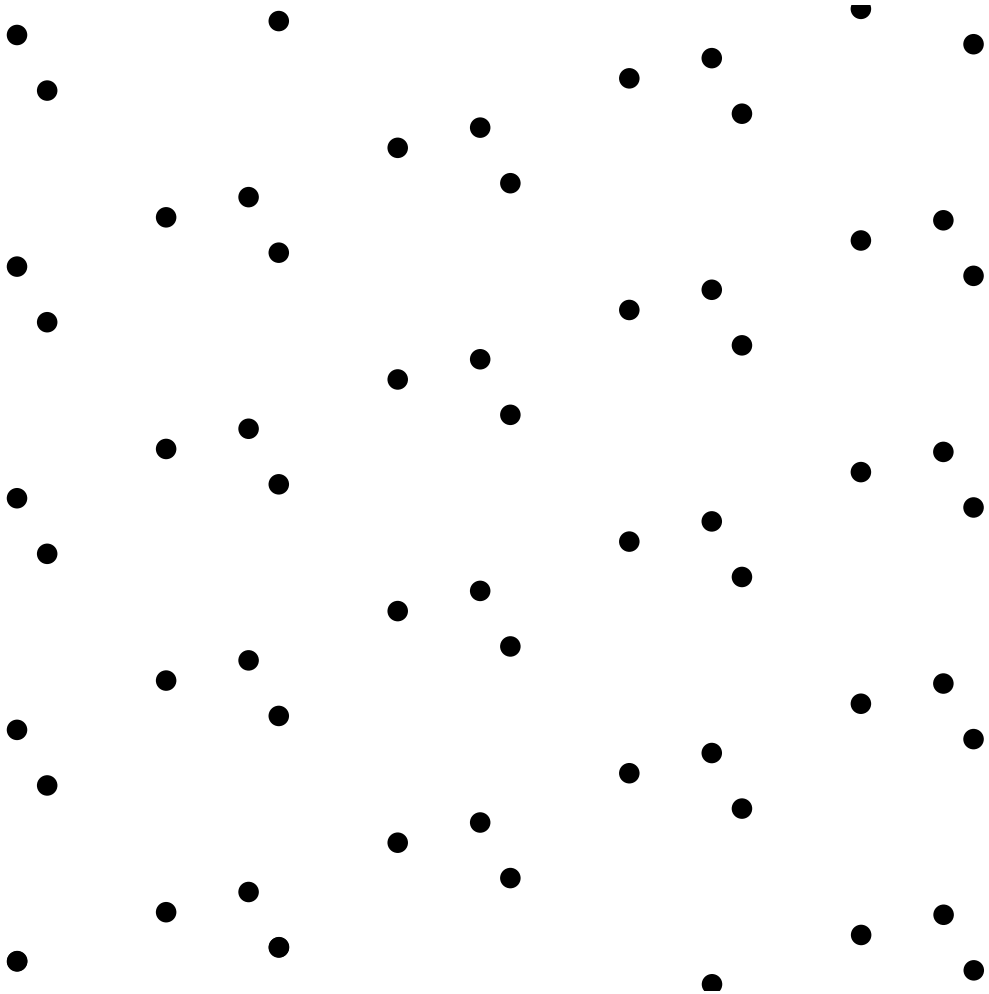
Generalizing CGAL Periodic Delaunay Triangulations

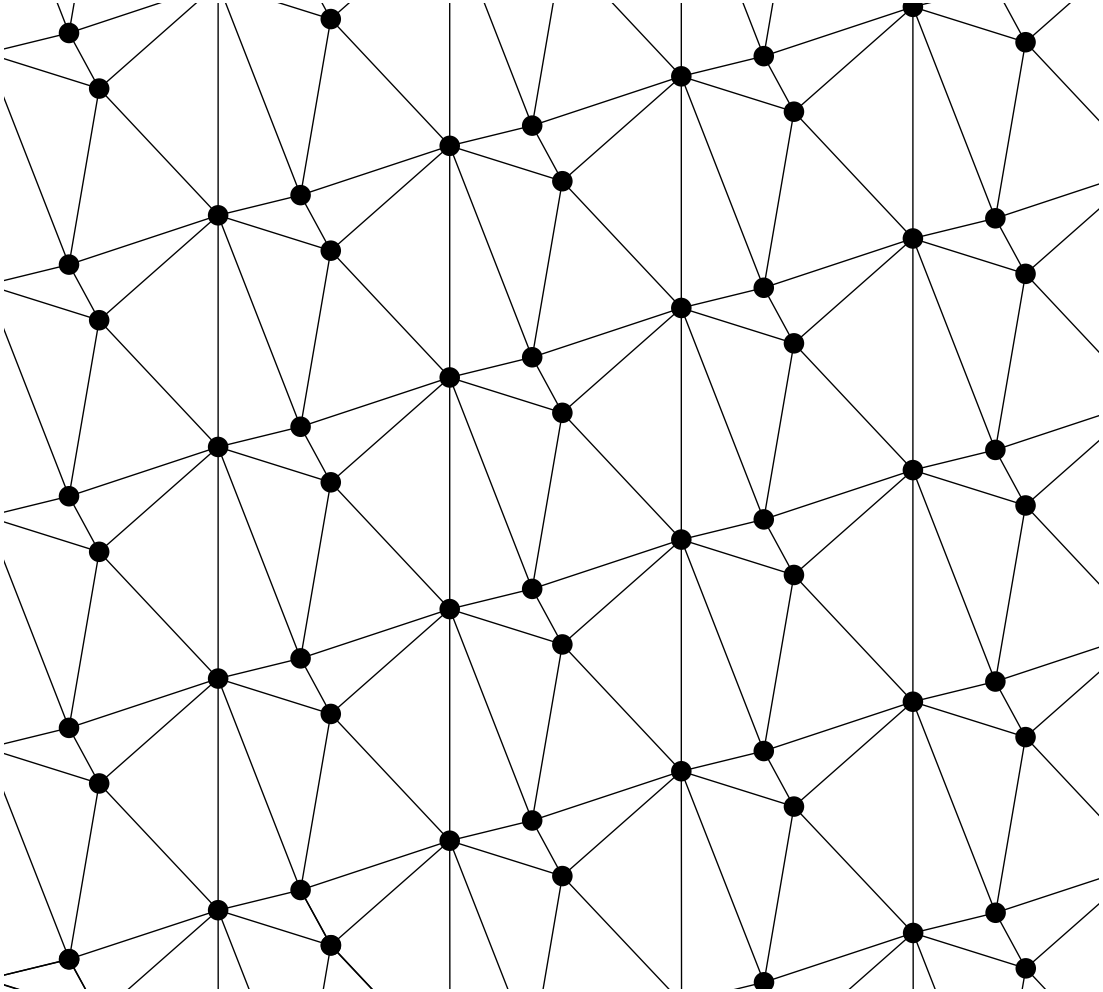
Georg Osang, Mael Rouxel-Labbé and Monique Teillaud

September 8th, 2020

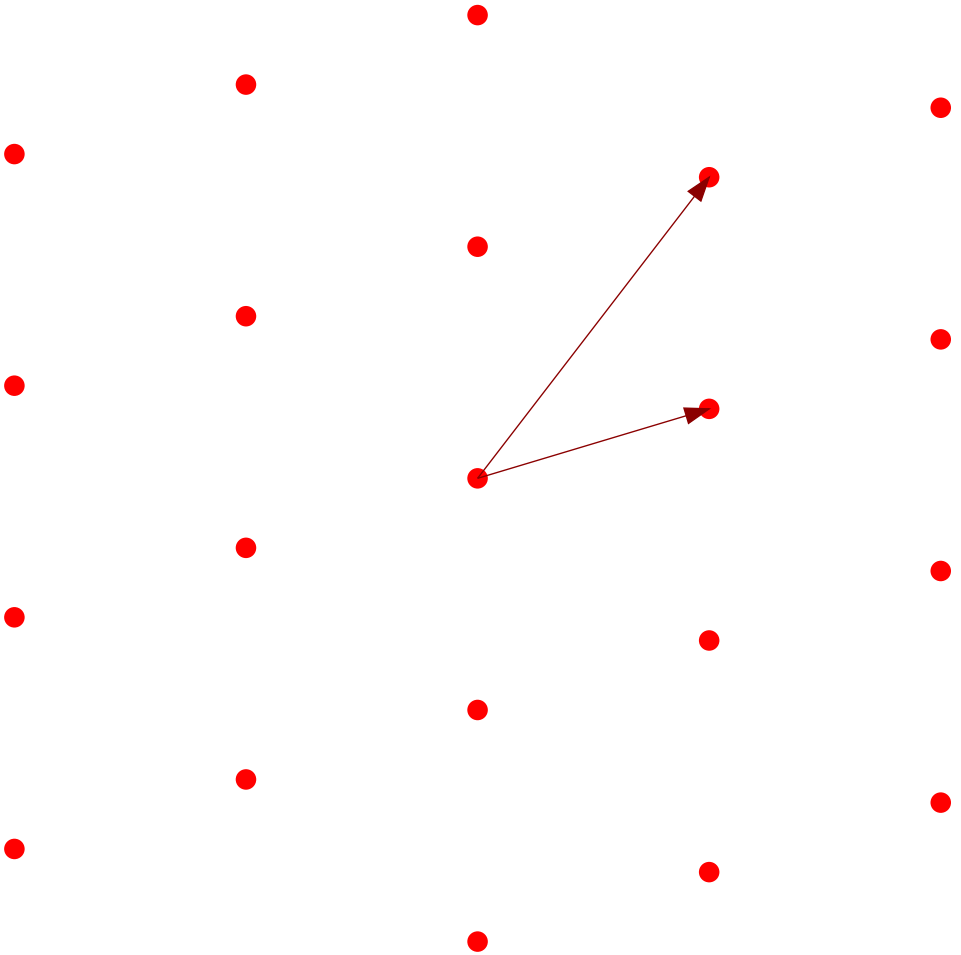
Periodic Delaunay triangulations



Periodic Delaunay triangulations



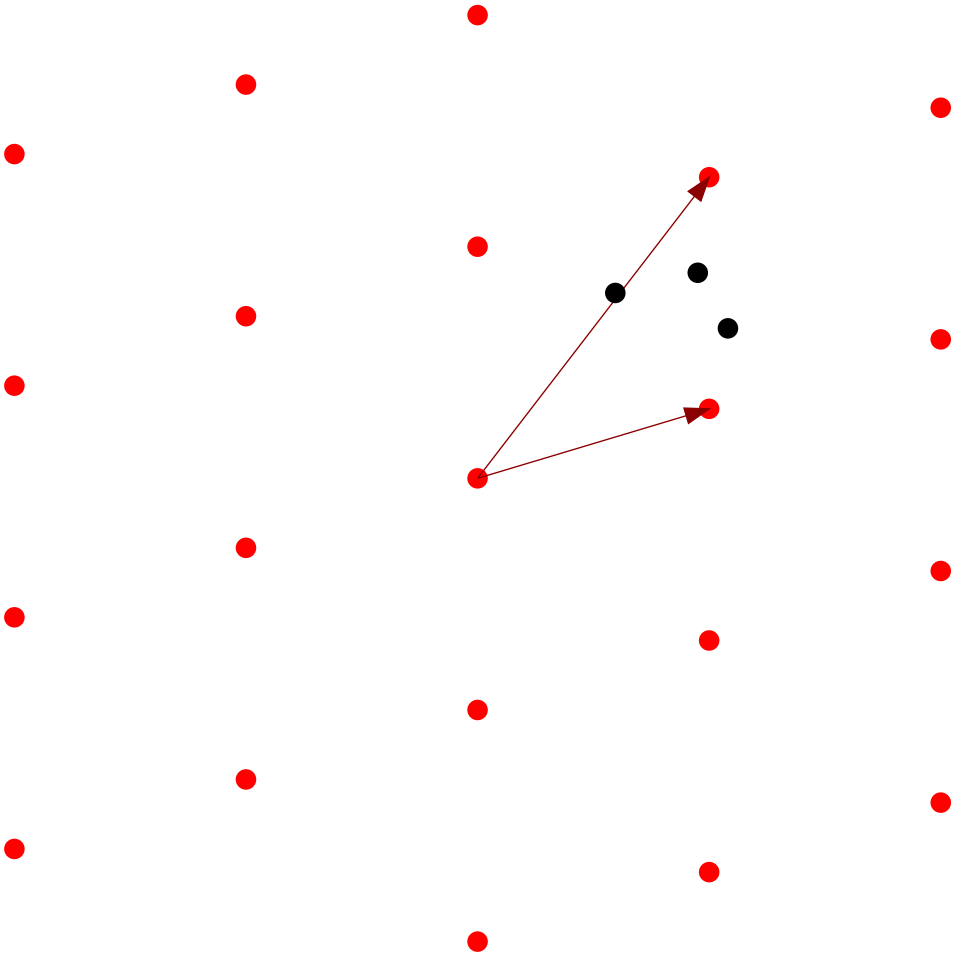
Periodic Delaunay triangulations



Input:

- Lattice Λ with basis B

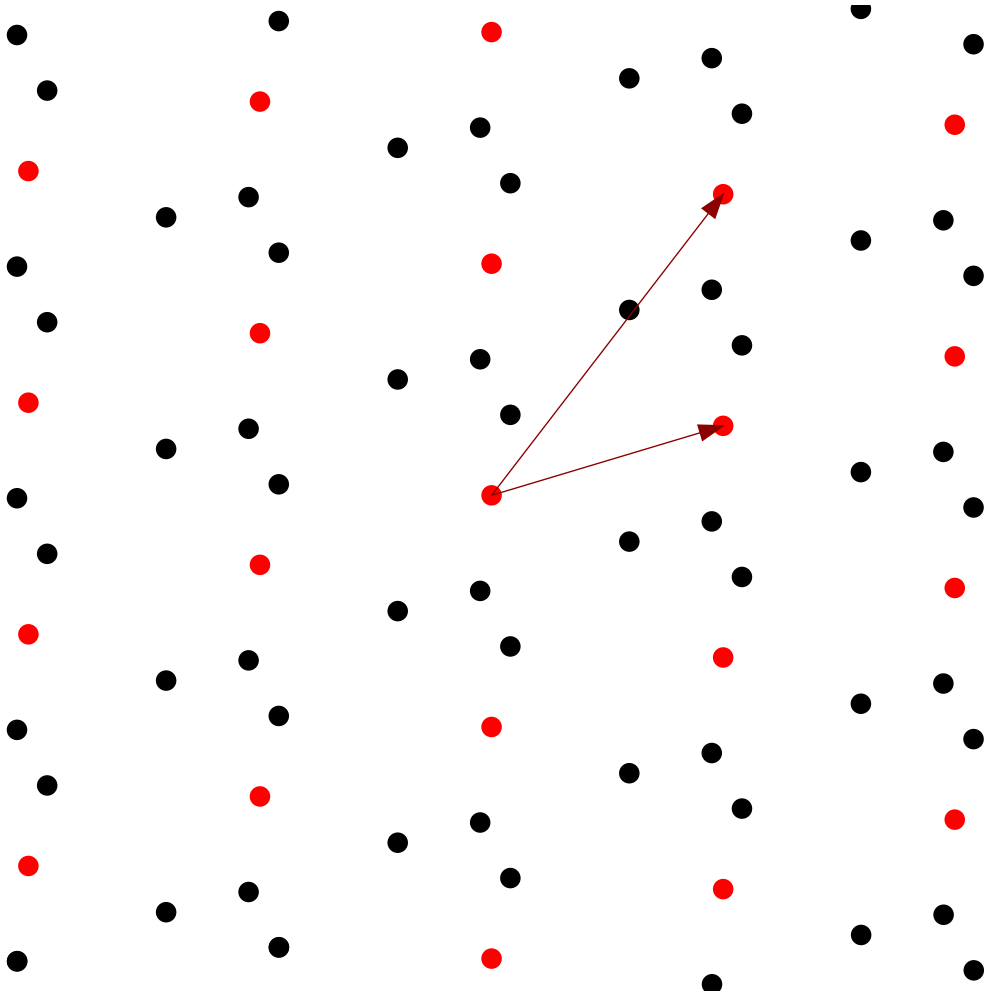
Periodic Delaunay triangulations



Input:

- Lattice Λ with basis B
- Representative points X

Periodic Delaunay triangulations

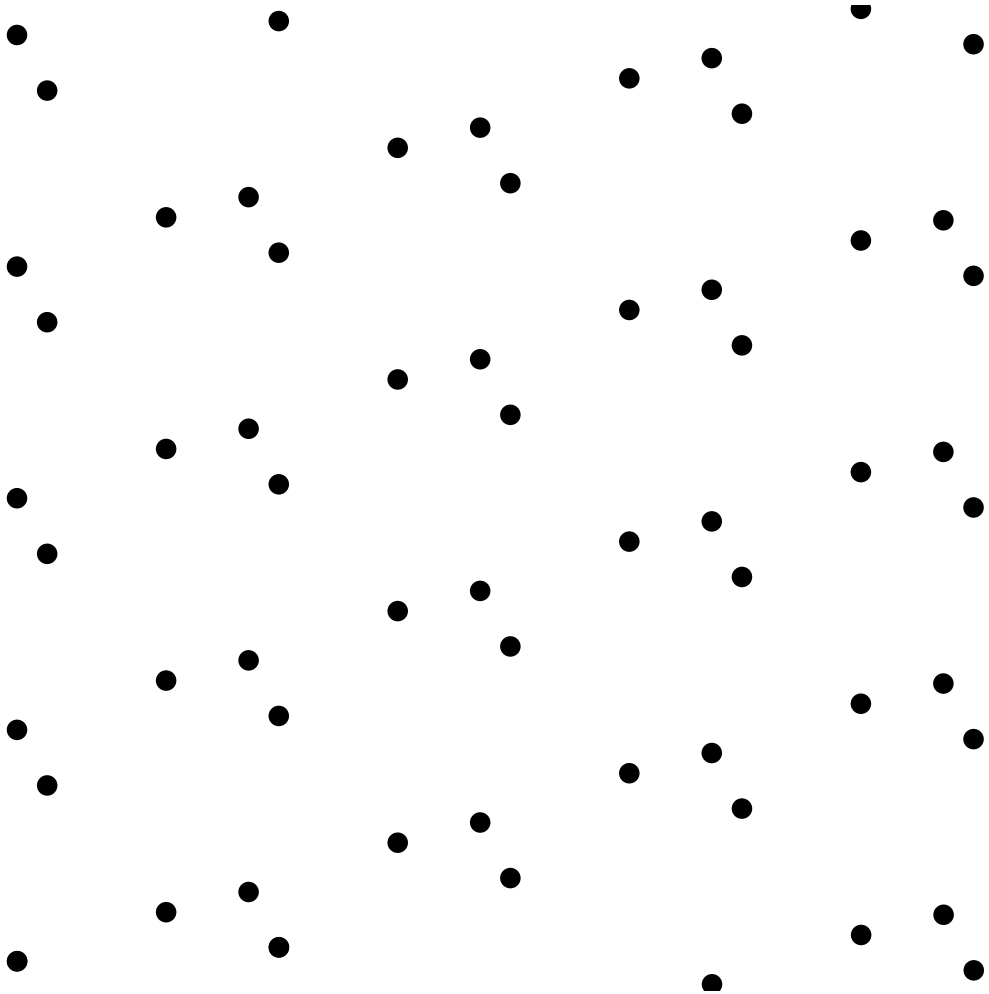


Input:

