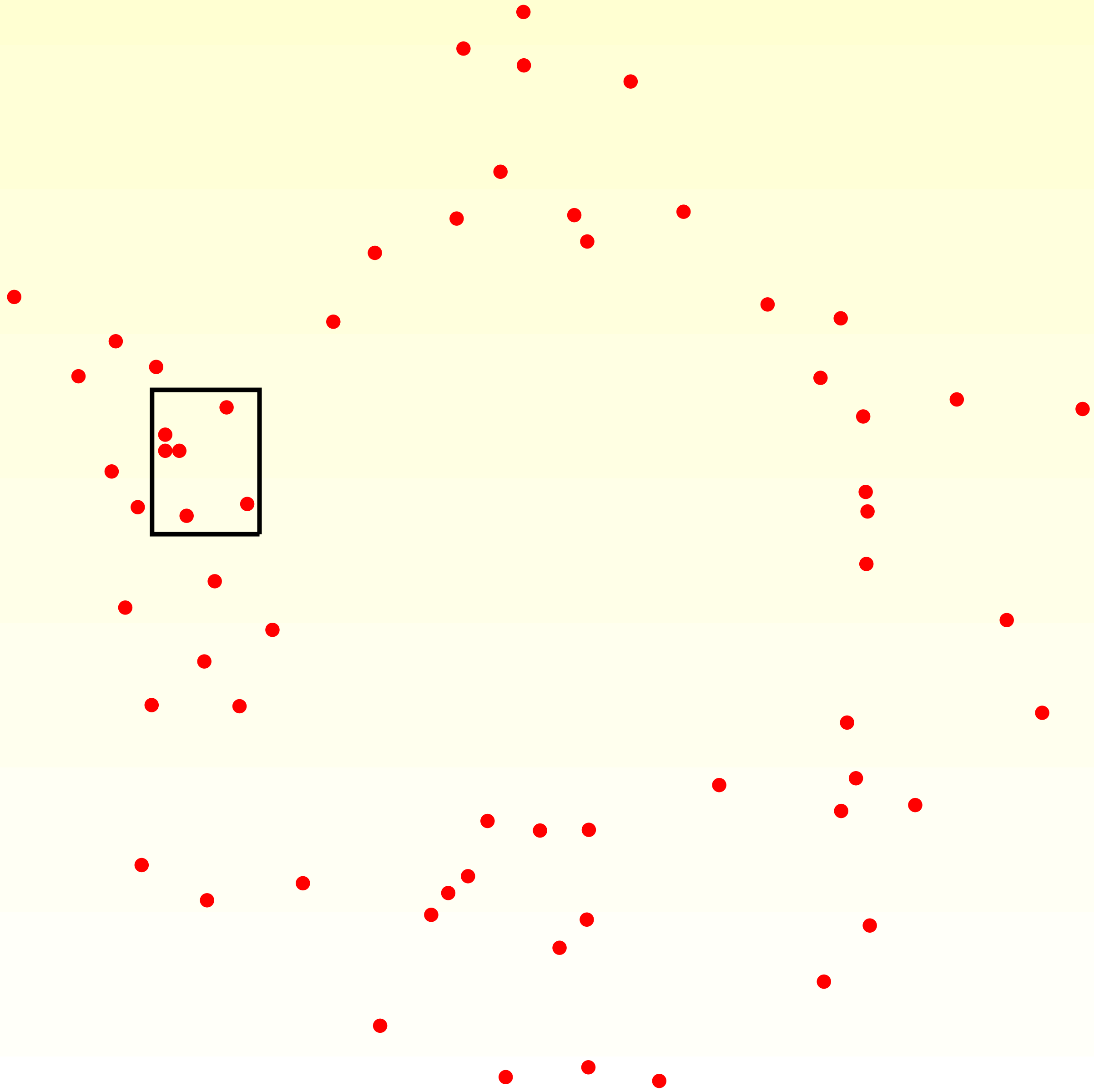
Basic operation of the construction algorithm is fill



Stable Refinement Algorithm

Basic operation of the construction algorithm is fill

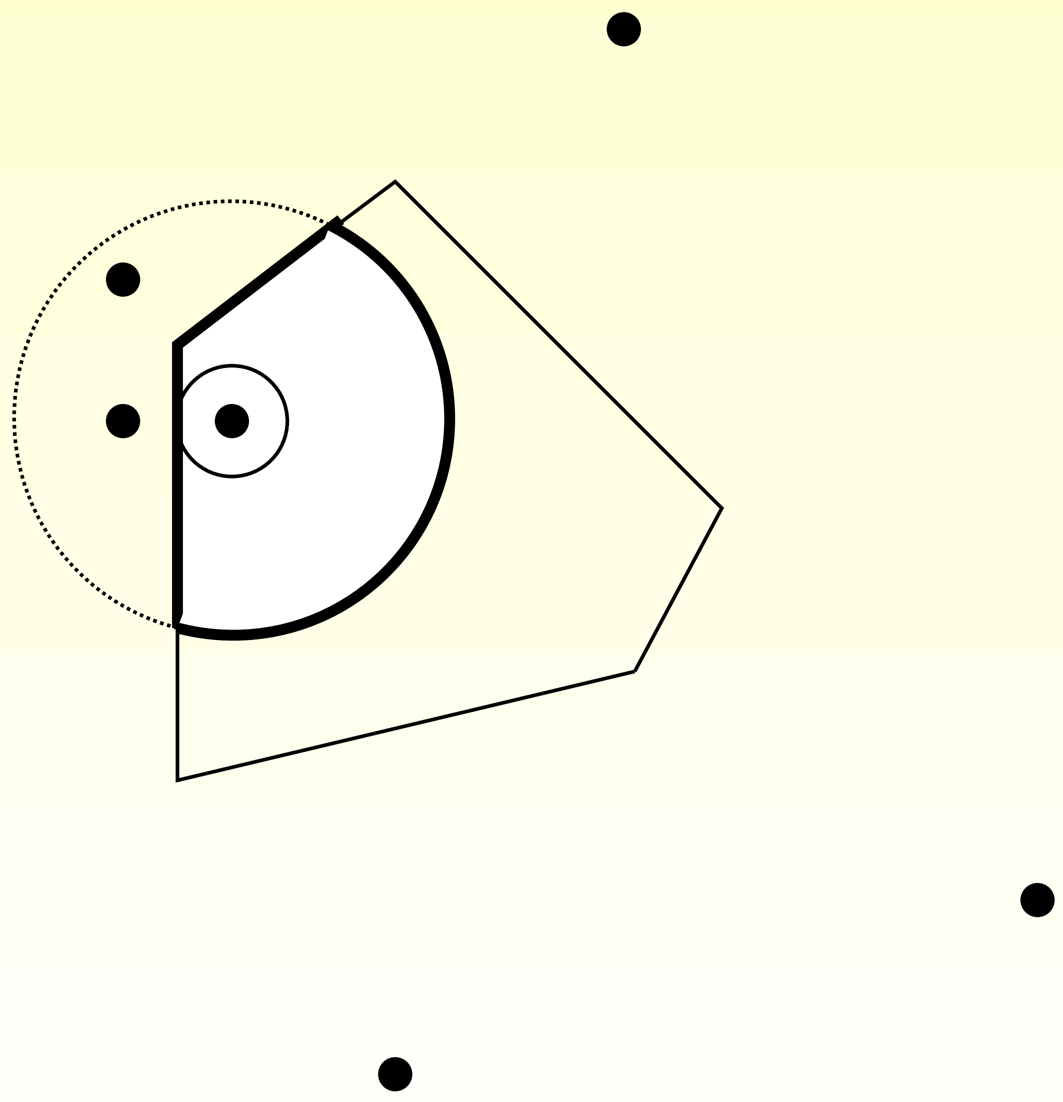
Pick a point that is not well-spaced and fill (insert Steiner points)