- Lattice Λ with basis B
- Representative points X

$\rightarrow$ Forming periodic point set ΛX (in 2D or 3D)

Periodic Delaunay triangulations

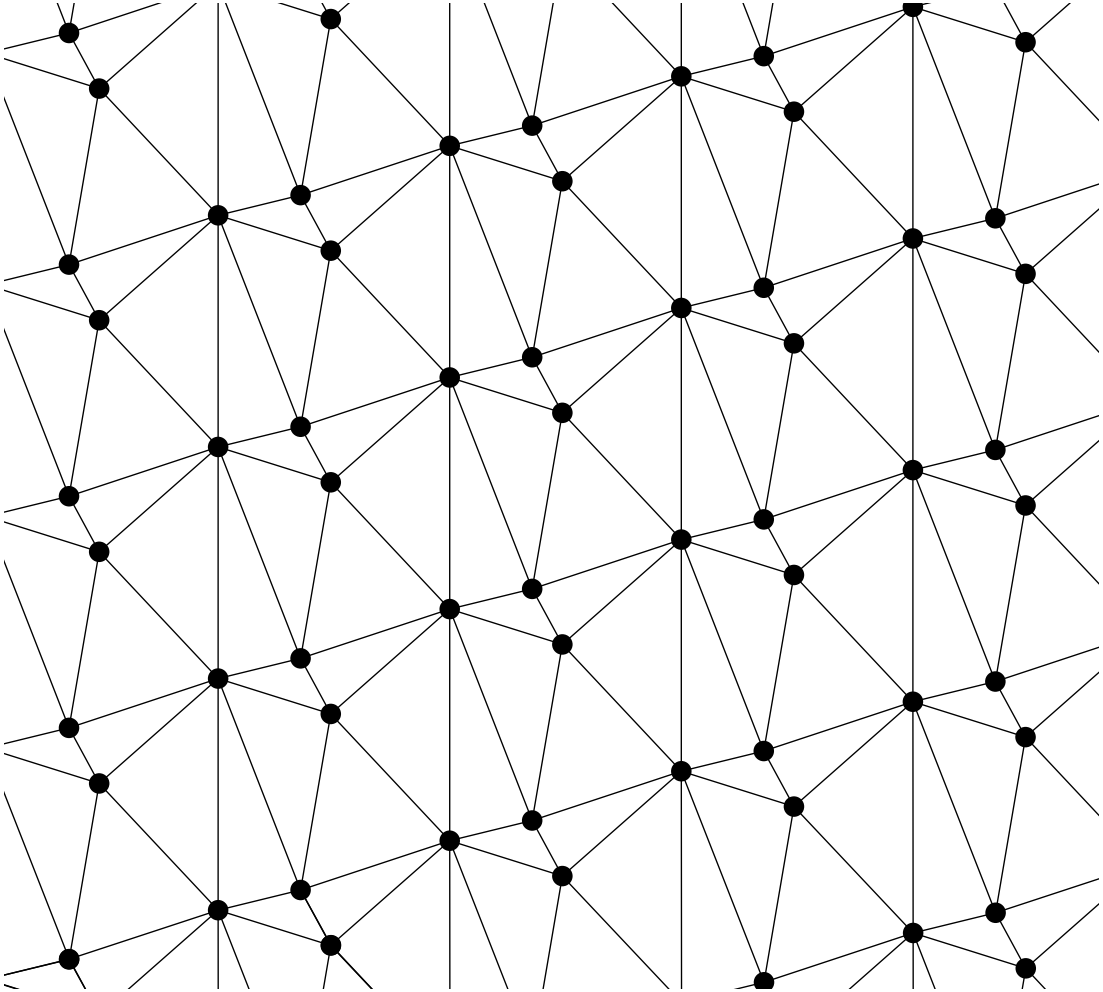


Input:

- Lattice Λ with basis B
- Representative points X

$\rightarrow$ Forming periodic point set ΛX (in 2D or 3D)

Periodic Delaunay triangulations



Input:

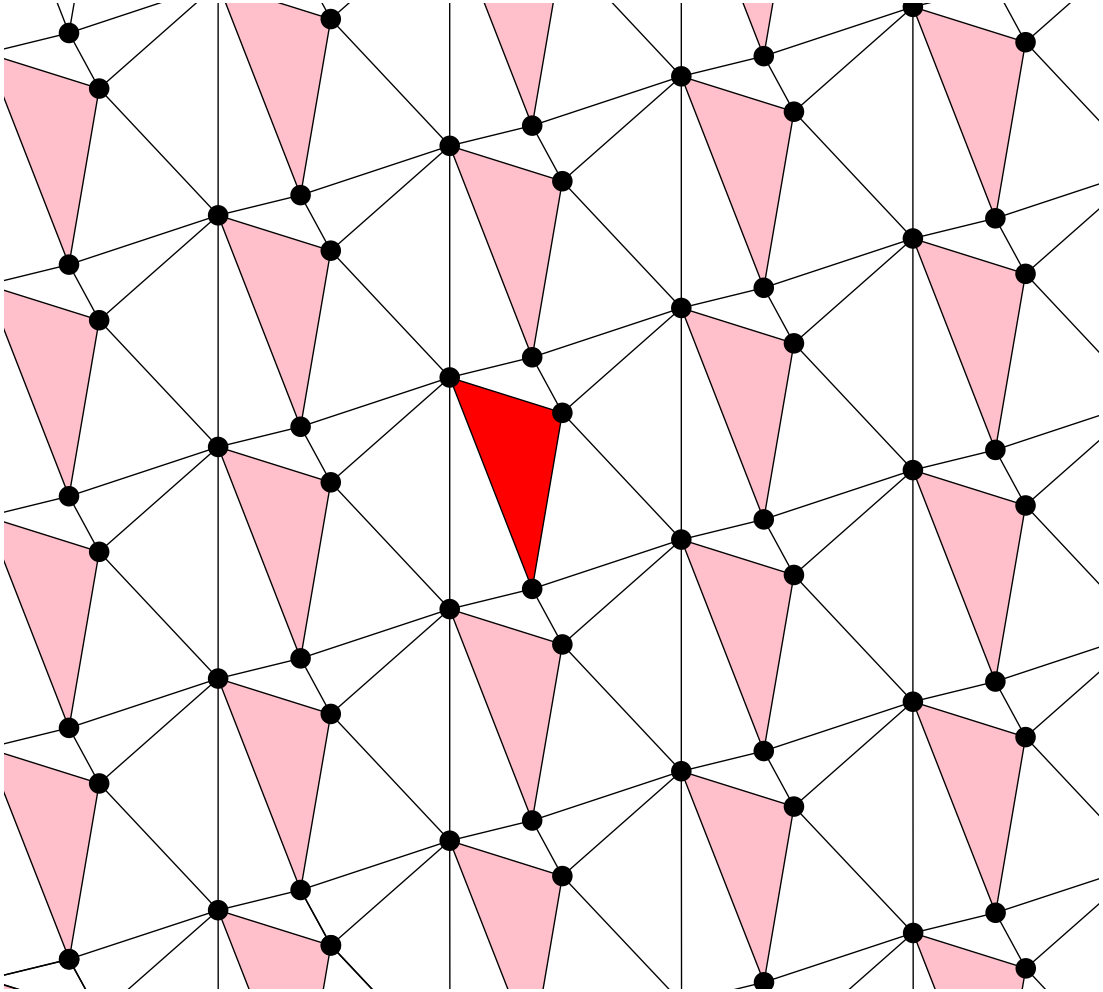
- Lattice Λ with basis B
- Representative points X

$\rightarrow$ Forming periodic point set ΛX (in 2D or 3D)

Output:

- Periodic triangulation

Periodic Delaunay triangulations



Input:

- Lattice Λ with basis B
- Representative points X

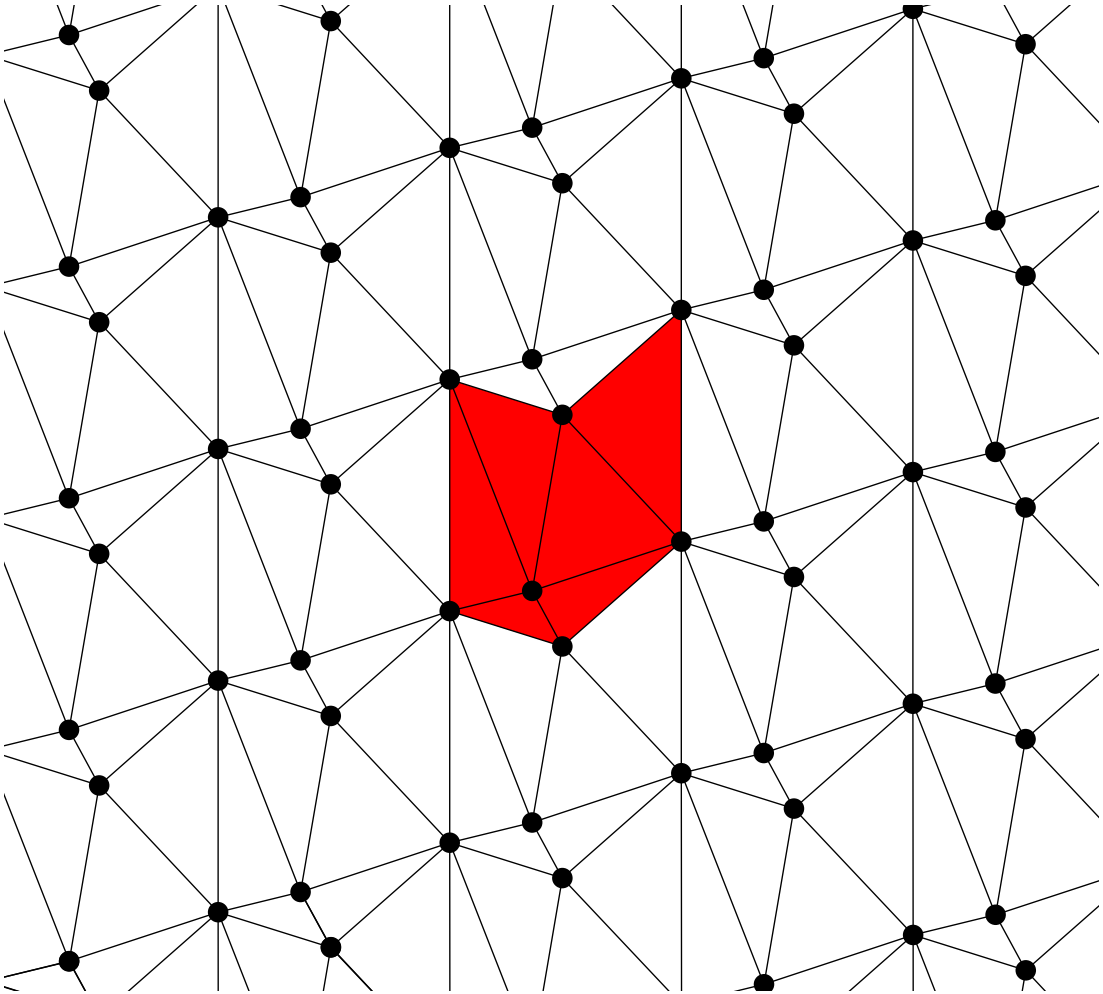
$\rightarrow$ Forming periodic point set ΛX (in 2D or 3D)

Output:

- Periodic triangulation

$\rightarrow$ Representative cells (one per equivalence class)

Periodic Delaunay triangulations



Input:

- Lattice Λ with basis B
- Representative points X

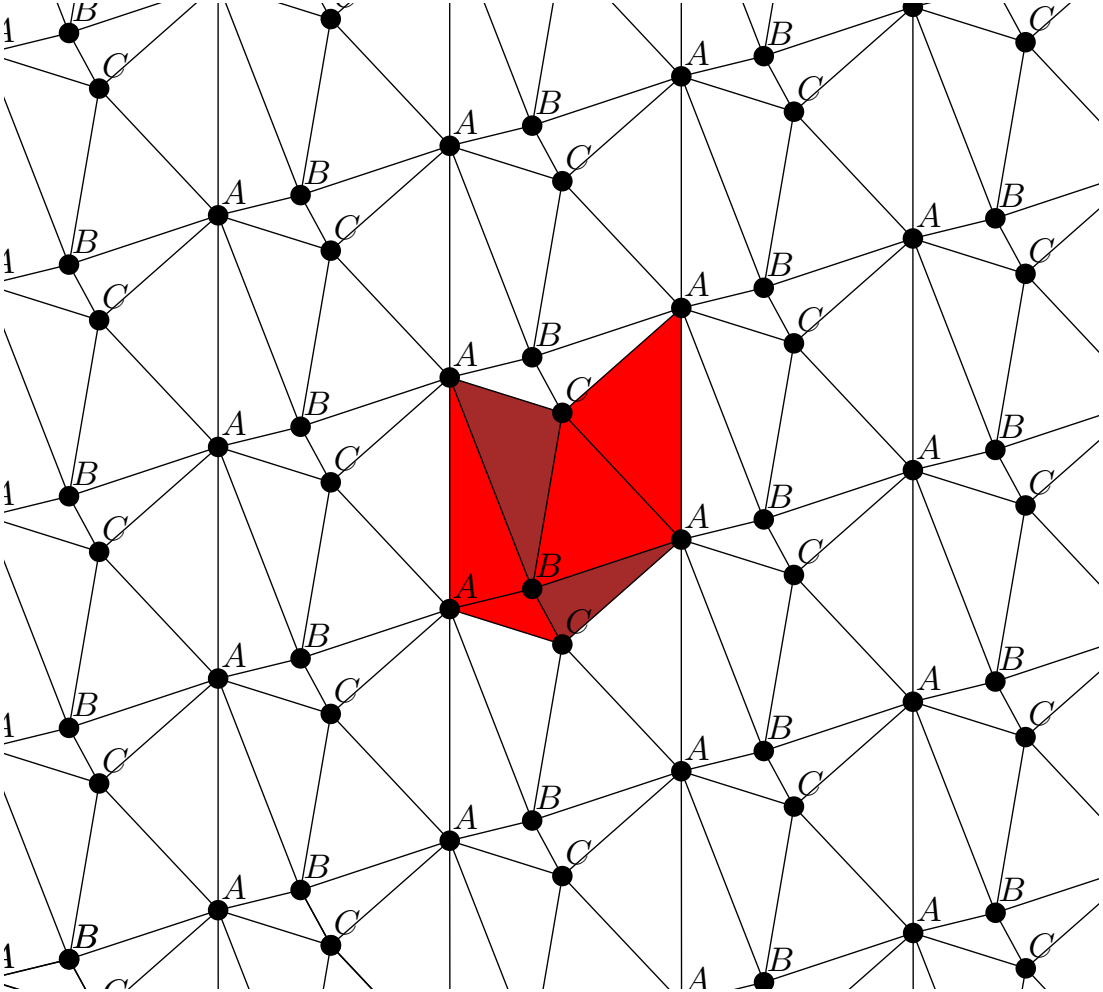
$\rightarrow$ Forming periodic point set ΛX (in 2D or 3D)

Output:

- Periodic triangulation

$\rightarrow$ Representative cells (one per equivalence class)

Periodic Delaunay triangulations



Input:

- Lattice Λ with basis B
- Representative points X

$\rightarrow$ Forming periodic point set ΛX (in 2D or 3D)

Output:

- Periodic triangulation

$\rightarrow$ Representative cells (one per equivalence class)

$\rightarrow$ with vertex translation information

Applications

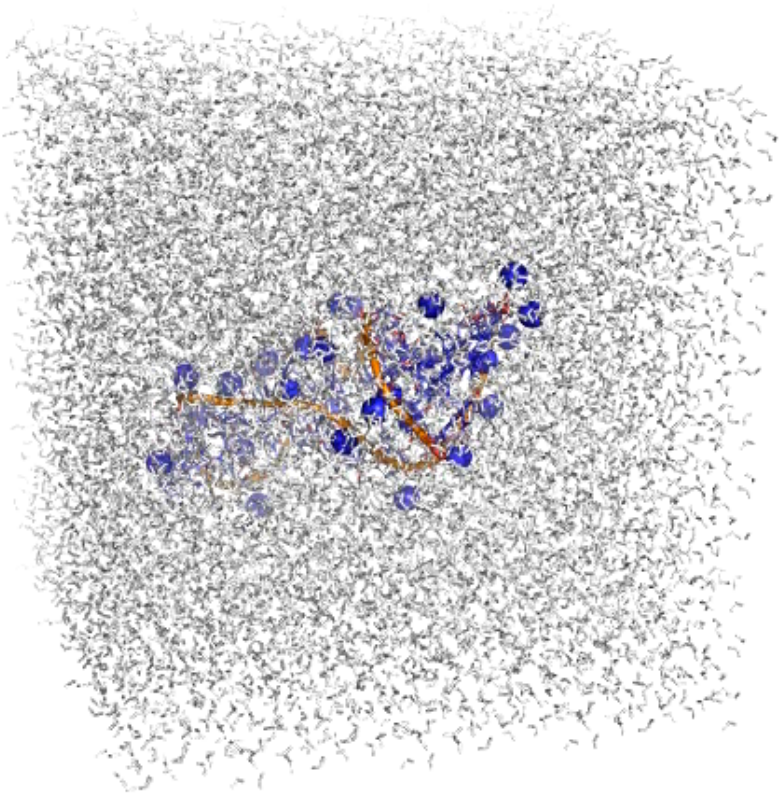


Image courtesy of Julie Bernauer

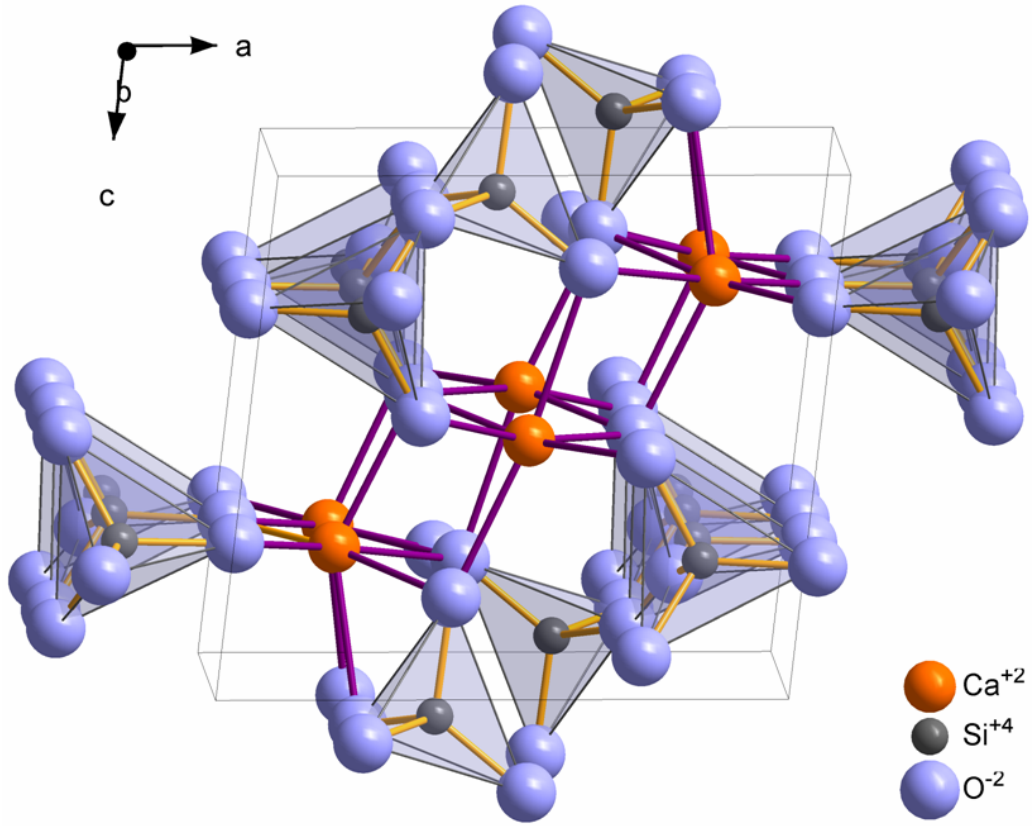


Image by Solid State, created with Diamond 3.1, CC BY-SA 3.0, `https://commons.wikimedia.org/w/index.php?curid=841165`

Existing implementations

Voro++ (3D, orthogonal basis), Zeo++ (3D, generic):

- focus on particle systems (e.g. Voronoi cell statistics, ...)

Existing implementations

Voro++ (3D, orthogonal basis), Zeo++ (3D, generic):

- focus on particle systems (e.g. Voronoi cell statistics, ...)

CGAL periodic triangulations (2D & 3D, cubic):

- Flexible design

Existing implementations

Voro++ (3D, orthogonal basis), Zeo++ (3D, generic):

- focus on particle systems (e.g. Voronoi cell statistics, ...)

CGAL periodic triangulations (2D & 3D, cubic):

- Flexible design
  - Geometry kernel: allows for e.g. exact arithmetics
  - Vertex/Cell_base: can attach custom information to vertices/cells
  - ...

Existing implementations

Voro++ (3D, orthogonal basis), Zeo++ (3D, generic):

- focus on particle systems (e.g. Voronoi cell statistics, ...)

CGAL periodic triangulations (2D & 3D, cubic):

- Flexible design
  - Geometry kernel: allows for e.g. exact arithmetics
  - Vertex/Cell_base: can attach custom information to vertices/cells
  - ...
- Access to triangulation primitives

Existing implementations

Voro++ (3D, orthogonal basis), Zeo++ (3D, generic):

- focus on particle systems (e.g. Voronoi cell statistics, ...)

CGAL periodic triangulations (2D & 3D, cubic):

- Flexible design
  - Geometry kernel: allows for e.g. exact arithmetics
  - Vertex/Cell_base: can attach custom information to vertices/cells
  - ...
- Access to triangulation primitives
  - vertex, edge, face and cell iterators
  - adjacency relations between cells
  - edges, faces and cells incident to a vertex

Existing implementations

Voro++ (3D, orthogonal basis), Zeo++ (3D, generic):

- focus on particle systems (e.g. Voronoi cell statistics, ...)

CGAL periodic triangulations (2D & 3D, cubic):

- Flexible design
  - Geometry kernel: allows for e.g. exact arithmetics
  - Vertex/Cell_base: can attach custom information to vertices/cells
  - ...
- Access to triangulation primitives
  - vertex, edge, face and cell iterators
  - adjacency relations between cells
  - edges, faces and cells incident to a vertex

$\Rightarrow$ Goal: provide similar user interface for generic case

Current CGAL implementation

- only covers cubic periodicity

Current CGAL implementation

- only covers cubic periodicity
- points inserted incrementally
  - via Bowyer-Watson algorithm

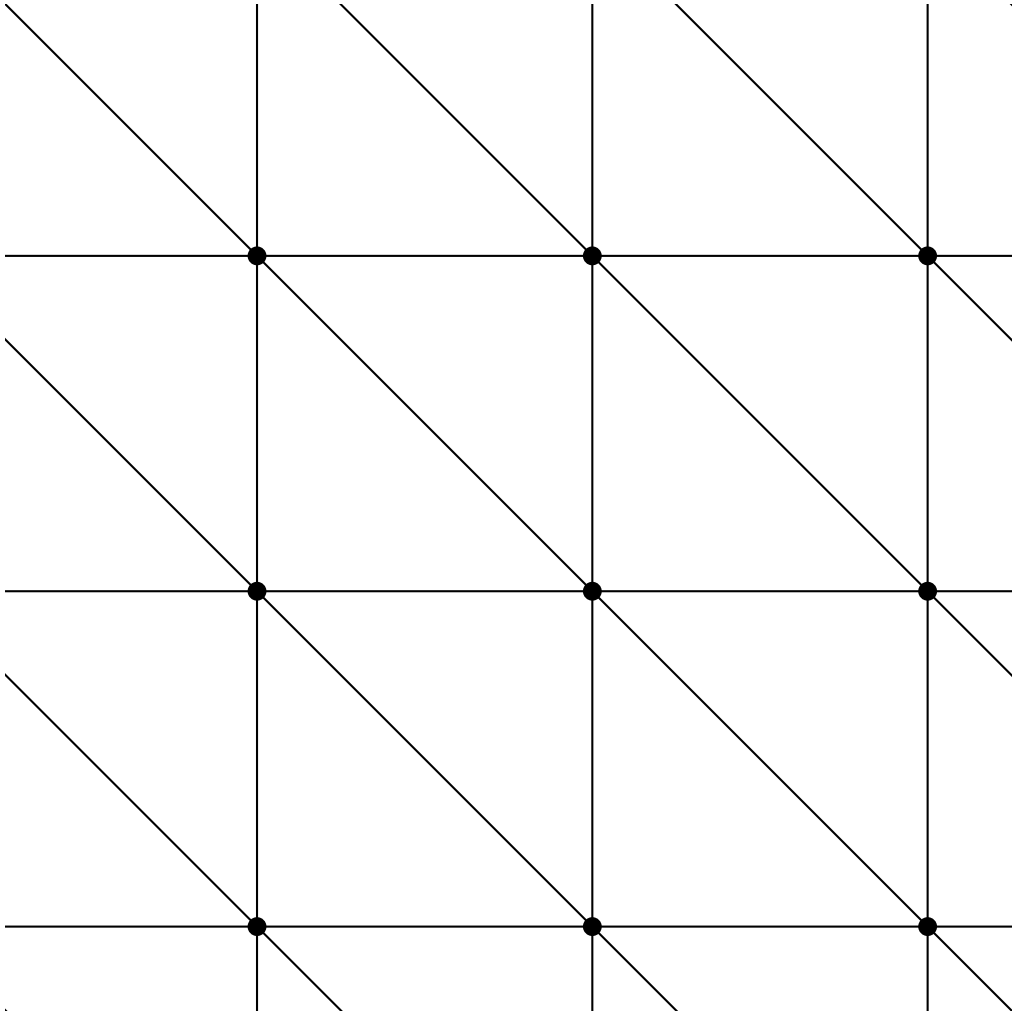
Current CGAL implementation

- only covers cubic periodicity
- points inserted incrementally
  - via Bowyer-Watson algorithm
- periodic triangulation data structure

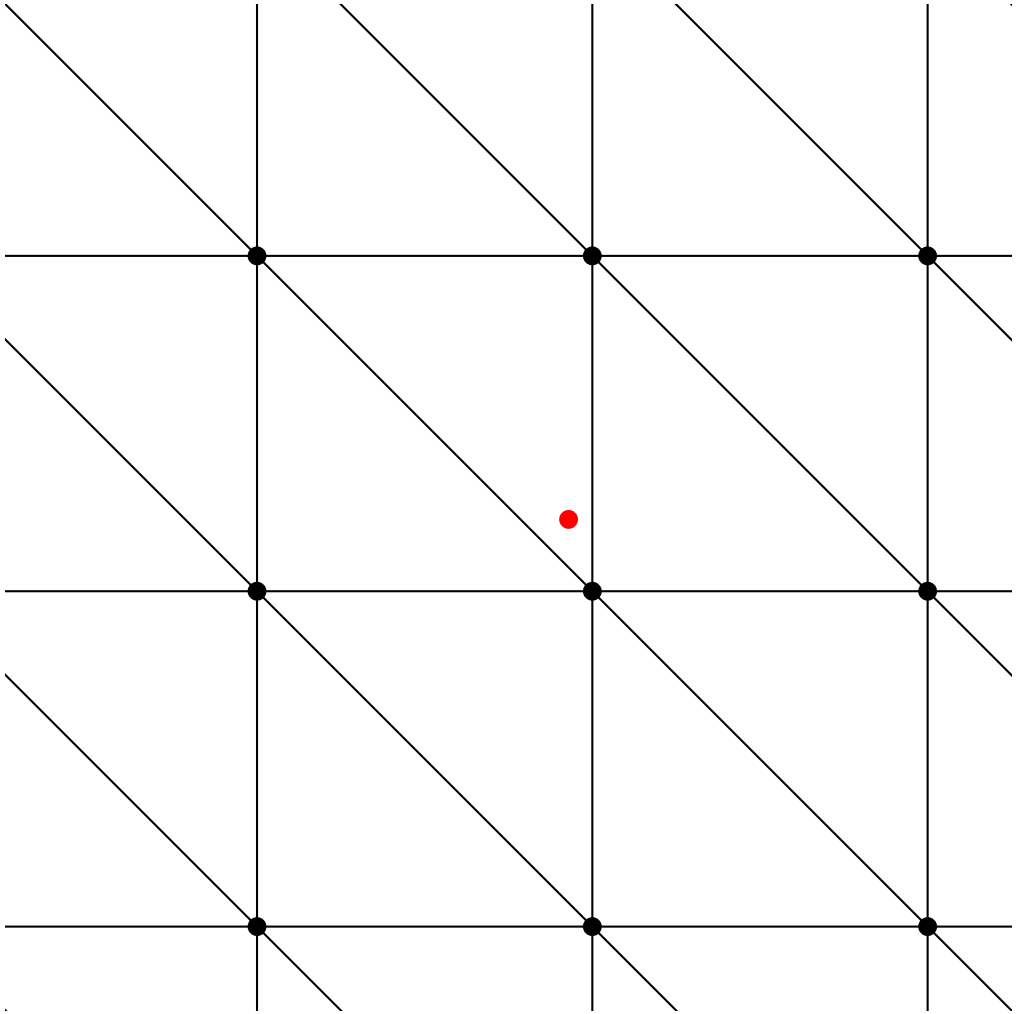
Current CGAL implementation

- only covers cubic periodicity
- points inserted incrementally
  - via Bowyer-Watson algorithm
- periodic triangulation data structure
  - initially 3^d copies of everything
  - after certain criterion met, 1 copy of everything