Stable Refinement Algorithm

Basic operation of the construction algorithm is fill

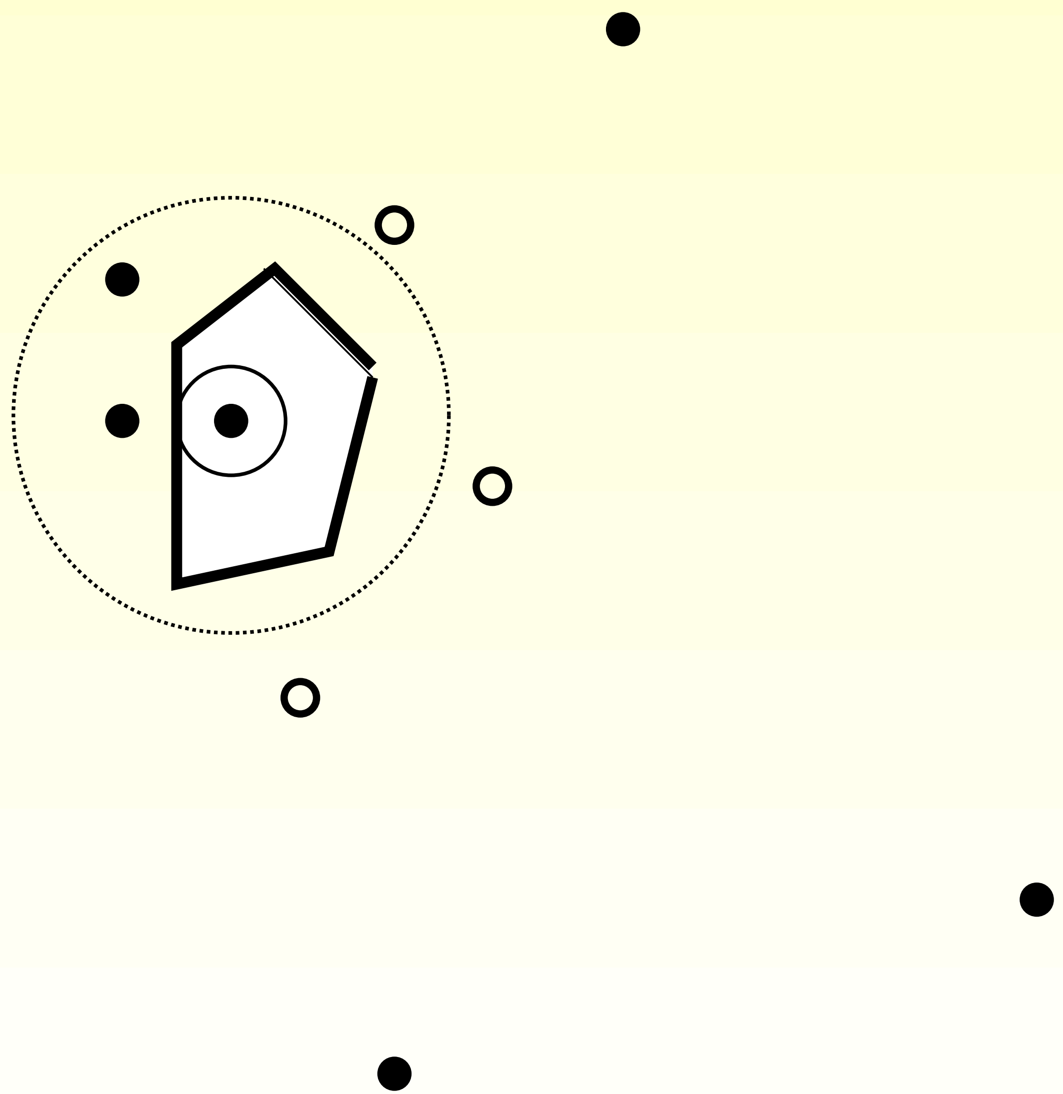
Pick a point that is not well-spaced and fill (insert Steiner points)



Stable Refinement Algorithm

Basic operation of the construction algorithm is fill

Pick a point that is not well-spaced and fill (insert Steiner points)

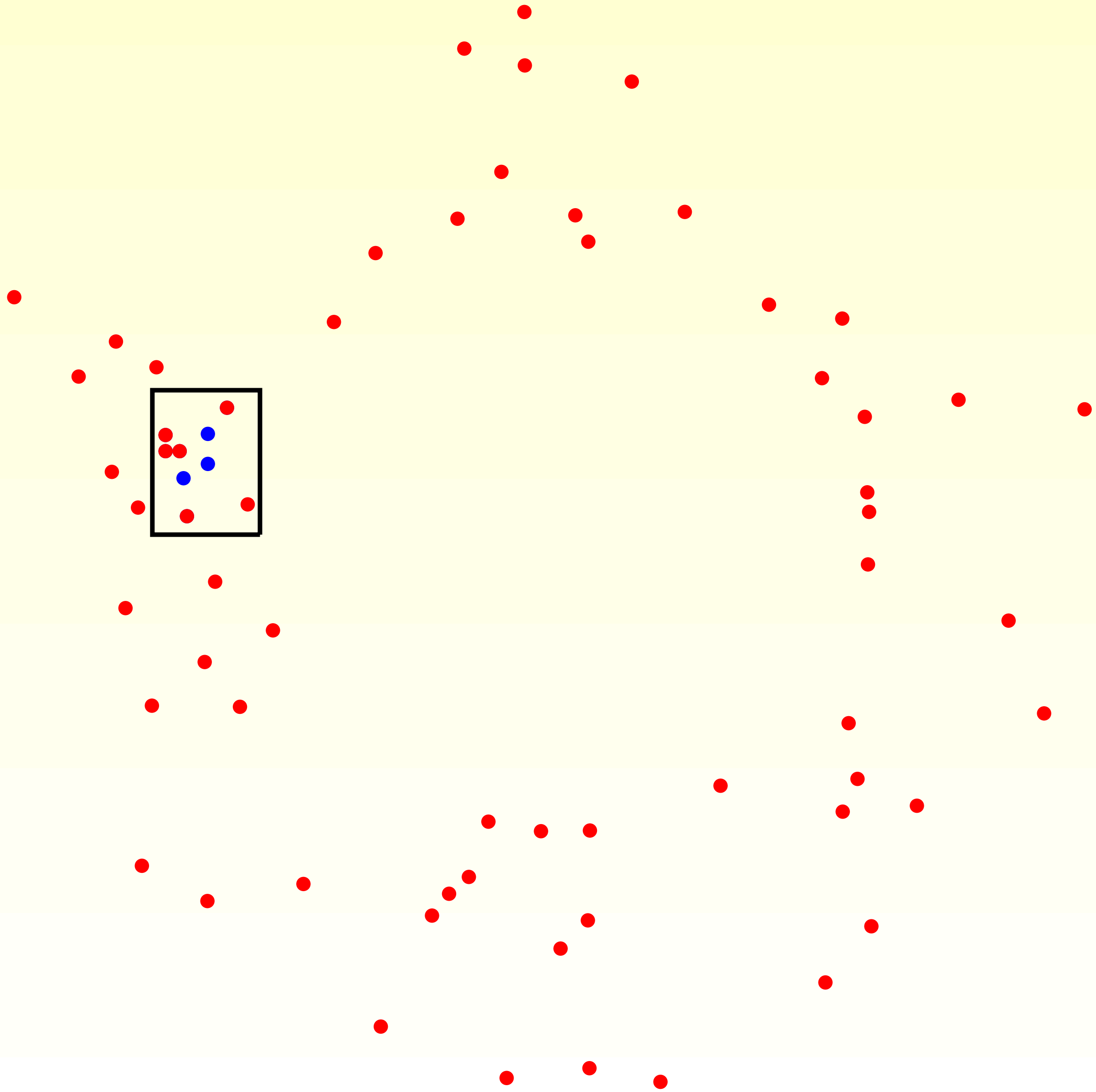


Stable Refinement Algorithm

Basic operation of the construction algorithm is fill

Pick a point that is not well-spaced and fill (insert Steiner points)