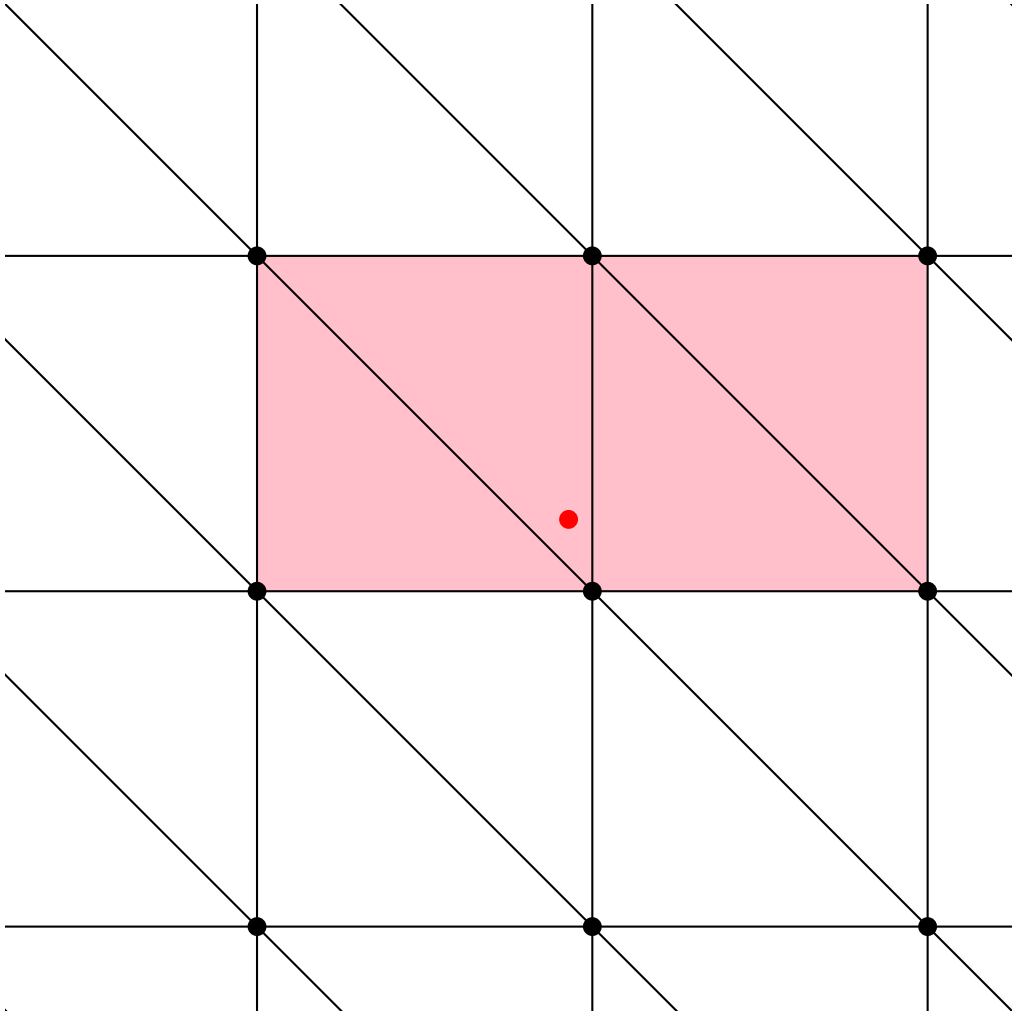
Current CGAL implementation



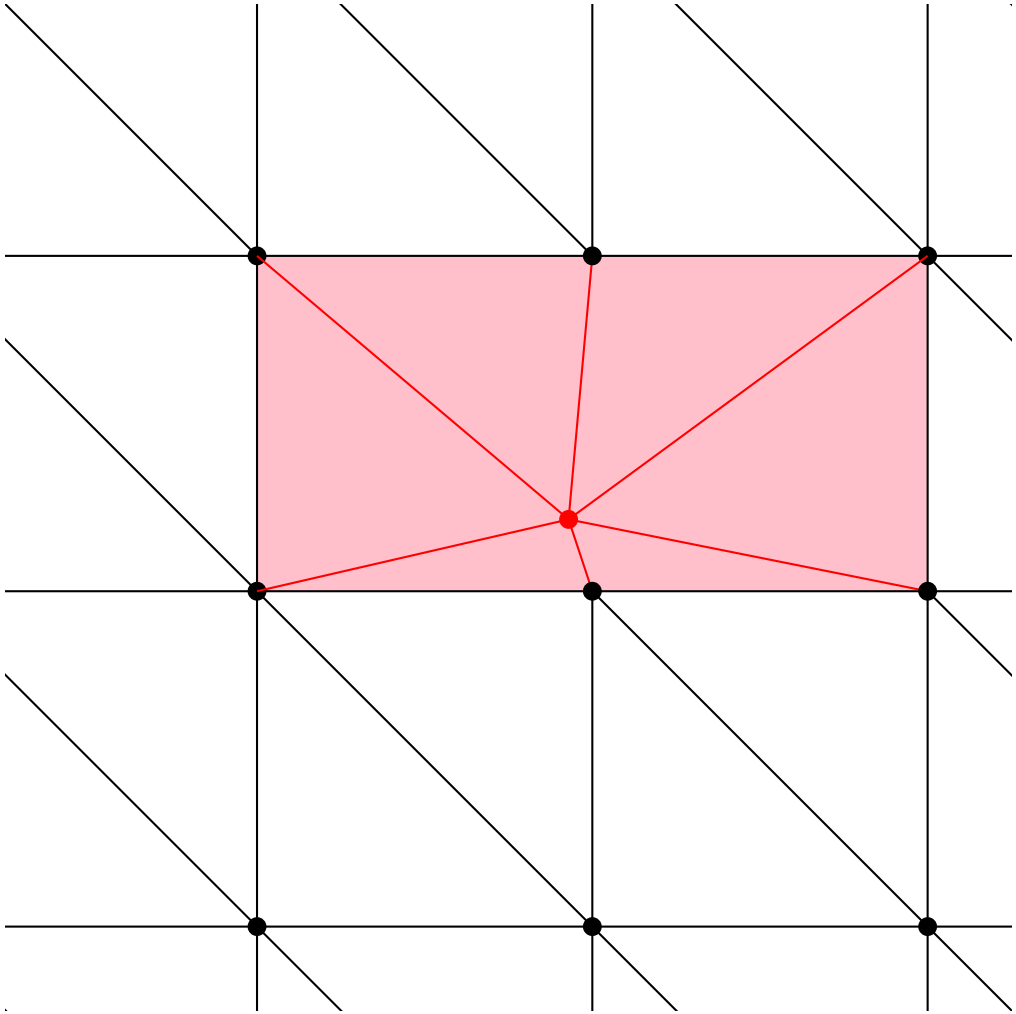
Current CGAL implementation



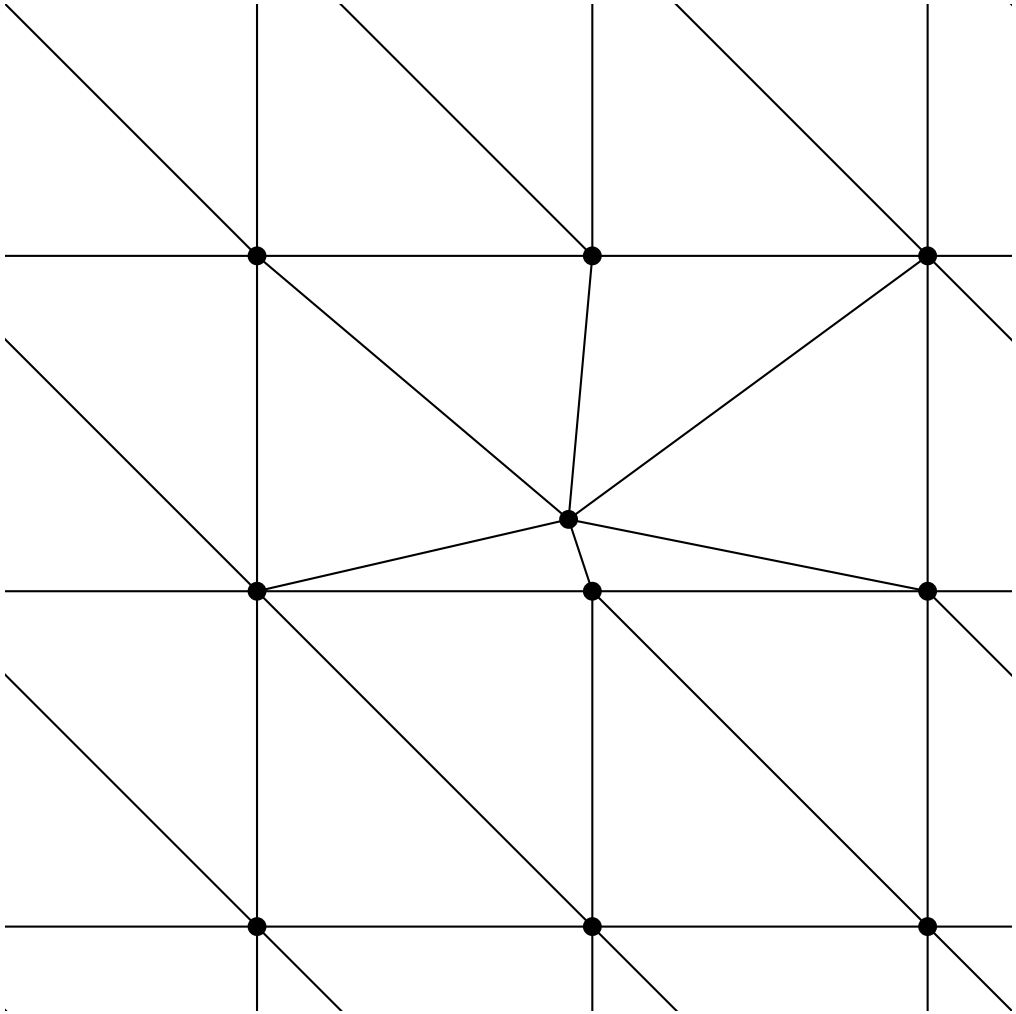
Current CGAL implementation



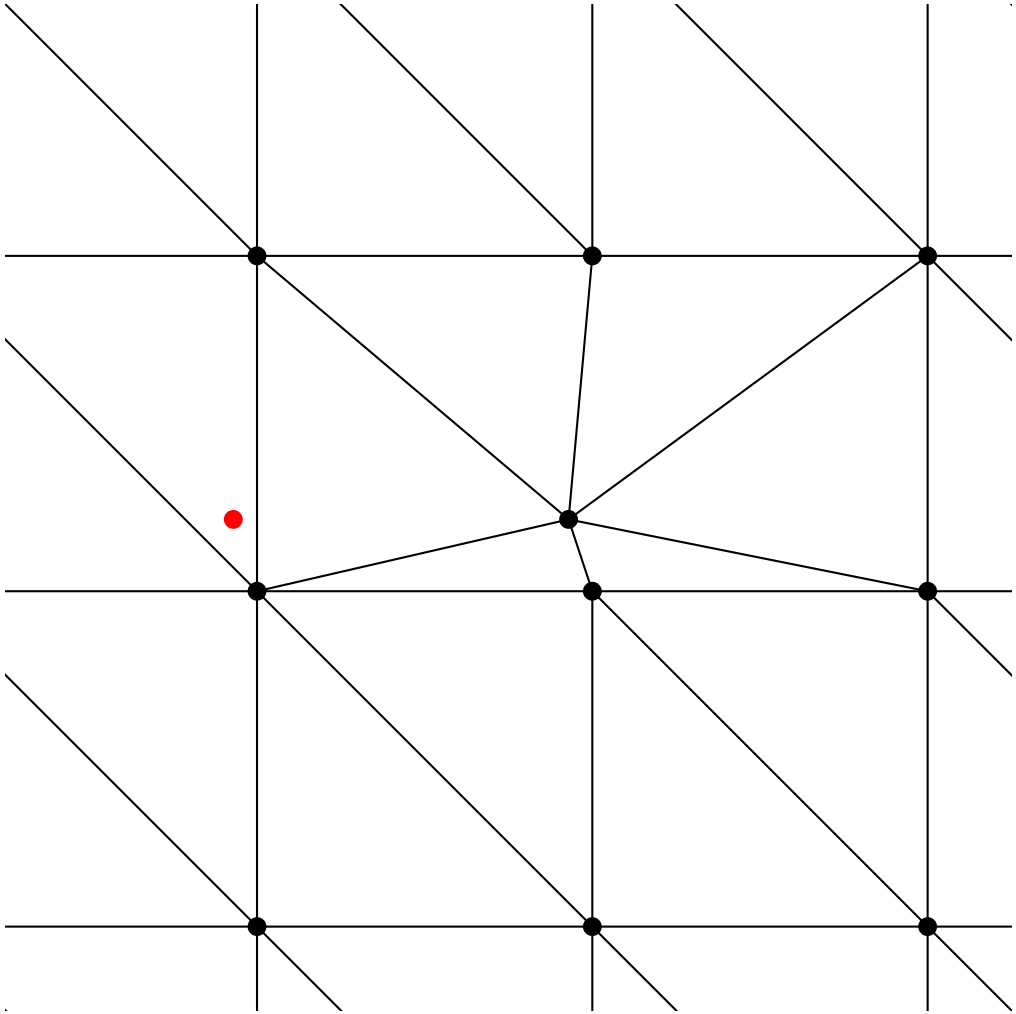
Current CGAL implementation



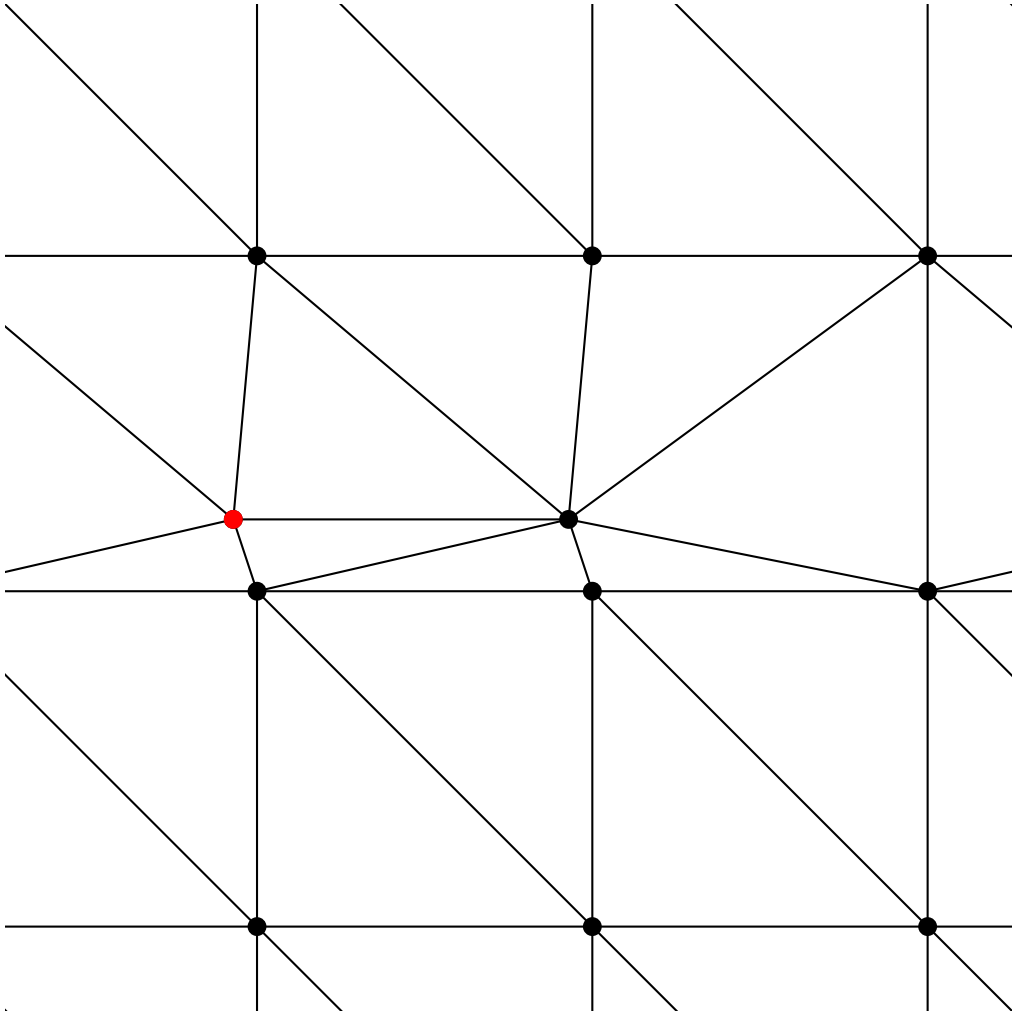
Current CGAL implementation



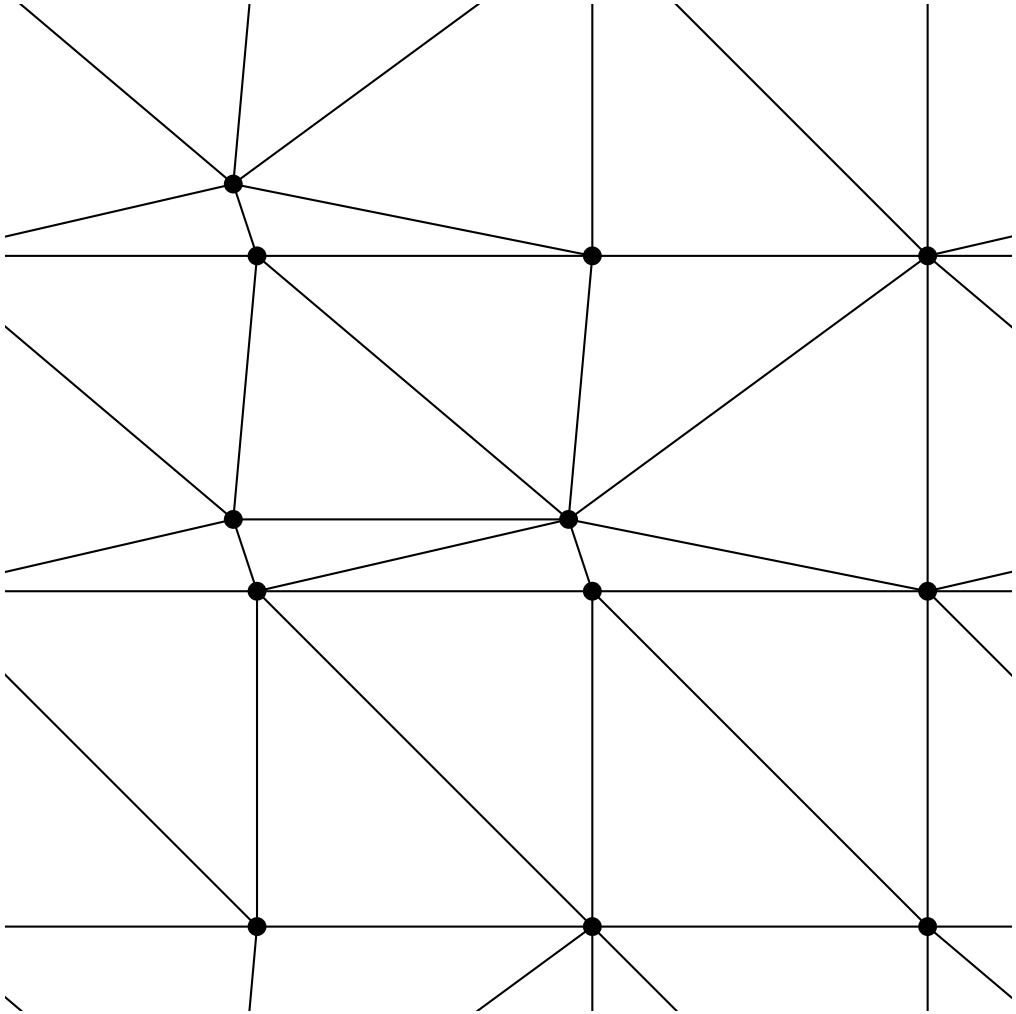
Current CGAL implementation



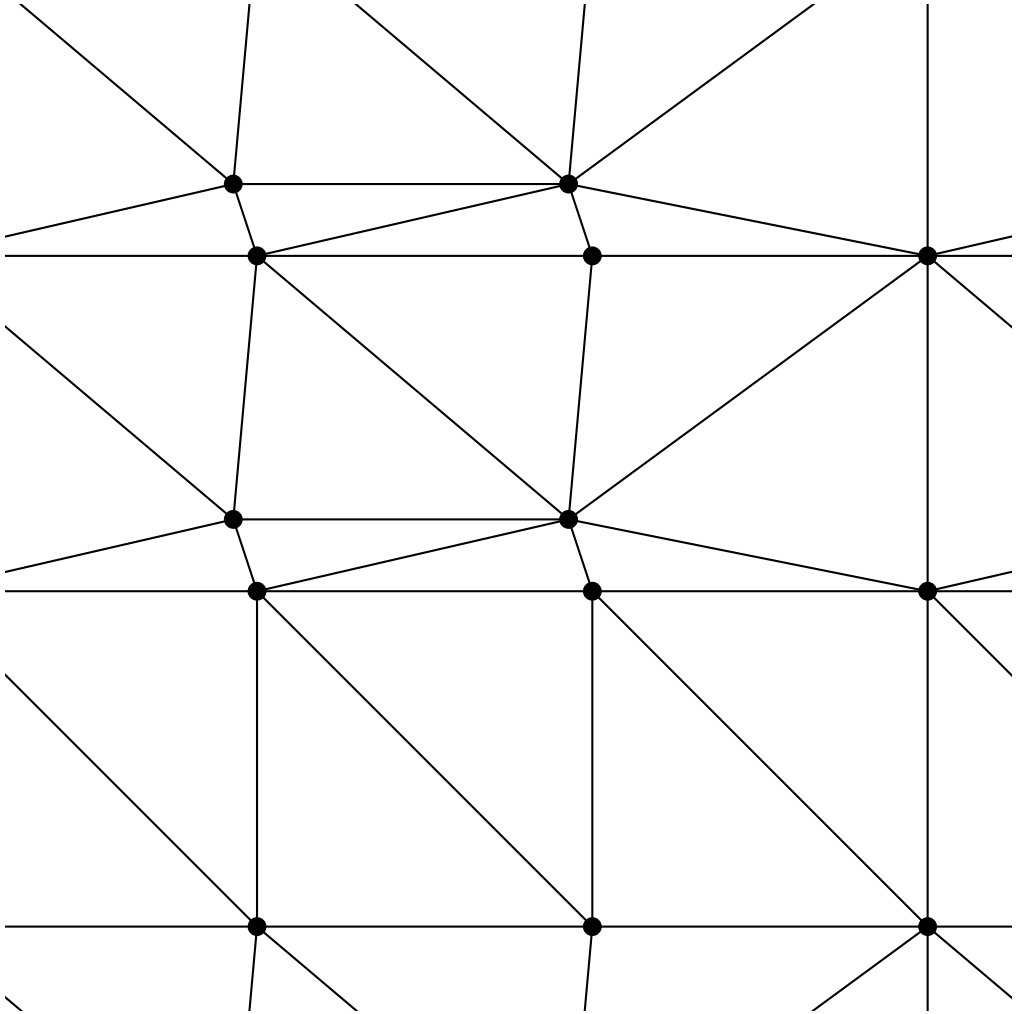
Current CGAL implementation



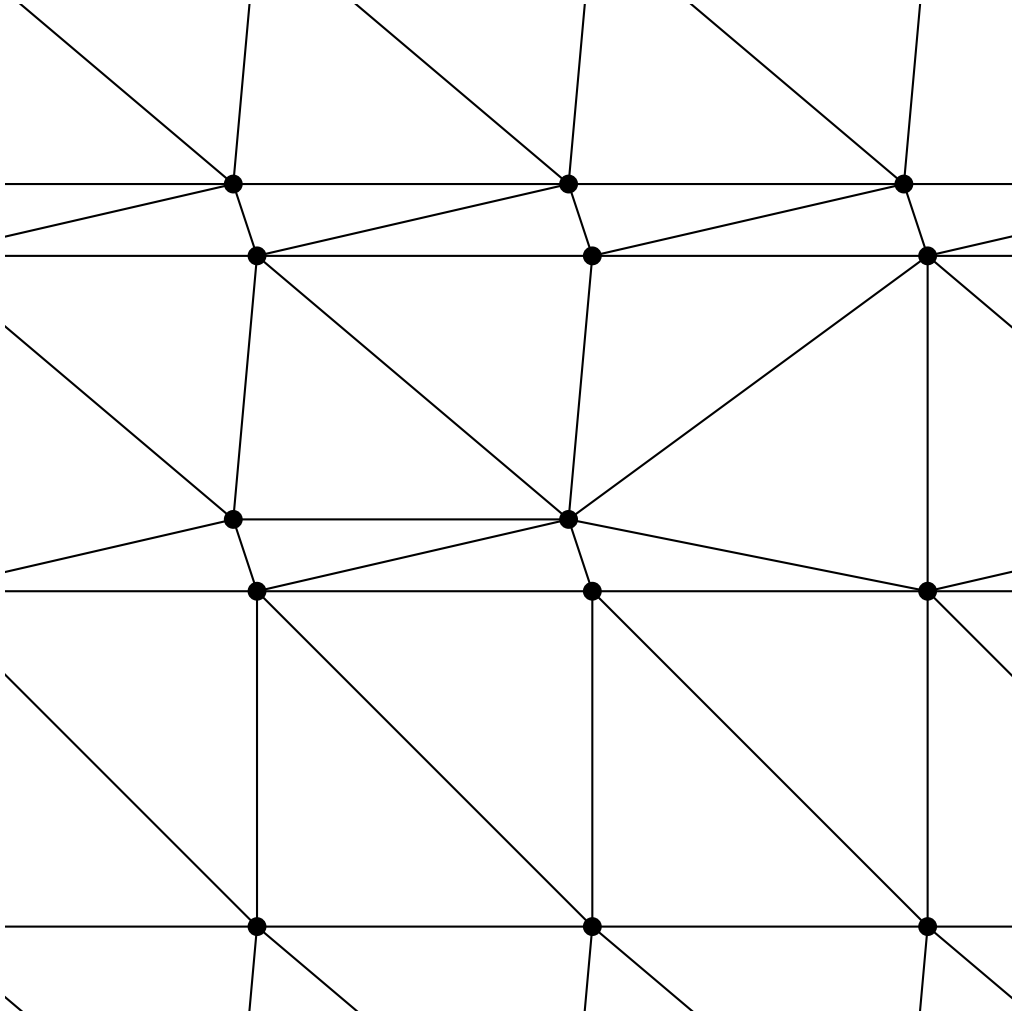
Current CGAL implementation



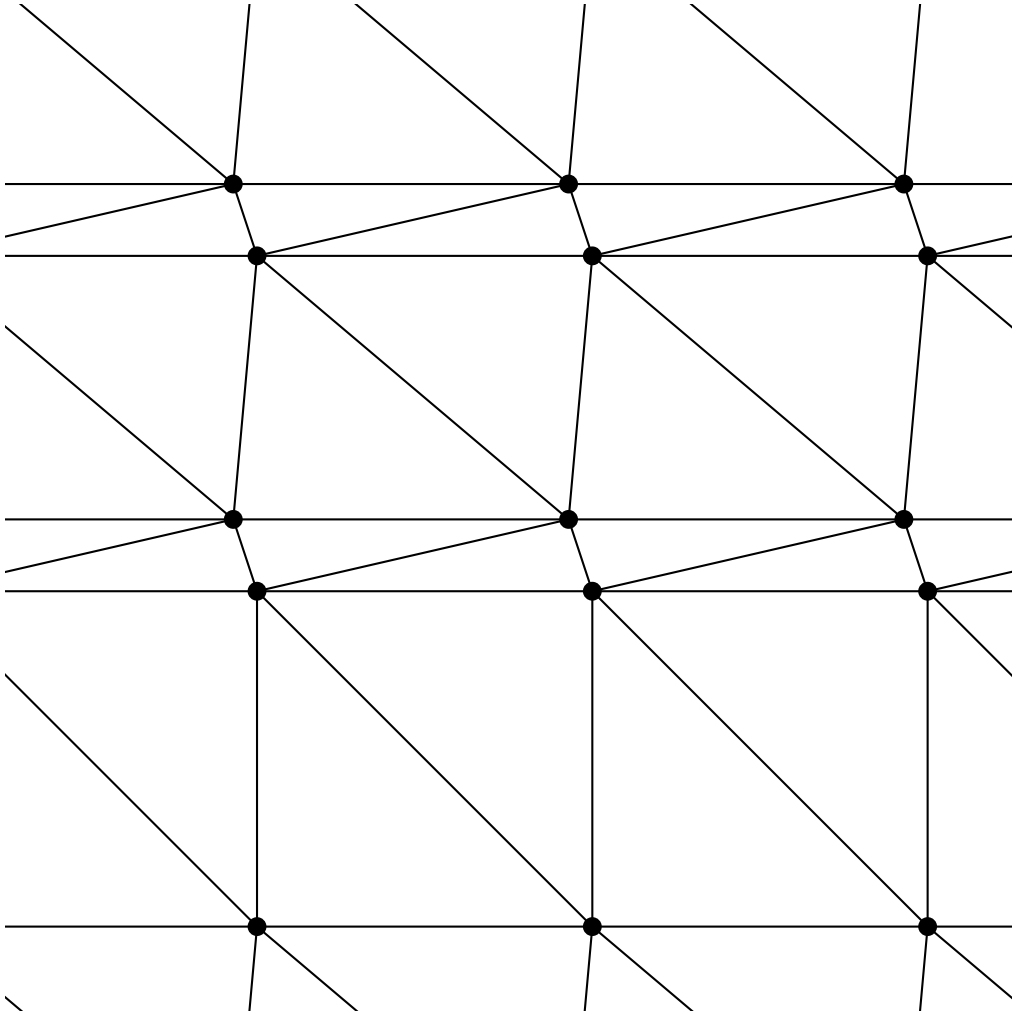
Current CGAL implementation



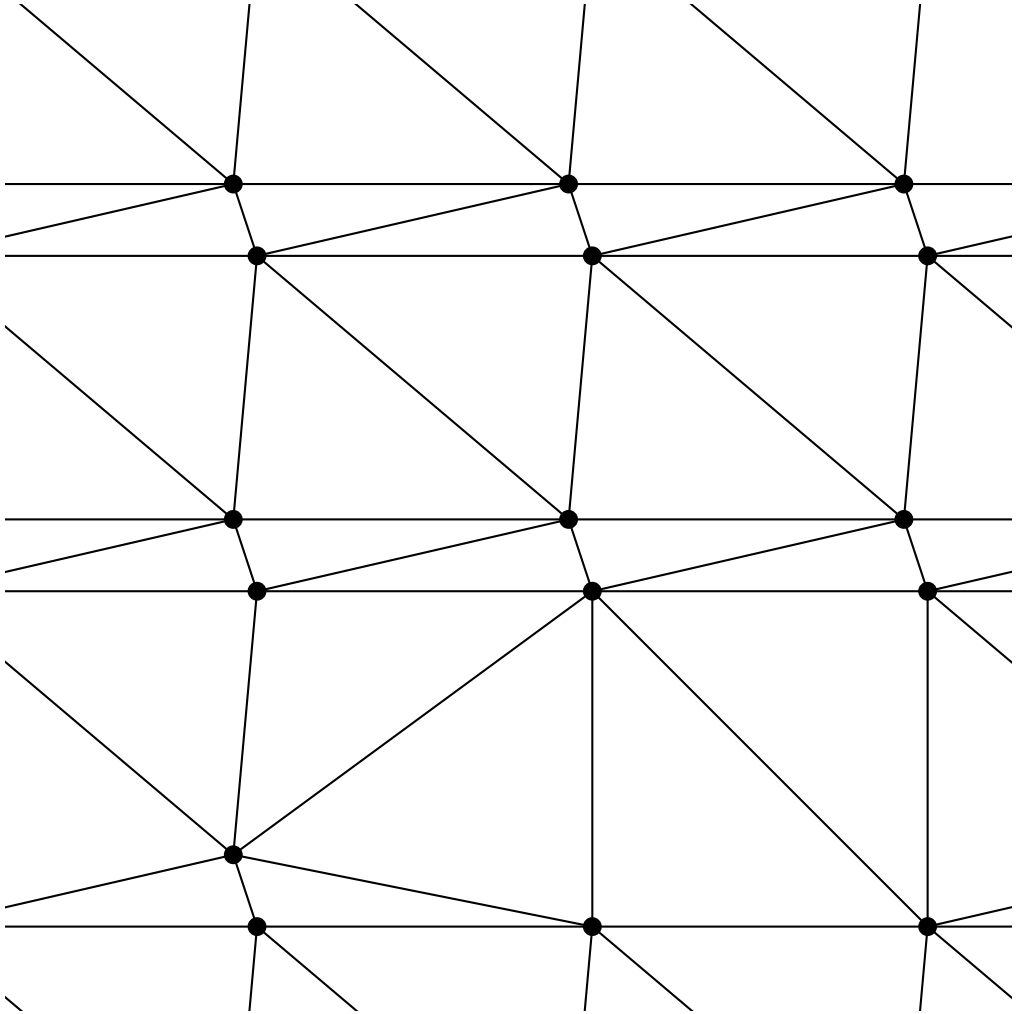
Current CGAL implementation



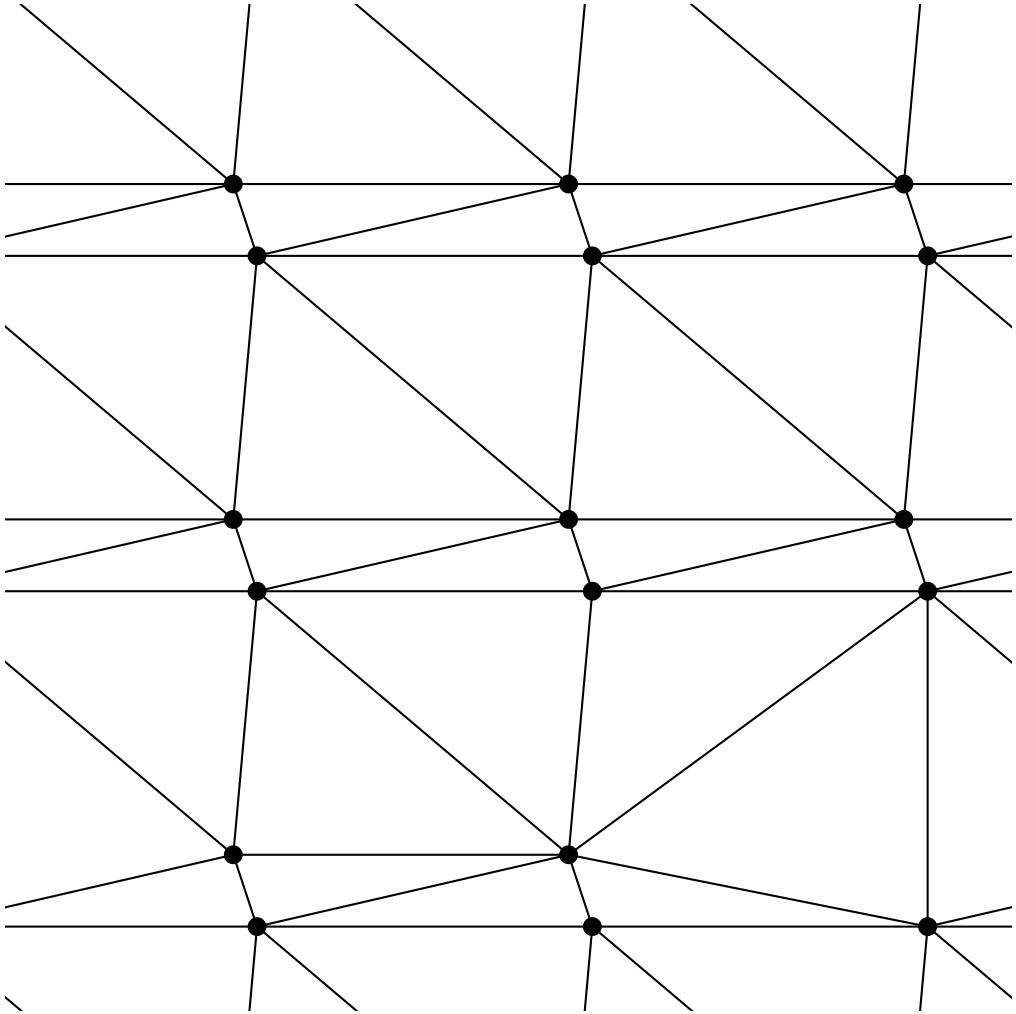
Current CGAL implementation



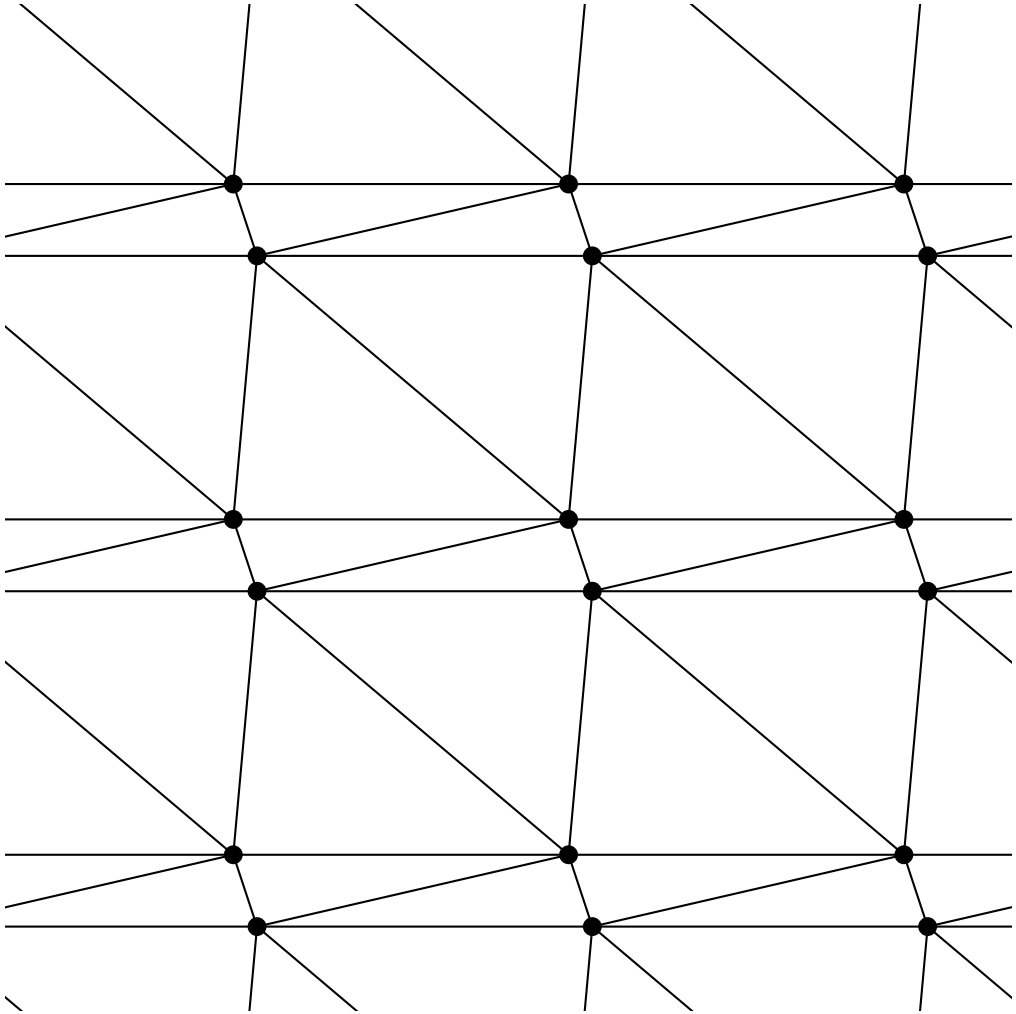
Current CGAL implementation



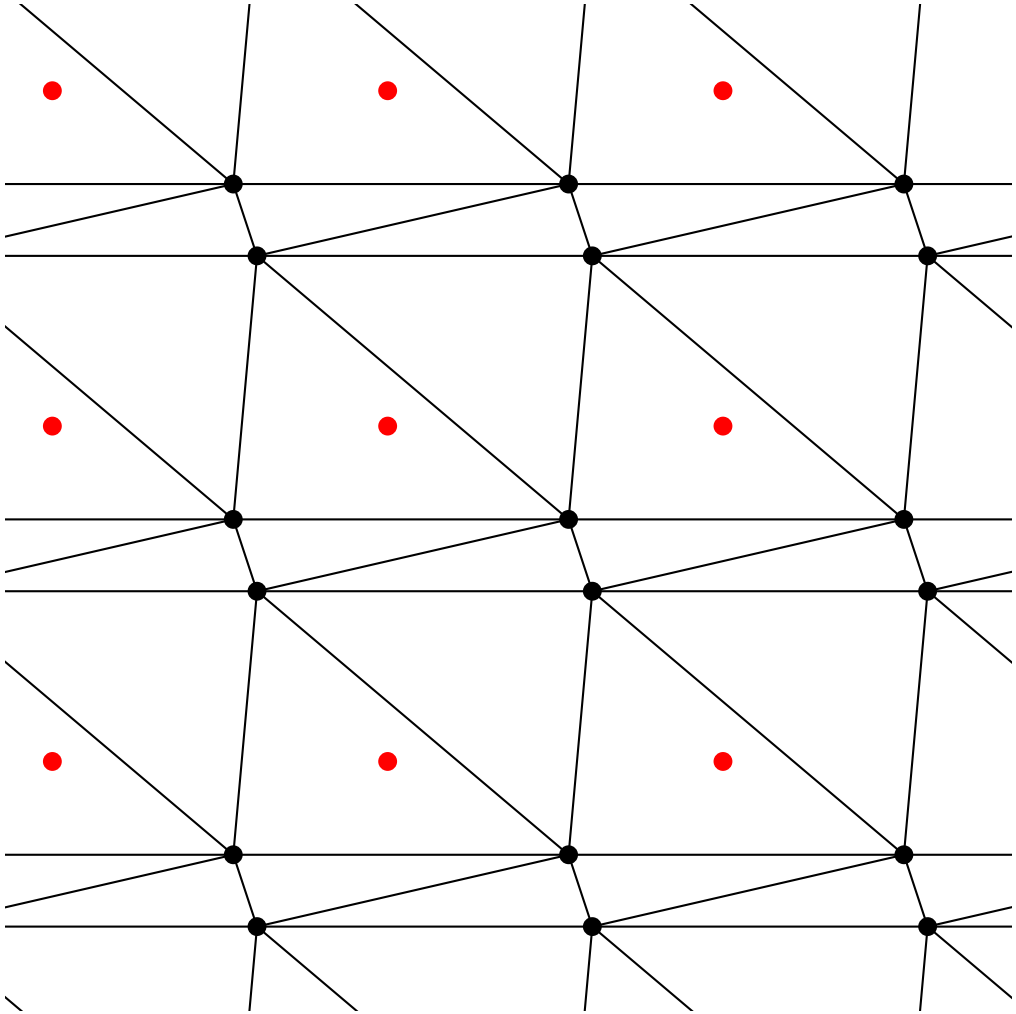
Current CGAL implementation



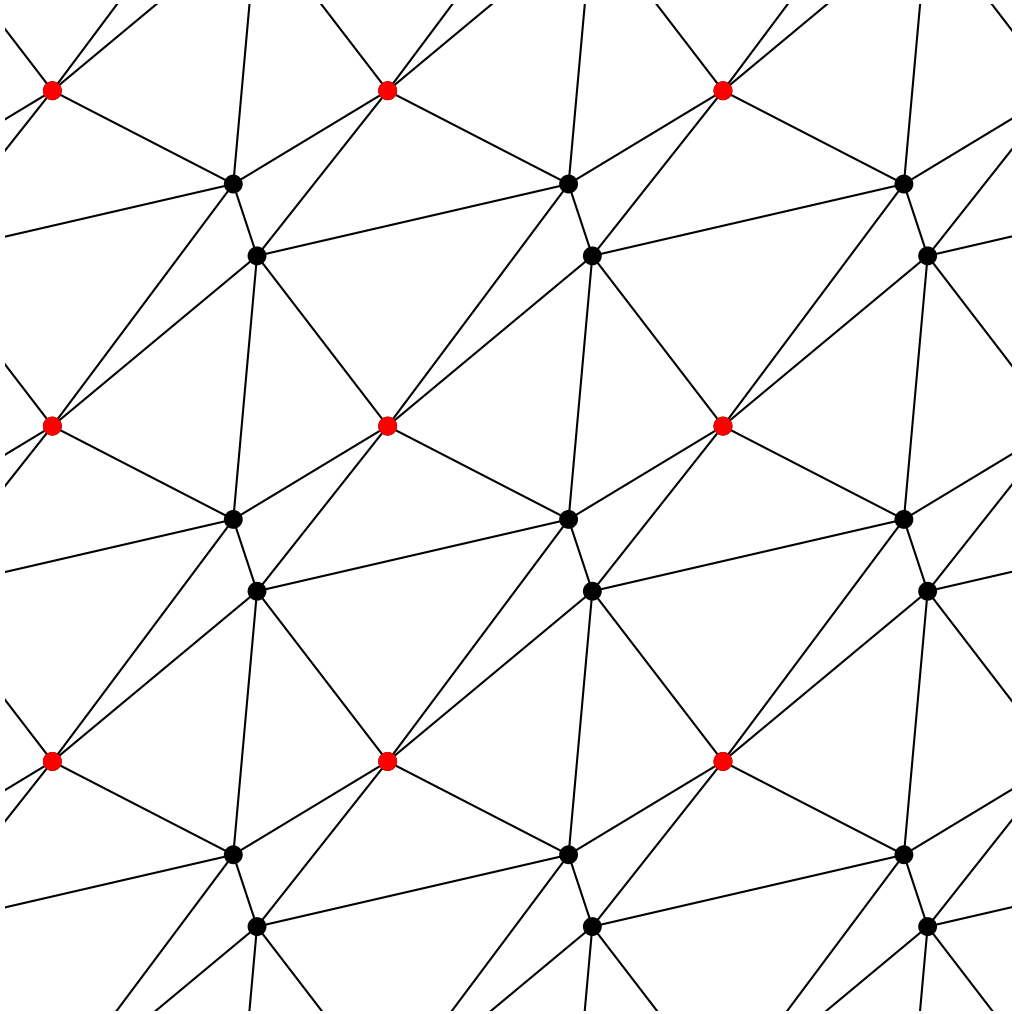
Current CGAL implementation



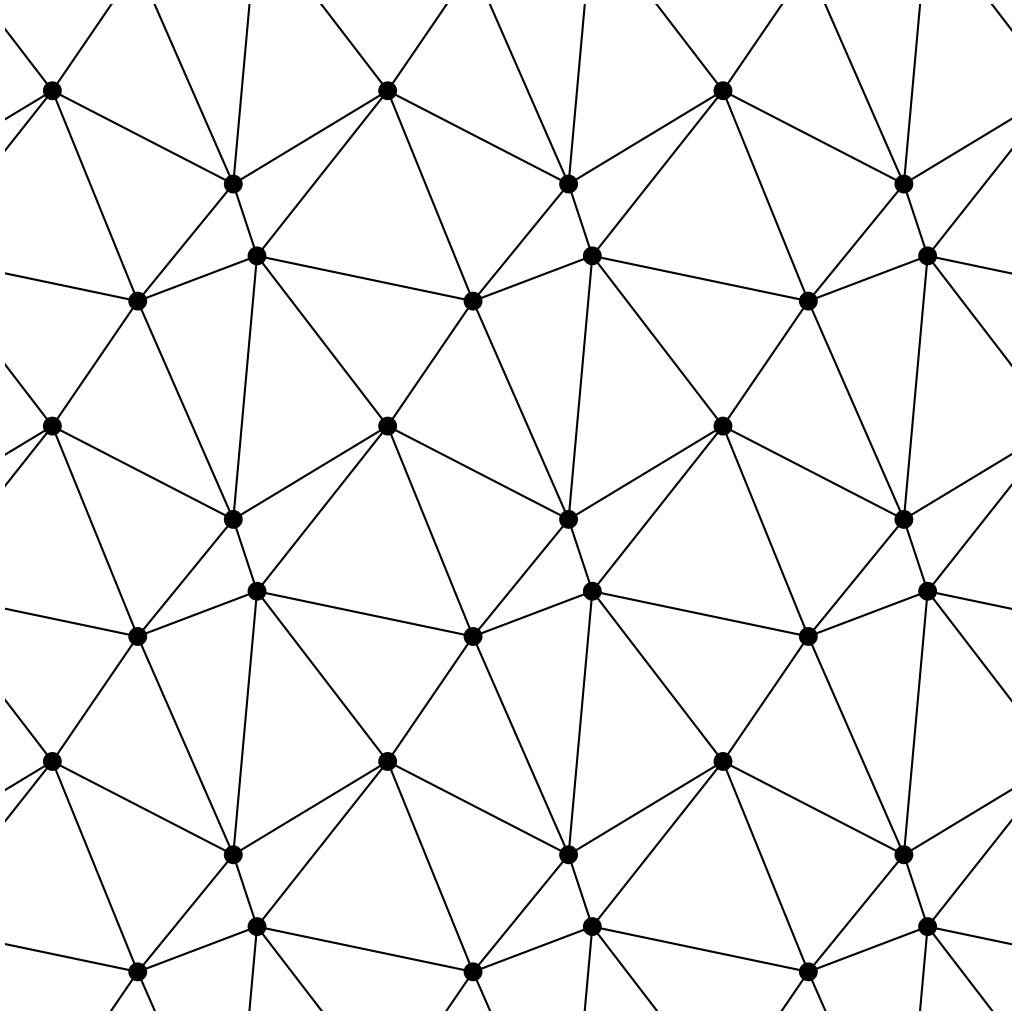
Current CGAL implementation



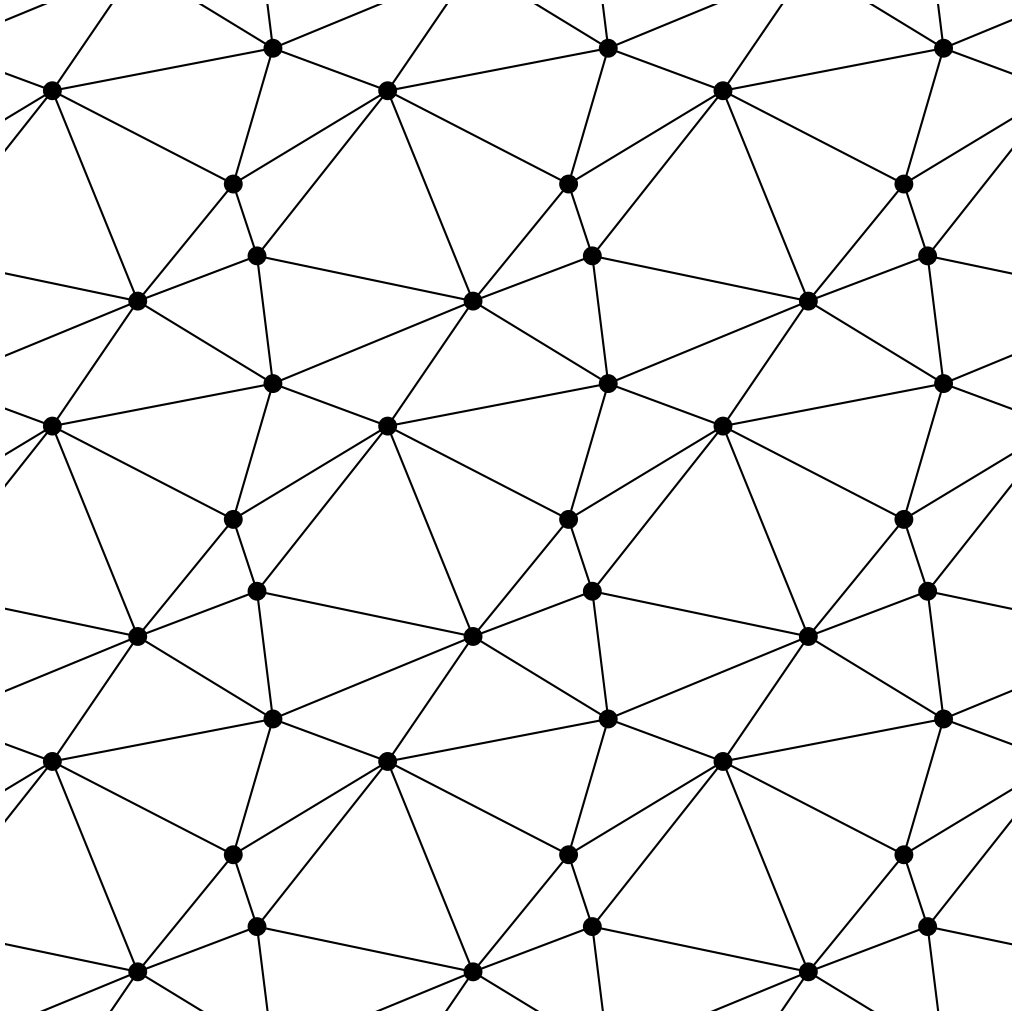