Idea: Fill points in a single pass

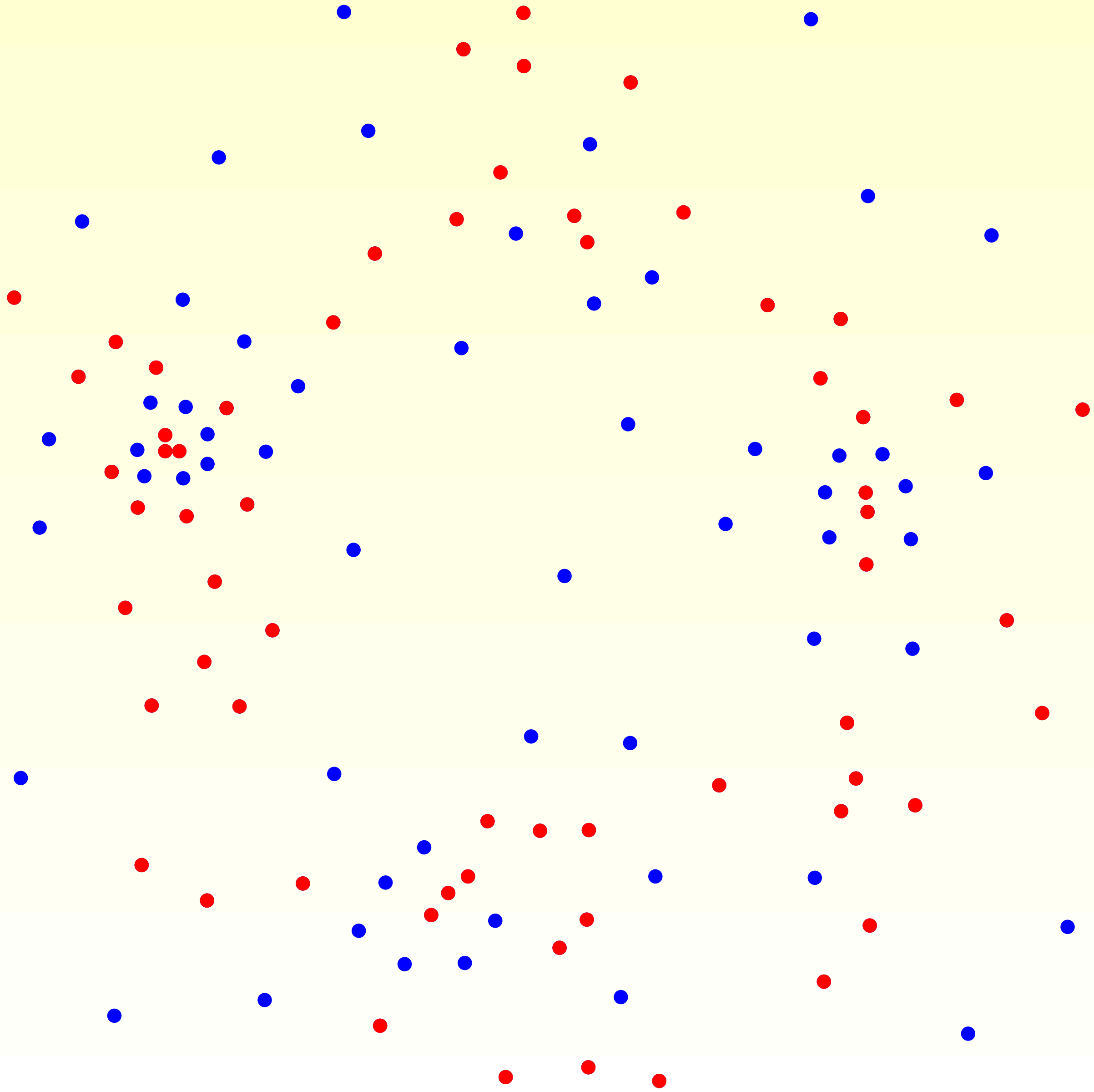


Stable Refinement Algorithm

Basic operation of the construction algorithm is fill

Pick a point that is not well-spaced and fill (insert Steiner points)

Idea: Fill points in a single pass



Fill Operation and Picking Region

Goal: Ensure that filling points in a single pass is sufficient

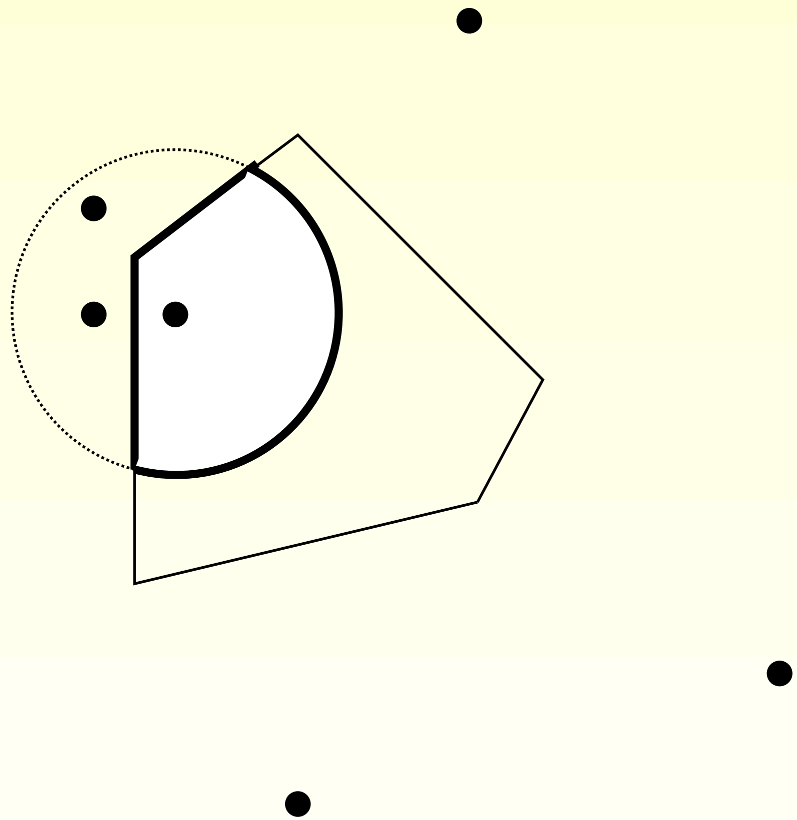
Approach: Expansion

Insert Steiner points that have ρ times larger empty balls

Goal: Efficiency, Independence

Approach: Clipped Voronoi

Location of Steiner points depend only on close Voronoi neighbors



Fill Operation and Picking Region

Goal: Ensure that filling points in a single pass is sufficient

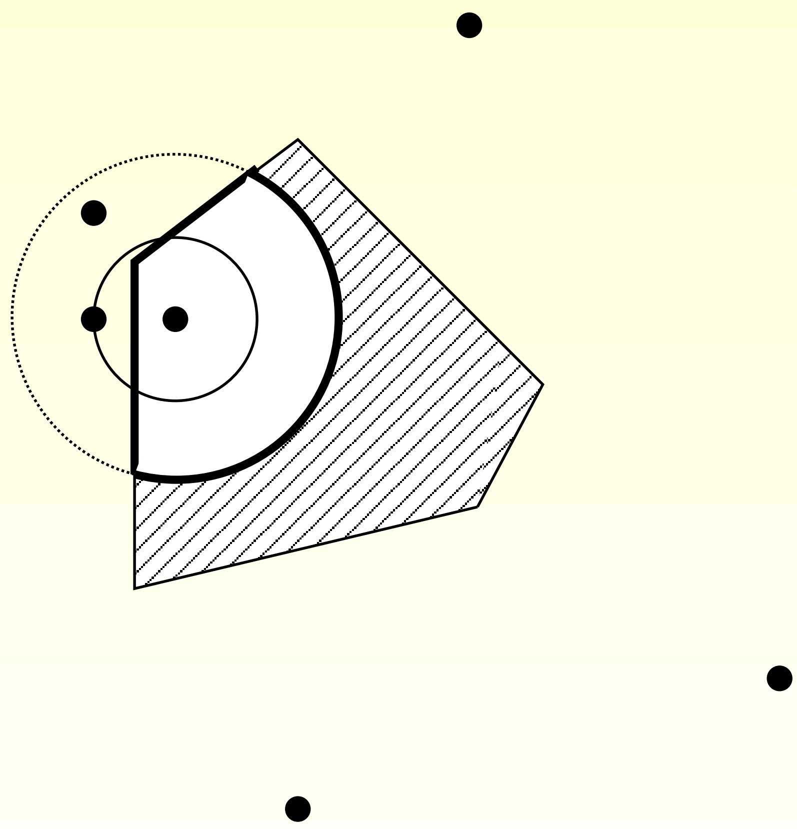
Approach: Expansion

Insert Steiner points that have ρ times larger empty balls

Goal: Efficiency, Independence

Approach: Clipped Voronoi

Location of Steiner points depend only on close Voronoi neighbors



Fill Operation and Picking Region

Goal: Ensure that filling points in a single pass is sufficient

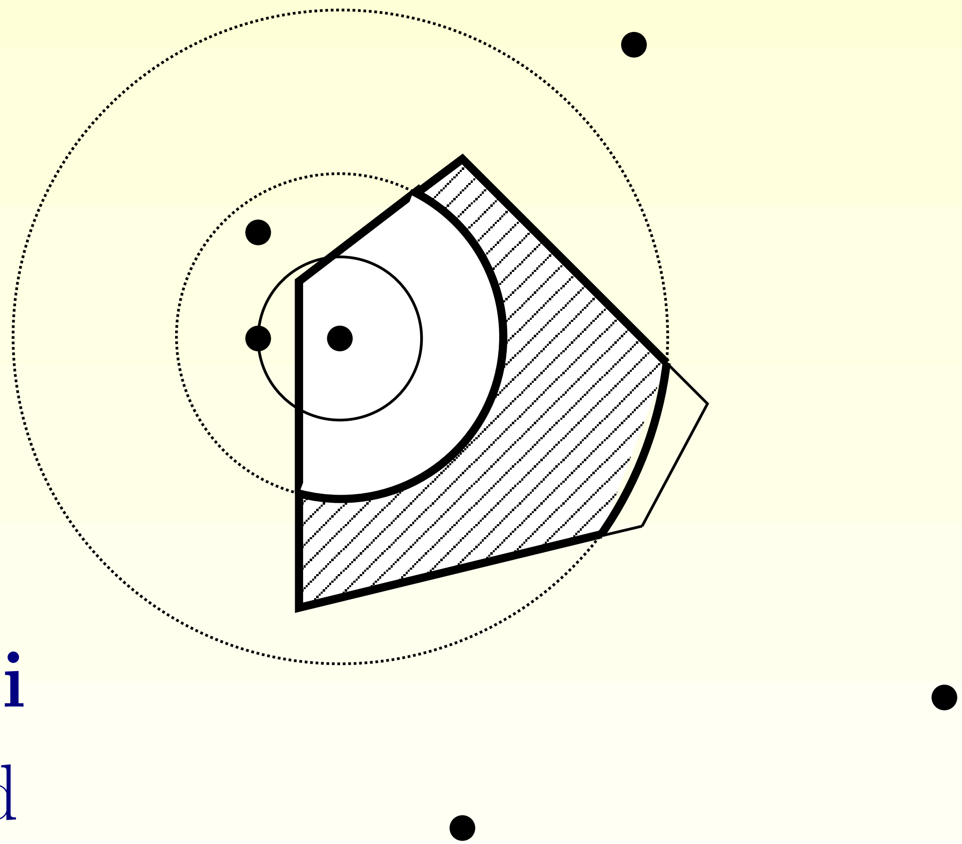
Approach: Expansion

Insert Steiner points that have ρ times larger empty balls

Goal: Efficiency, Independence

Approach: Clipped Voronoi

Location of Steiner points depend only on close Voronoi neighbors