Current CGAL implementation



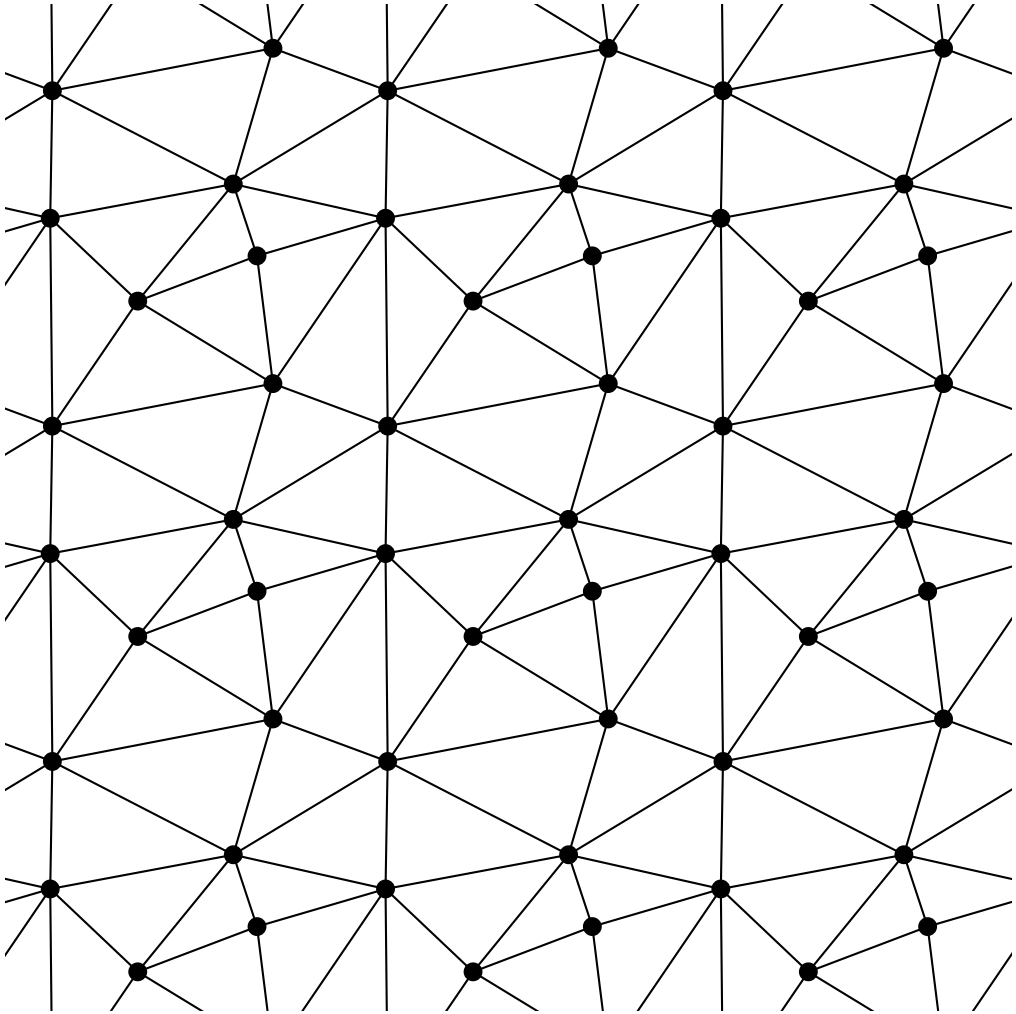
Current CGAL implementation



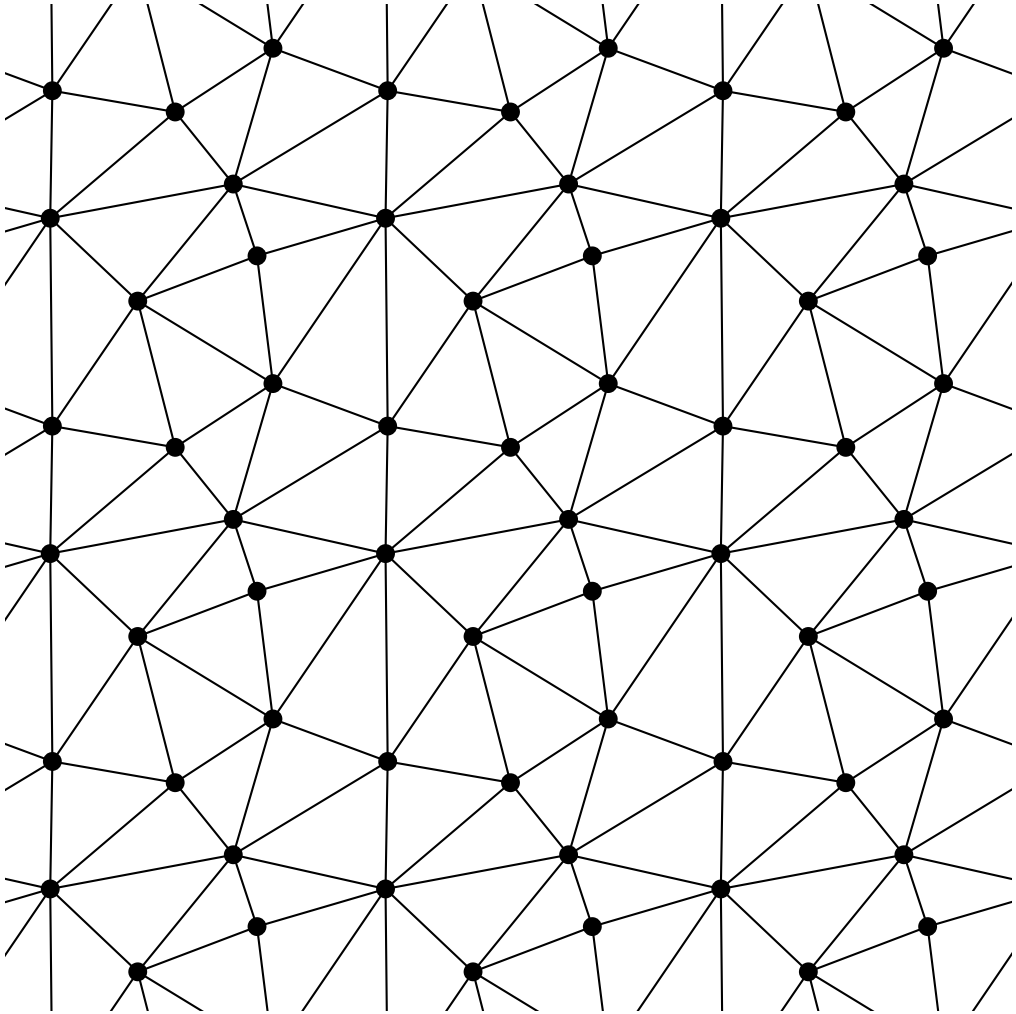
Current CGAL implementation



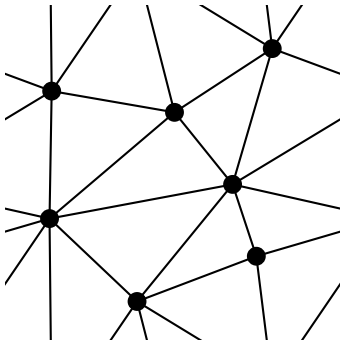
Current CGAL implementation



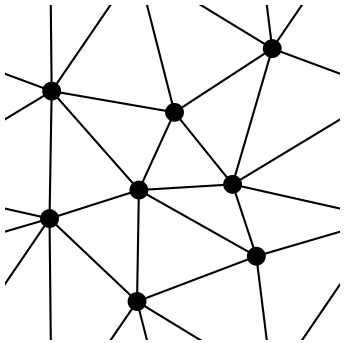
Current CGAL implementation



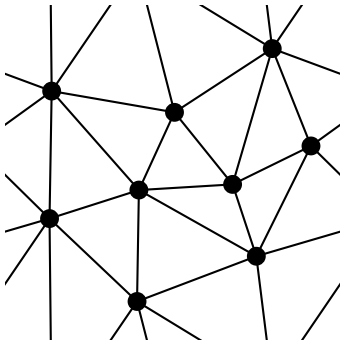
Current CGAL implementation



Current CGAL implementation

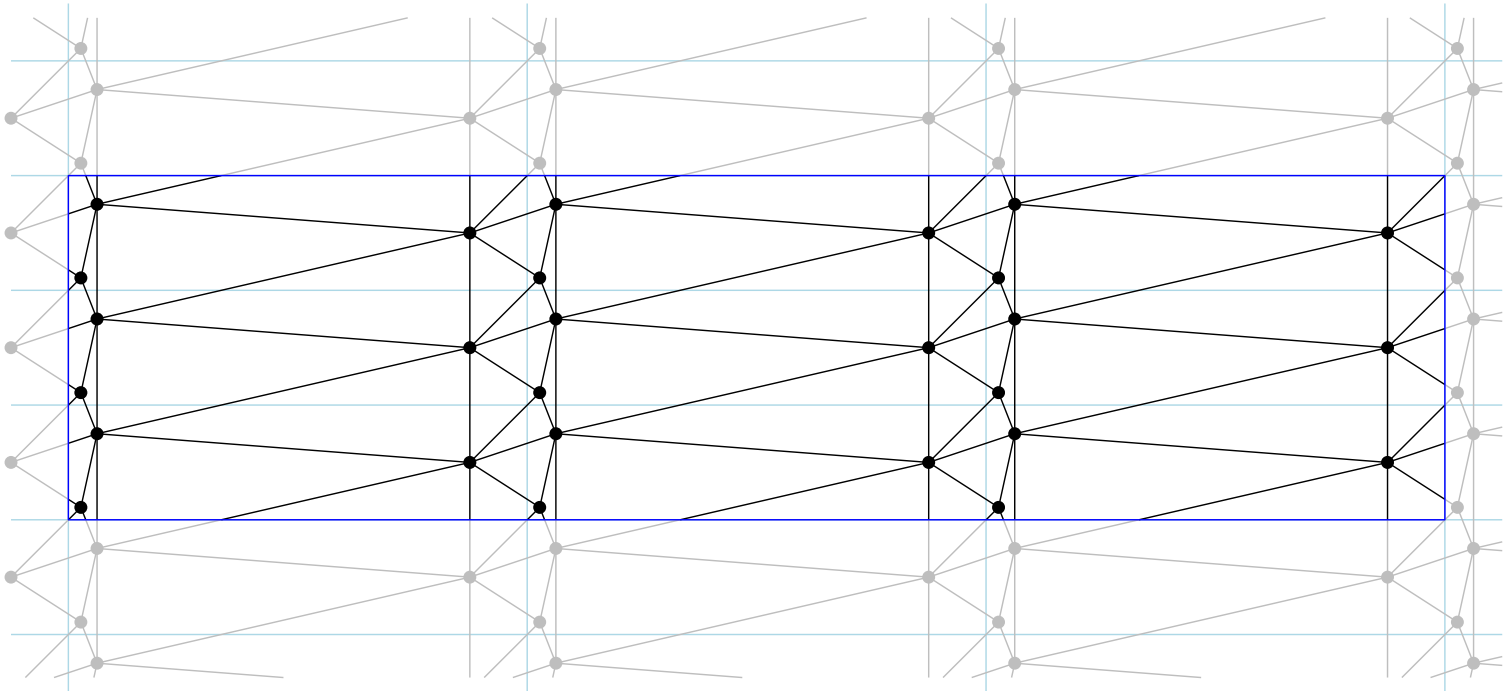


Current CGAL implementation



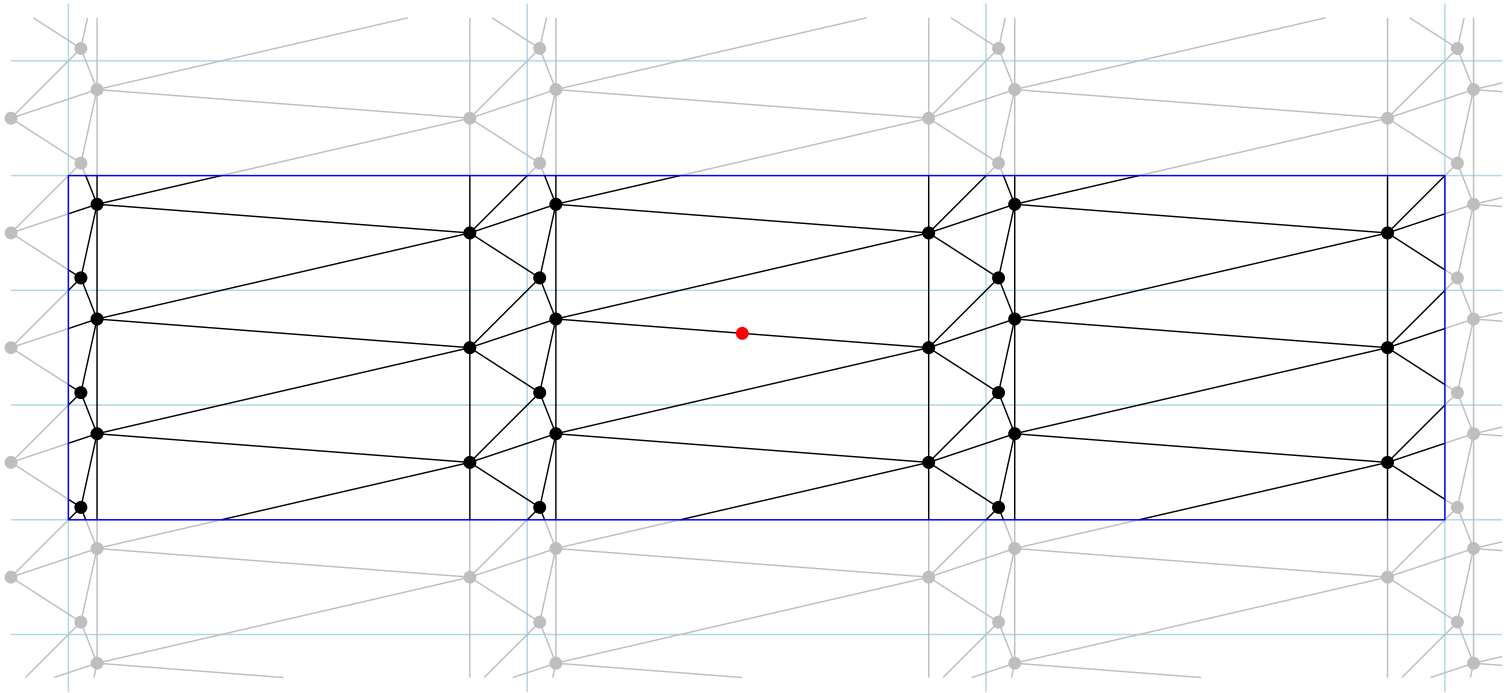