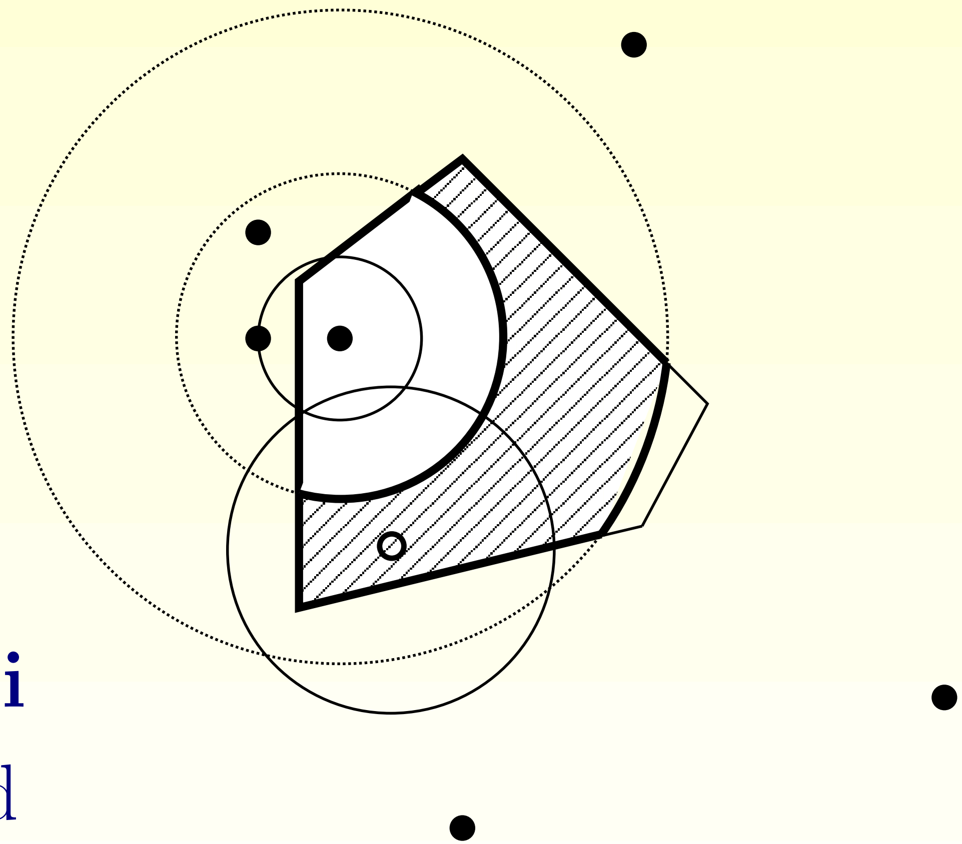


Fill Operation and Picking Region

Goal: Ensure that filling points in a single pass is sufficient

Approach: Expansion

Insert Steiner points that have ρ times larger empty balls



Goal: Efficiency, Independence

Approach: Clipped Voronoi

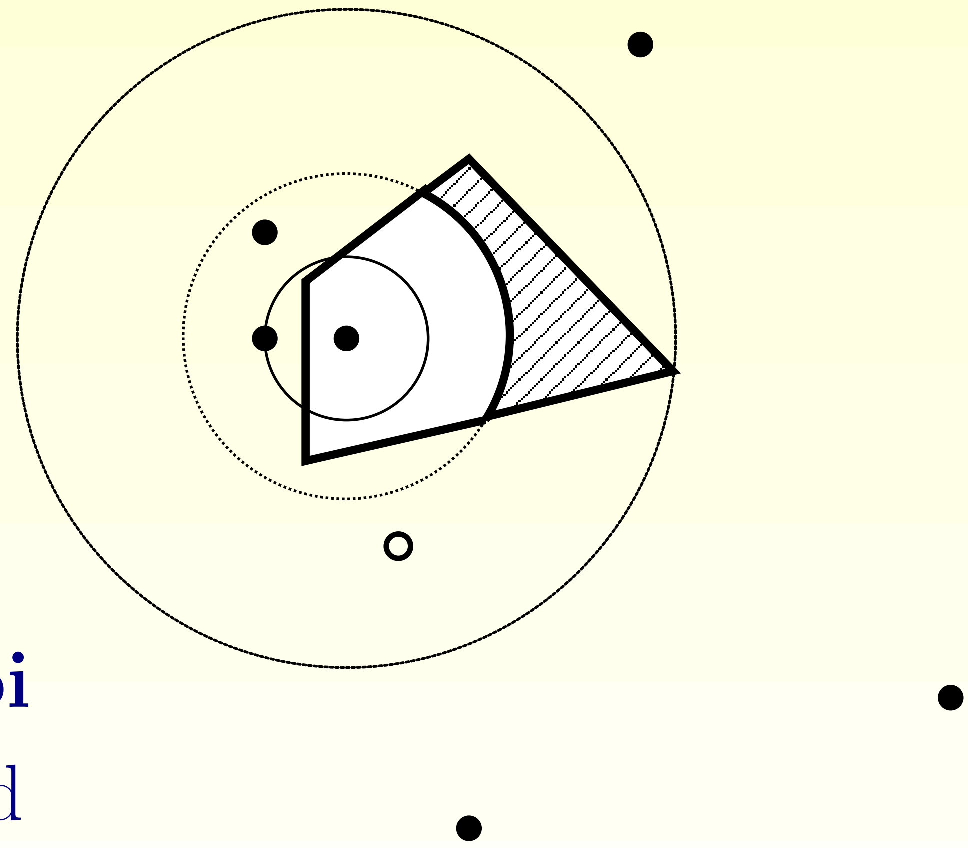
Location of Steiner points depend only on close Voronoi neighbors

Fill Operation and Picking Region

Goal: Ensure that filling points in a single pass is sufficient

Approach: Expansion

Insert Steiner points that have ρ times larger empty balls



Goal: Efficiency, Independence

Approach: Clipped Voronoi

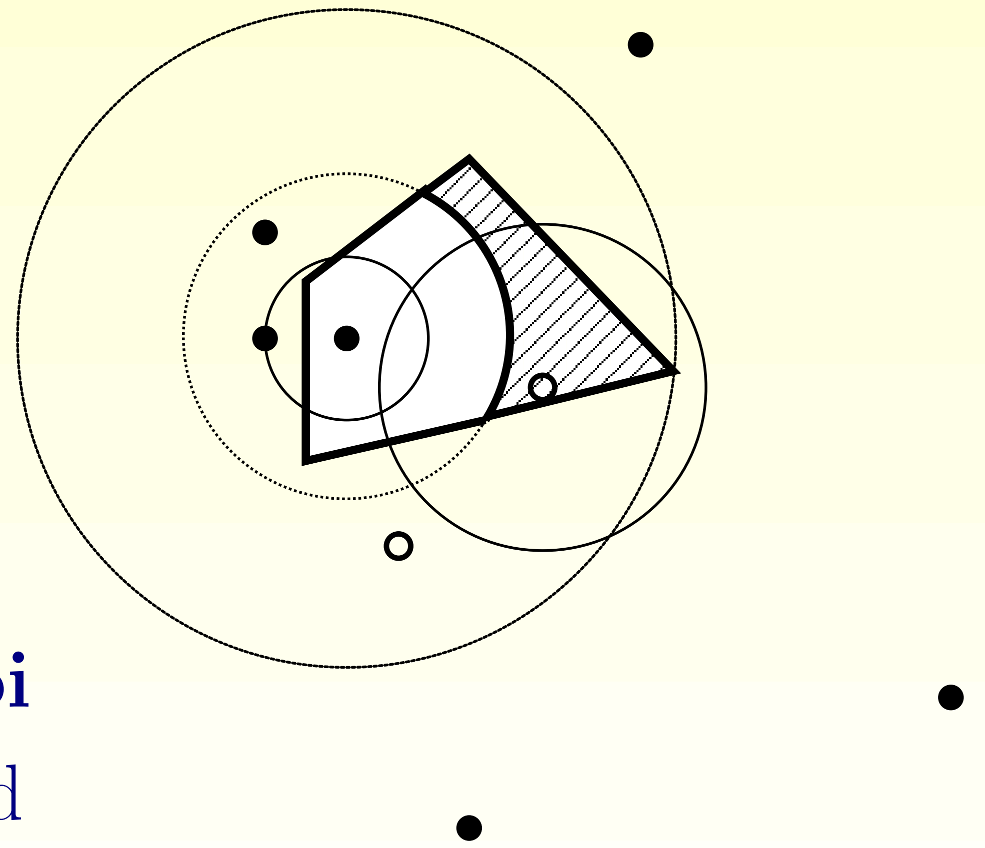
Location of Steiner points depend only on close Voronoi neighbors

Fill Operation and Picking Region

Goal: Ensure that filling points in a single pass is sufficient

Approach: Expansion

Insert Steiner points that have ρ times larger empty balls



Goal: Efficiency, Independence

Approach: Clipped Voronoi

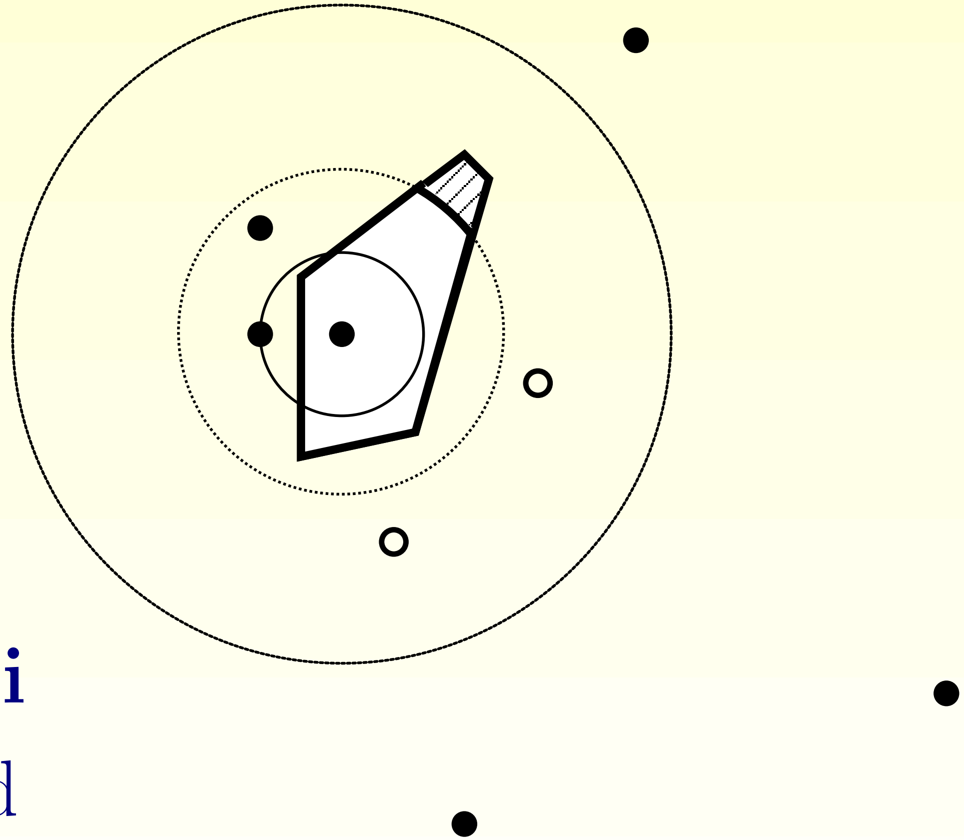
Location of Steiner points depend only on close Voronoi neighbors