Generalization

- Approach does not directly generalize
  - initial 3^d copies not sufficient



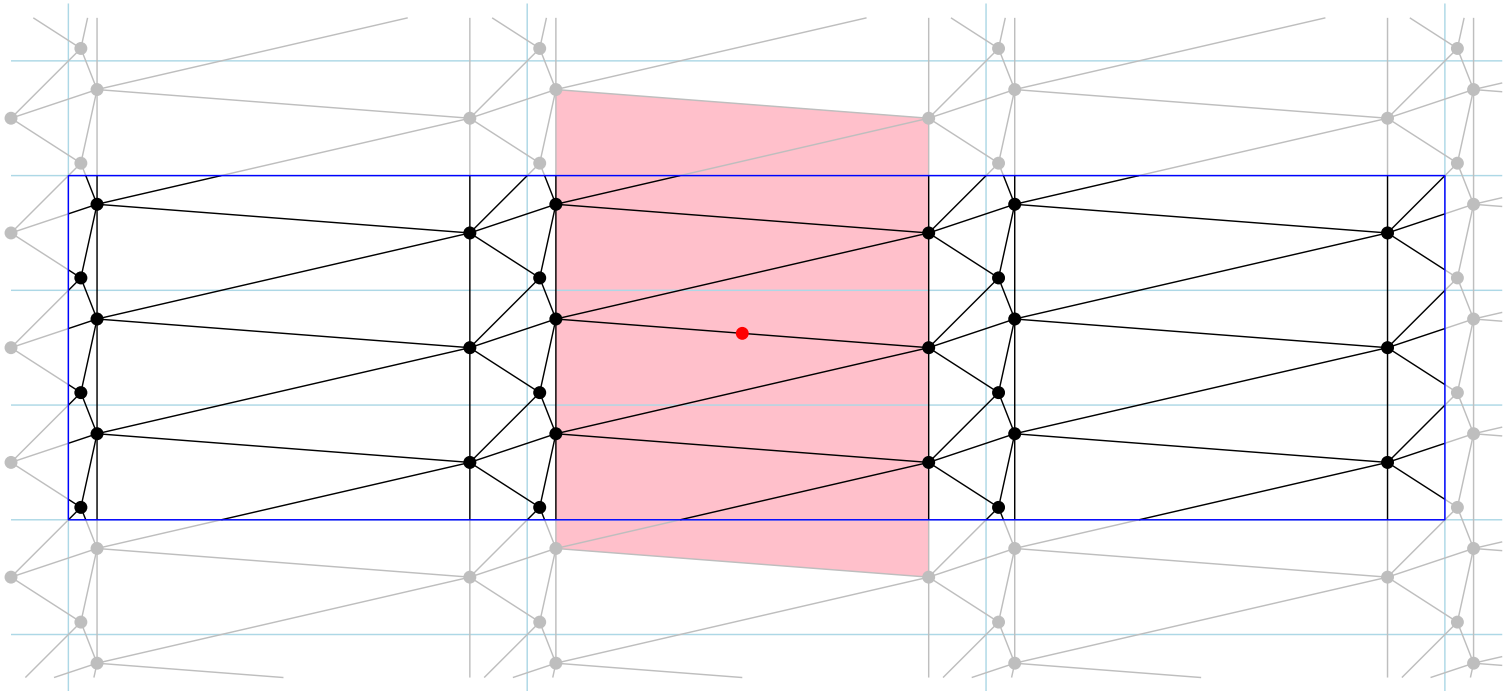
Generalization

- Approach does not directly generalize
  - initial 3^d copies not sufficient



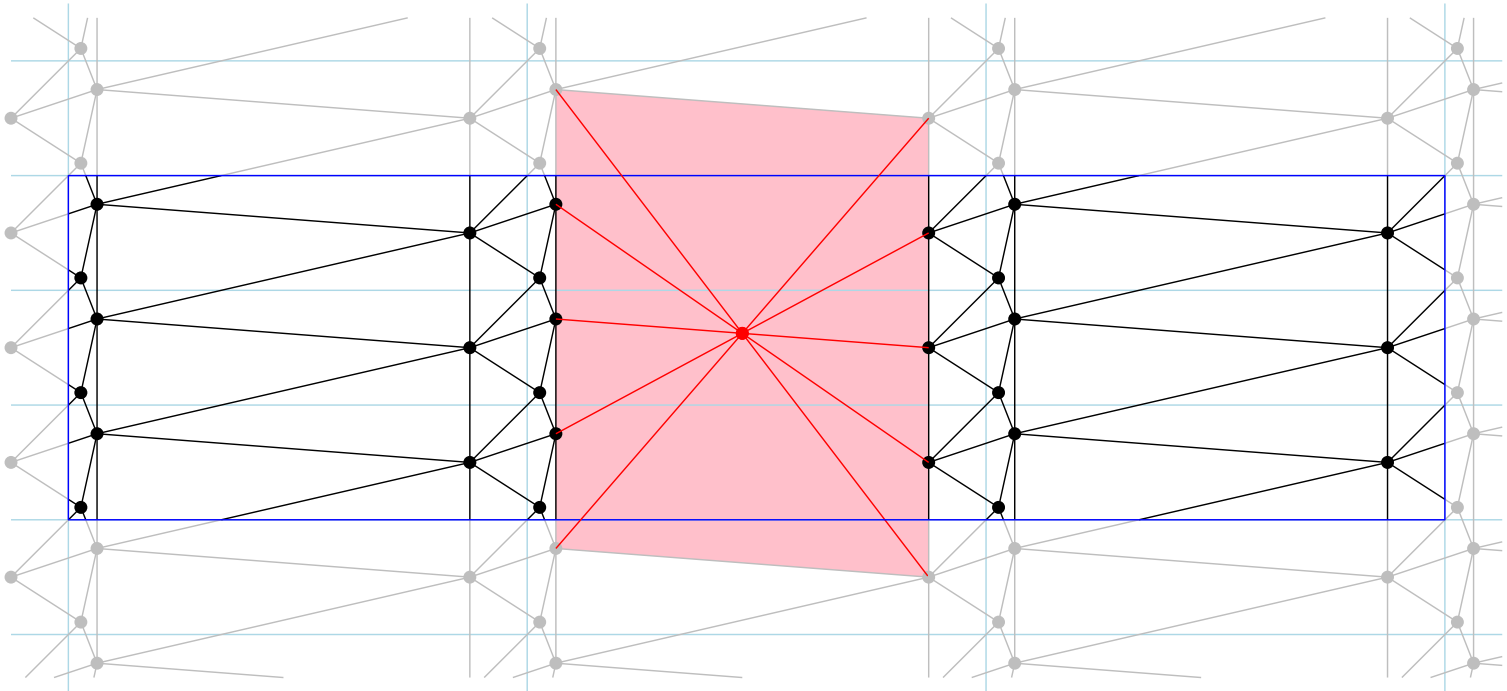
Generalization

- Approach does not directly generalize
  - initial 3^d copies not sufficient

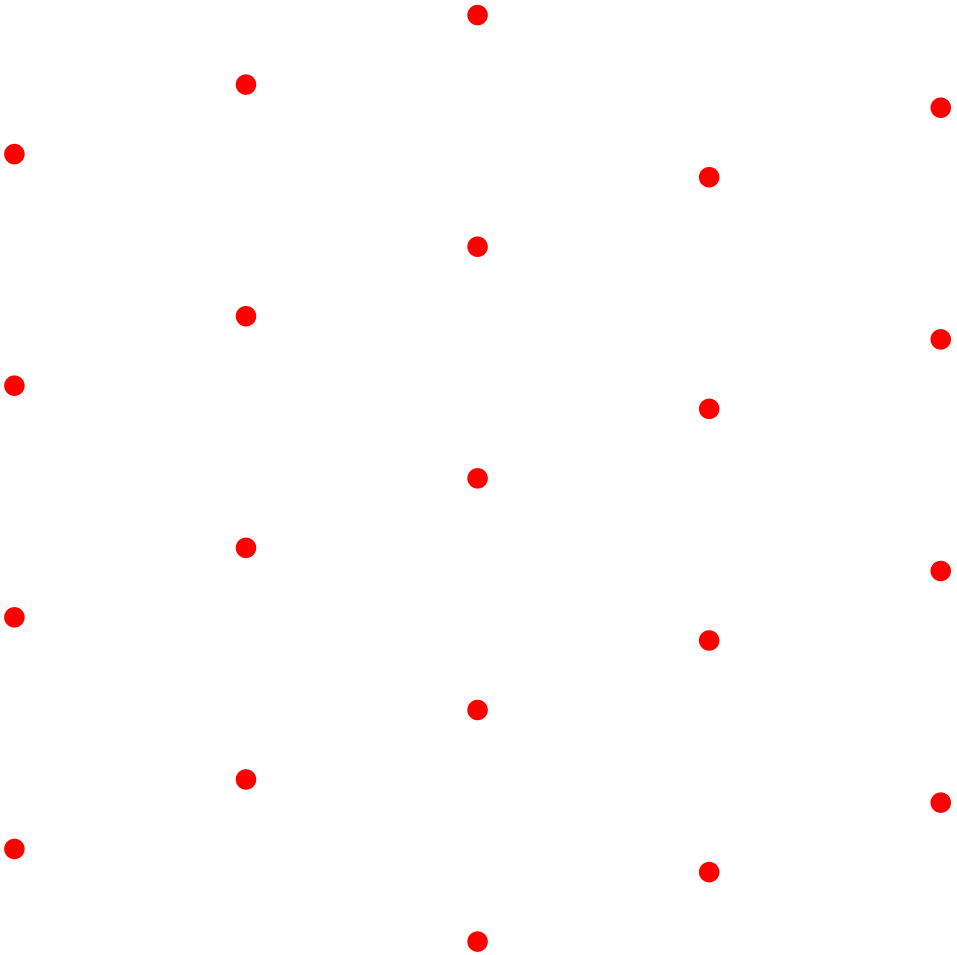


Generalization

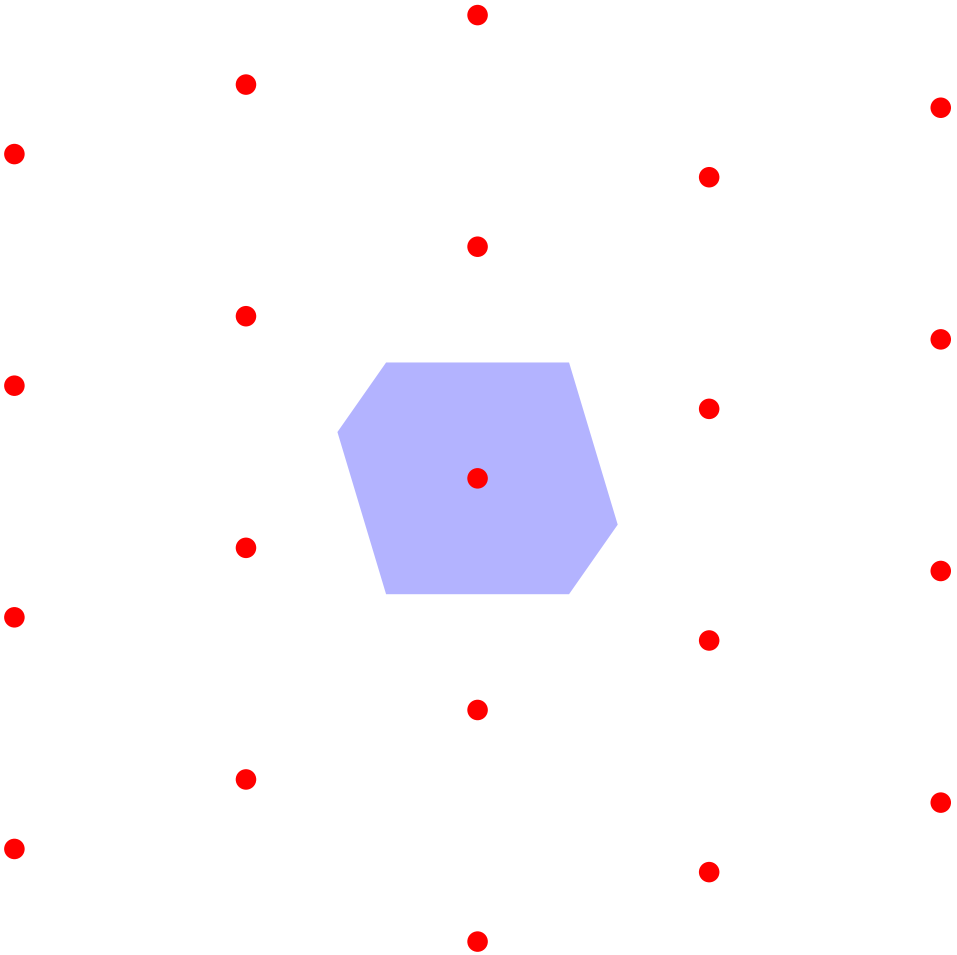
- Approach does not directly generalize
  - initial 3^d copies not sufficient