Fill Operation and Picking Region

Goal: Ensure that filling points in a single pass is sufficient

Approach: Expansion

Insert Steiner points that have ρ times larger empty balls



Goal: Efficiency, Independence

Approach: Clipped Voronoi

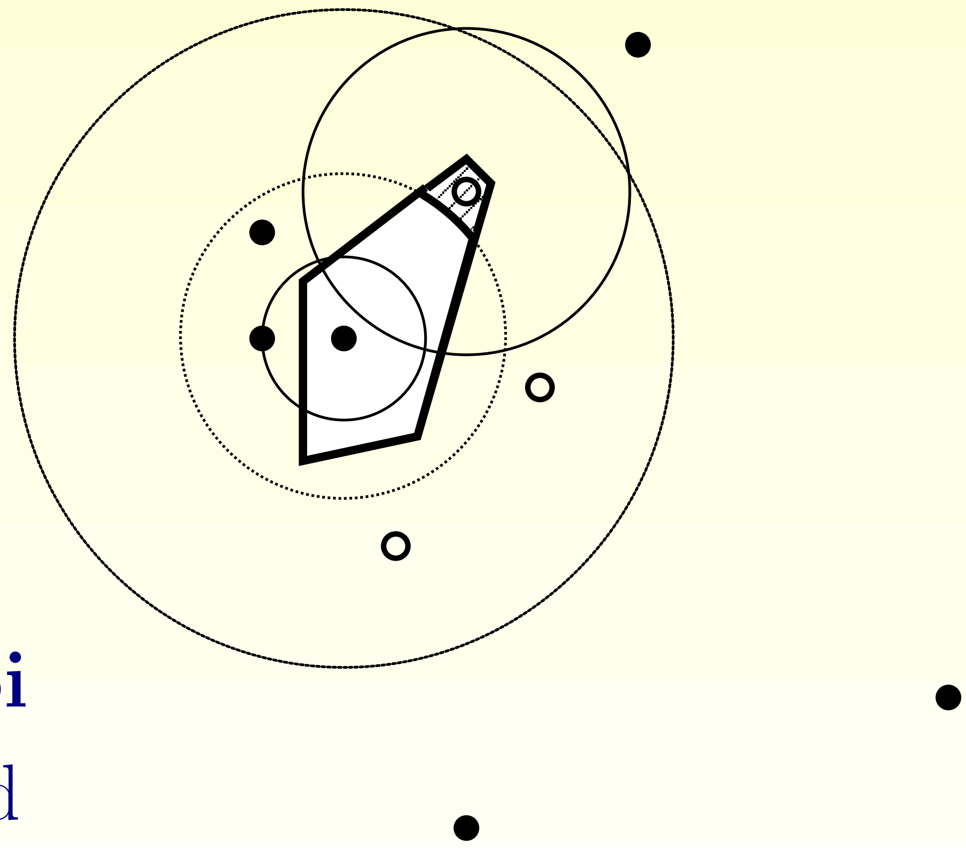
Location of Steiner points depend only on close Voronoi neighbors

Fill Operation and Picking Region

Goal: Ensure that filling points in a single pass is sufficient

Approach: Expansion

Insert Steiner points that have ρ times larger empty balls



Goal: Efficiency, Independence

Approach: Clipped Voronoi

Location of Steiner points depend only on close Voronoi neighbors

Fill Operation and Picking Region

Goal: Ensure that filling points in a single pass is sufficient

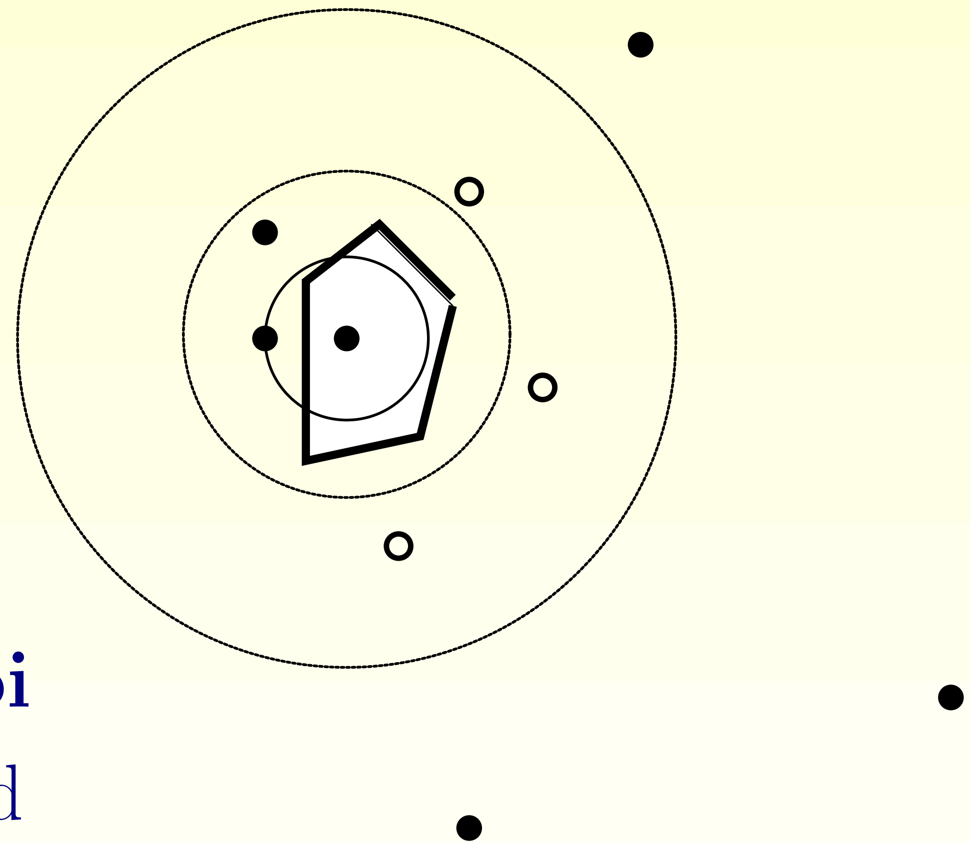
Approach: Expansion

Insert Steiner points that have ρ times larger empty balls

Goal: Efficiency, Independence

Approach: Clipped Voronoi

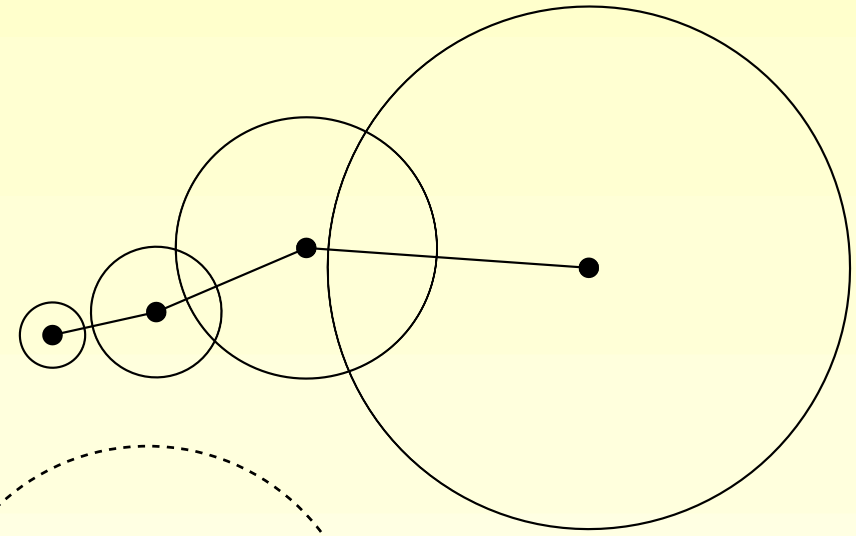
Location of Steiner points depend only on close Voronoi neighbors



Key Ideas for Stability

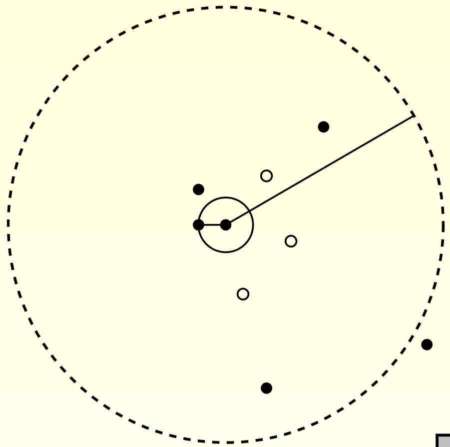
Expansion

Steiner points have ρ times larger empty balls



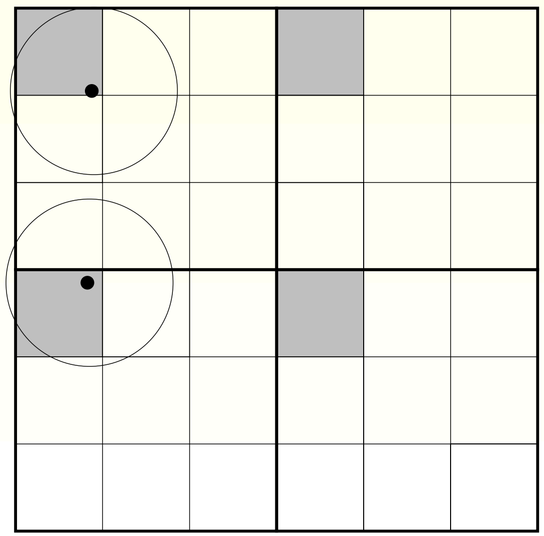
Locality

Steiner point selection depends on a bounded neighborhood



Independence

At each rank, filling distant points are guaranteed not to affect each other



Key Ideas for Stability

Expansion

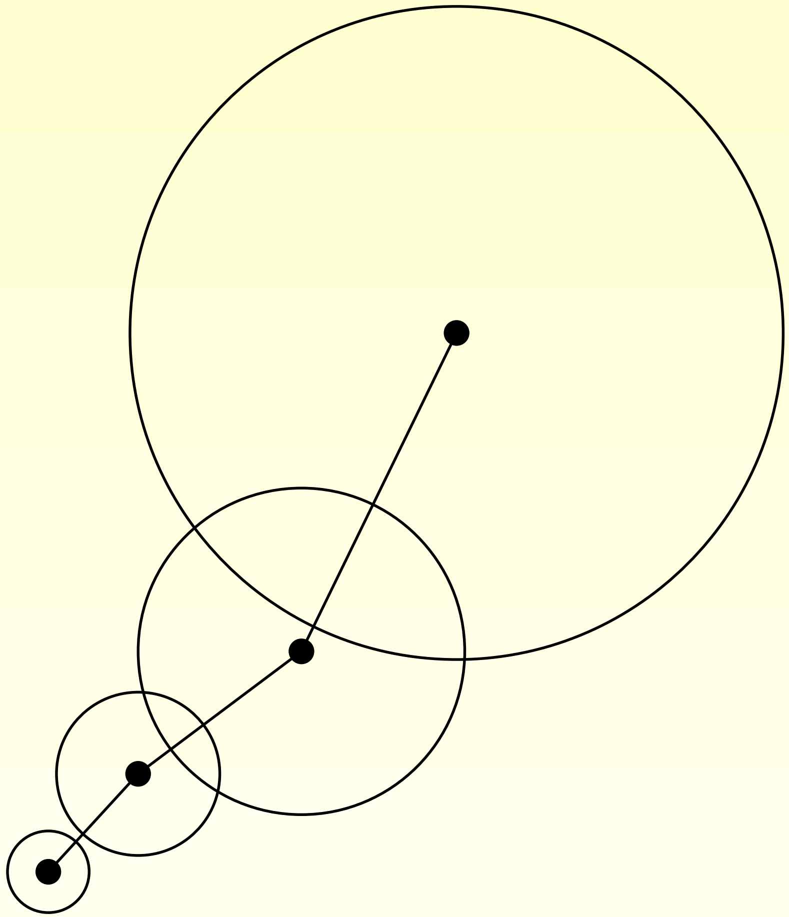
Steiner points have ρ times larger empty balls

Locality

Steiner point selection depends on a bounded neighborhood

Independence

At each rank, filling distant points are guaranteed not to affect each other



Rank = logarithm base ρ of nearest neighbor distance

In total $O(\log \Delta)$ ranks

Key Ideas for Stability

Expansion

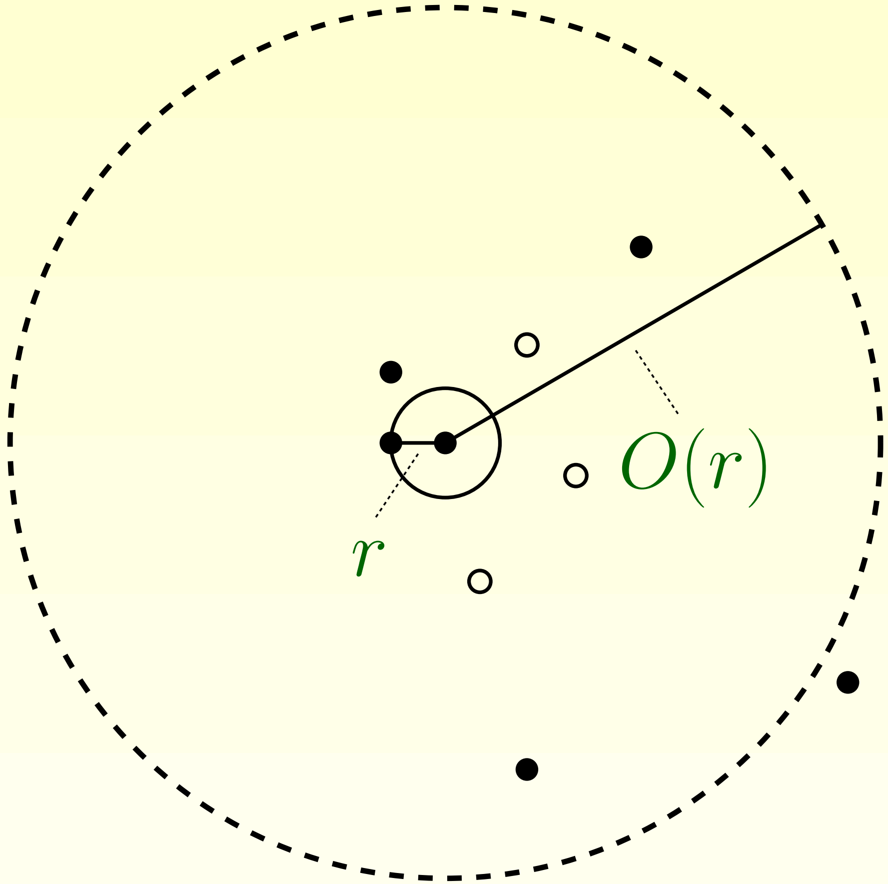
Steiner points have ρ times larger empty balls

Locality

Steiner point selection depends on a bounded neighborhood

Independence

At each rank, filling distant points are guaranteed not to affect each other



Points outside the big ball do not influence how we pick Steiner points

Key Ideas for Stability

Expansion

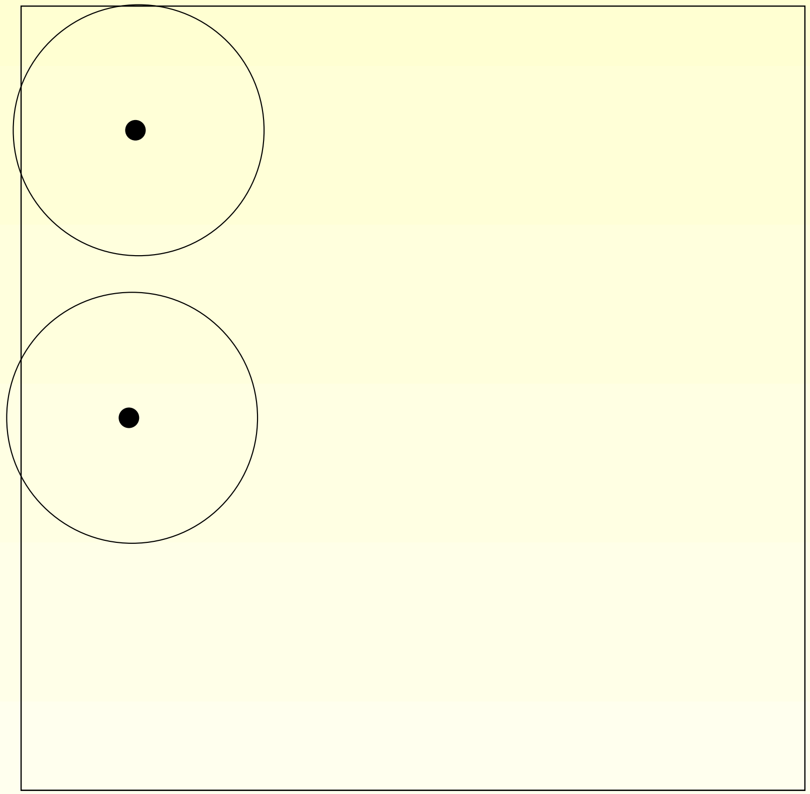
Steiner points have ρ times larger empty balls

Locality

Steiner point selection depends on a bounded neighborhood

Independence

At each rank, filling distant points are guaranteed not to affect each other



Process points in rank order

Two points at a given rank may not depend on each other

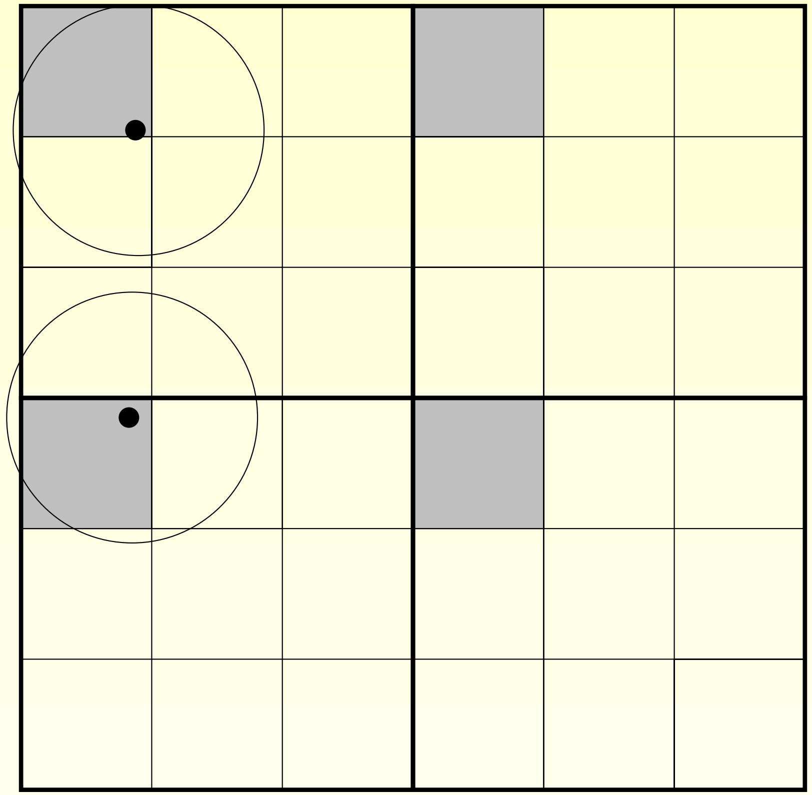
Key Ideas for Stability

Expansion

Steiner points have ρ times larger empty balls

Locality

Steiner point selection depends on a bounded neighborhood



Independence

At each rank, filling distant points are guaranteed not to affect each other

For ensuring independence partition space using $O(1)$ colors and process points in color order

Proof of Stability — Dependency Path

Schedule using Ranks and Colors

Algorithm processes points in stages by ordering them first by rank then by color

Dependency Paths

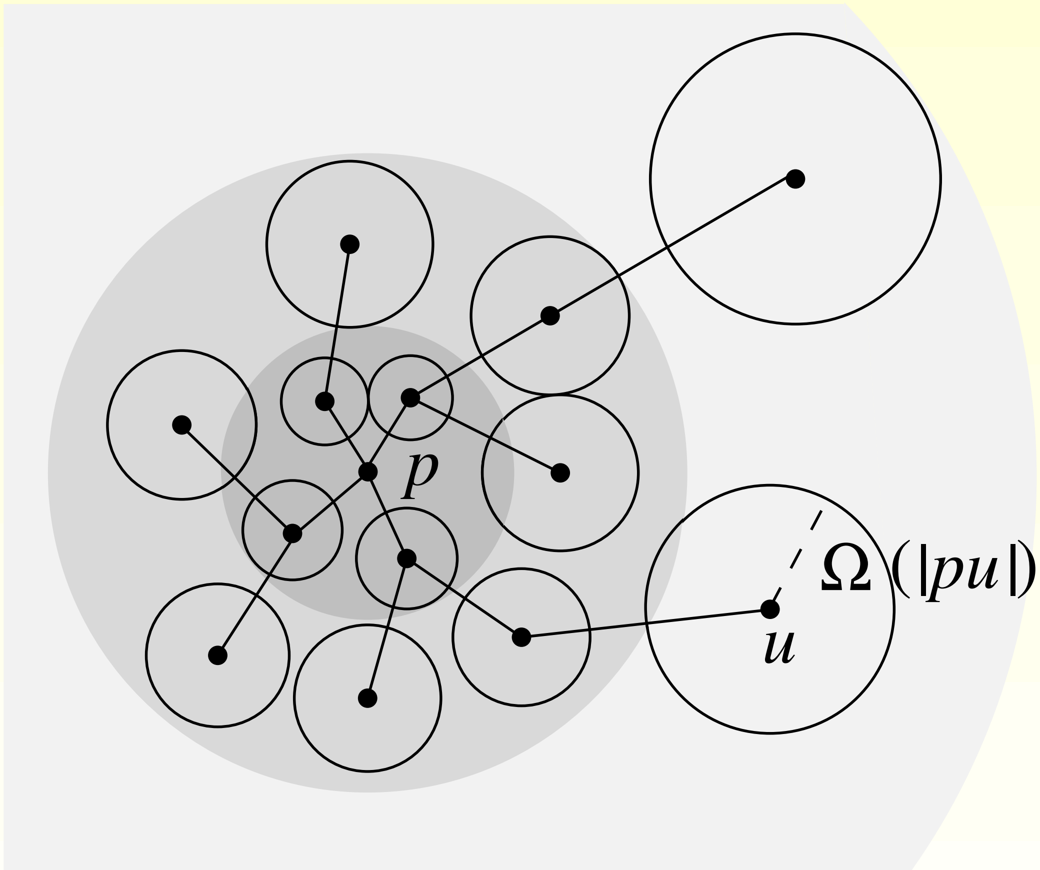
Expansion and independence guarantee that filling a point does not affect points processed in earlier stages and the points being processed at the current stage

There are $O(\log \Delta)$ stages in total

Proof of Stability — Spacing and Packing

Assume inserting (or deleting) a point p

Path: Given a point u at rank r if there is a dependency path from p to u, then $|pu| < O(\rho^r)$



Spacing: There is an empty ball around u of radius $\Omega(\rho^r)$ which consequently is $\Omega(|pu|)$

Packing: There can be only $O(1)$ such points at each rank $O(\log \Delta)$ in total

Thank You!

Questions?