Algorithm by Dolbilin & Huson, '97

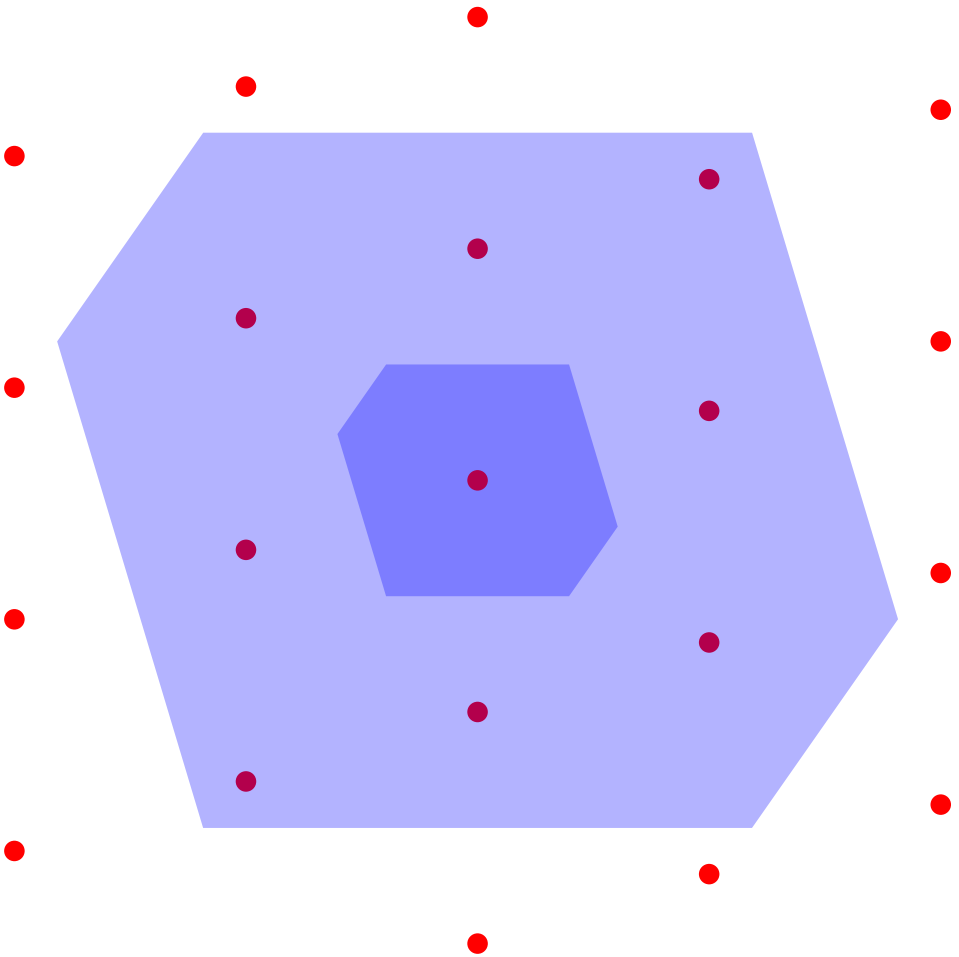


Algorithm by Dolbilin & Huson, '97



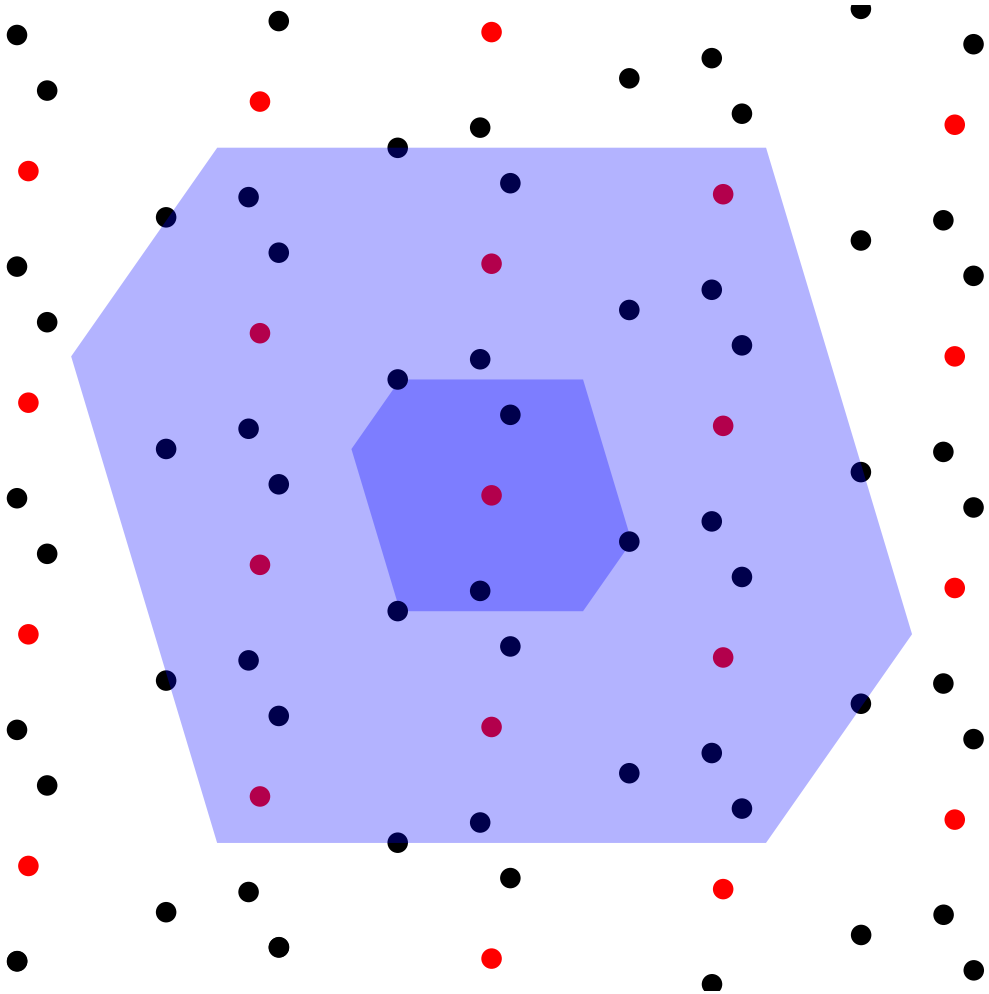
- $\mathrm{dom}(0, \Lambda)$: Voronoi domain of origin

Algorithm by Dolbilin & Huson, '97



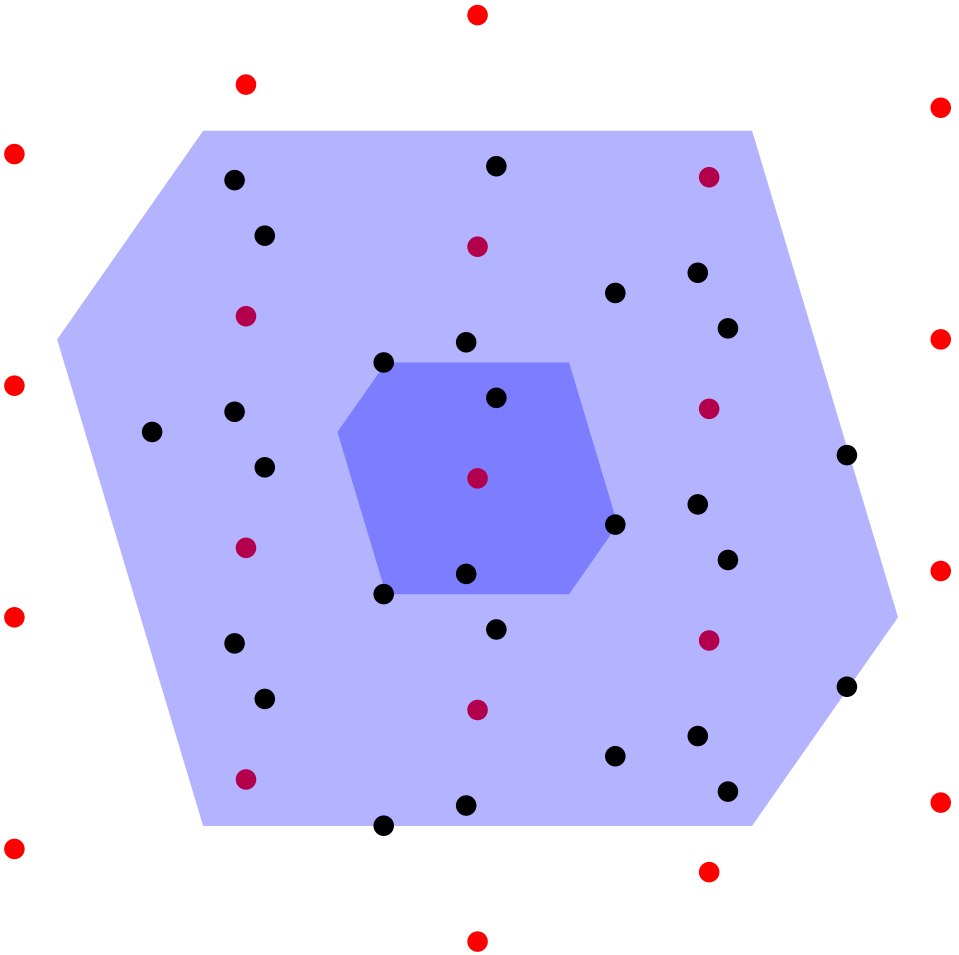
- $\mathrm{dom}(0, \Lambda)$: Voronoi domain of origin
- $\mathrm{dom}(0, 3\Lambda)$: scaled domain

Algorithm by Dolbilin & Huson, '97



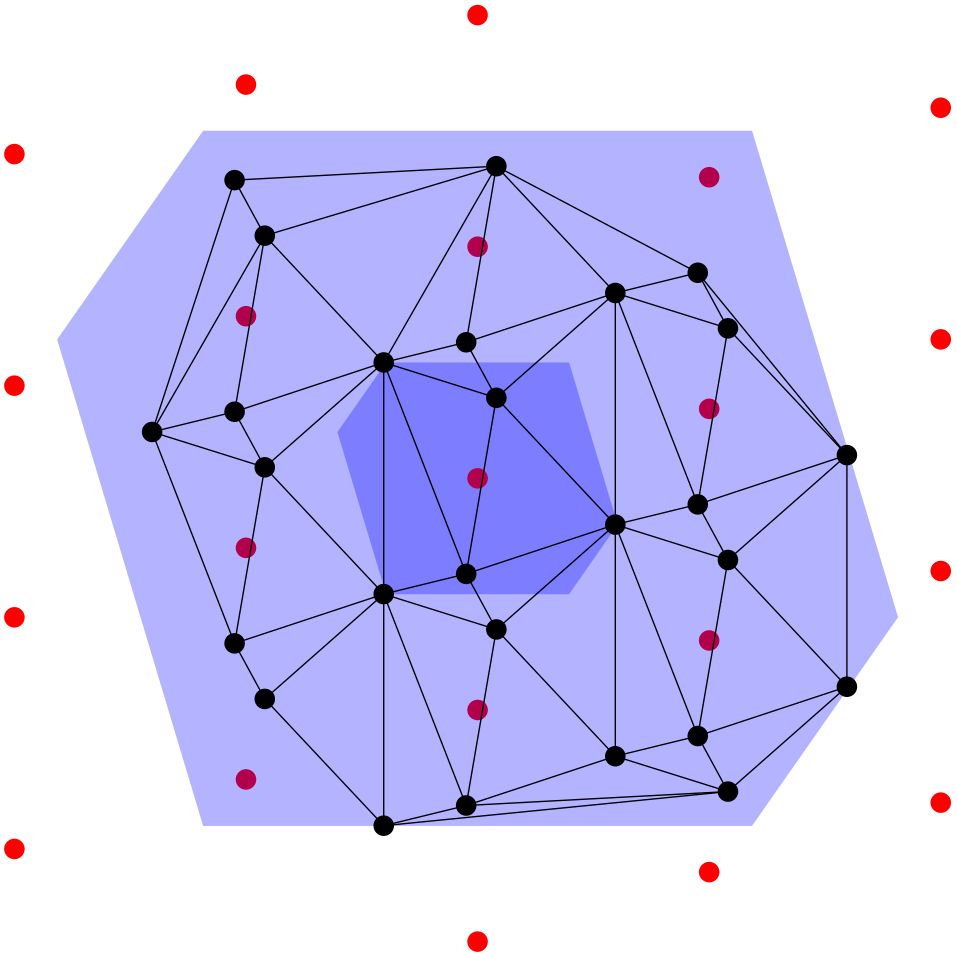
- $\mathrm{dom}(0, \Lambda)$: Voronoi domain of origin
- $\mathrm{dom}(0, 3\Lambda)$: scaled domain

Algorithm by Dolbilin & Huson, '97



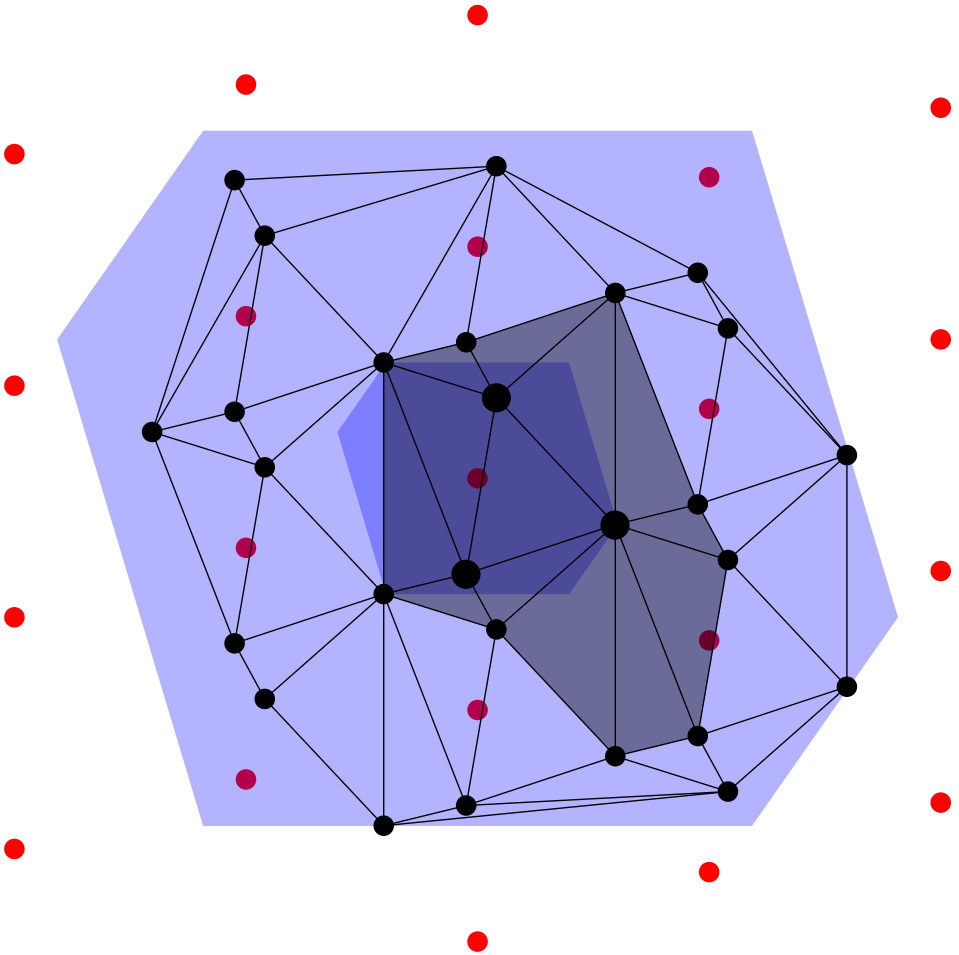
- $\mathrm{dom}(0, \Lambda)$: Voronoi domain of origin
- $\mathrm{dom}(0, 3\Lambda)$: scaled domain
- triangulate $\Lambda X \cap \mathrm{dom}(0, 3\Lambda)$

Algorithm by Dolbilin & Huson, '97



- $\mathrm{dom}(0, \Lambda)$: Voronoi domain of origin
- $\mathrm{dom}(0, 3\Lambda)$: scaled domain
- triangulate $\Lambda X \cap \mathrm{dom}(0, 3\Lambda)$

Algorithm by Dolbilin & Huson, '97



- $\mathrm{dom}(0, \Lambda)$: Voronoi domain of origin
- $\mathrm{dom}(0, 3\Lambda)$: scaled domain
- triangulate $\Lambda X \cap \mathrm{dom}(0, 3\Lambda)$
- Cells with a vertex in $\mathrm{dom}(0, \Lambda)$ are "good", i.e. part of periodic triangulation

Combined approach

Algorithm Summary:

- start with algorithm based on DH97 (“phase 1”)

Combined approach

Algorithm Summary:

- start with algorithm based on DH97 ("phase 1")
  - Euclidean triangulation of $\Lambda X \cap \mathrm{dom}(0, 3\Lambda)$
  - i.e., incrementally insert 3^d copies of each point

Combined approach

Algorithm Summary:

- start with algorithm based on DH97 ("phase 1")
  - Euclidean triangulation of $\Lambda X \cap \mathrm{dom}(0, 3\Lambda)$
  - i.e., incrementally insert 3^d copies of each point
  - provide interface for access to (implicit) periodic triangulation

Combined approach

Algorithm Summary:

- start with algorithm based on DH97 ("phase 1")
  - Euclidean triangulation of $\Lambda X \cap \mathrm{dom}(0, 3\Lambda)$
  - i.e., incrementally insert 3^d copies of each point
  - provide interface for access to (implicit) periodic triangulation
- once aforementioned criterion fulfilled, operate akin to cubic case in CGAL ("phase 2")

Combined approach

Algorithm Summary:

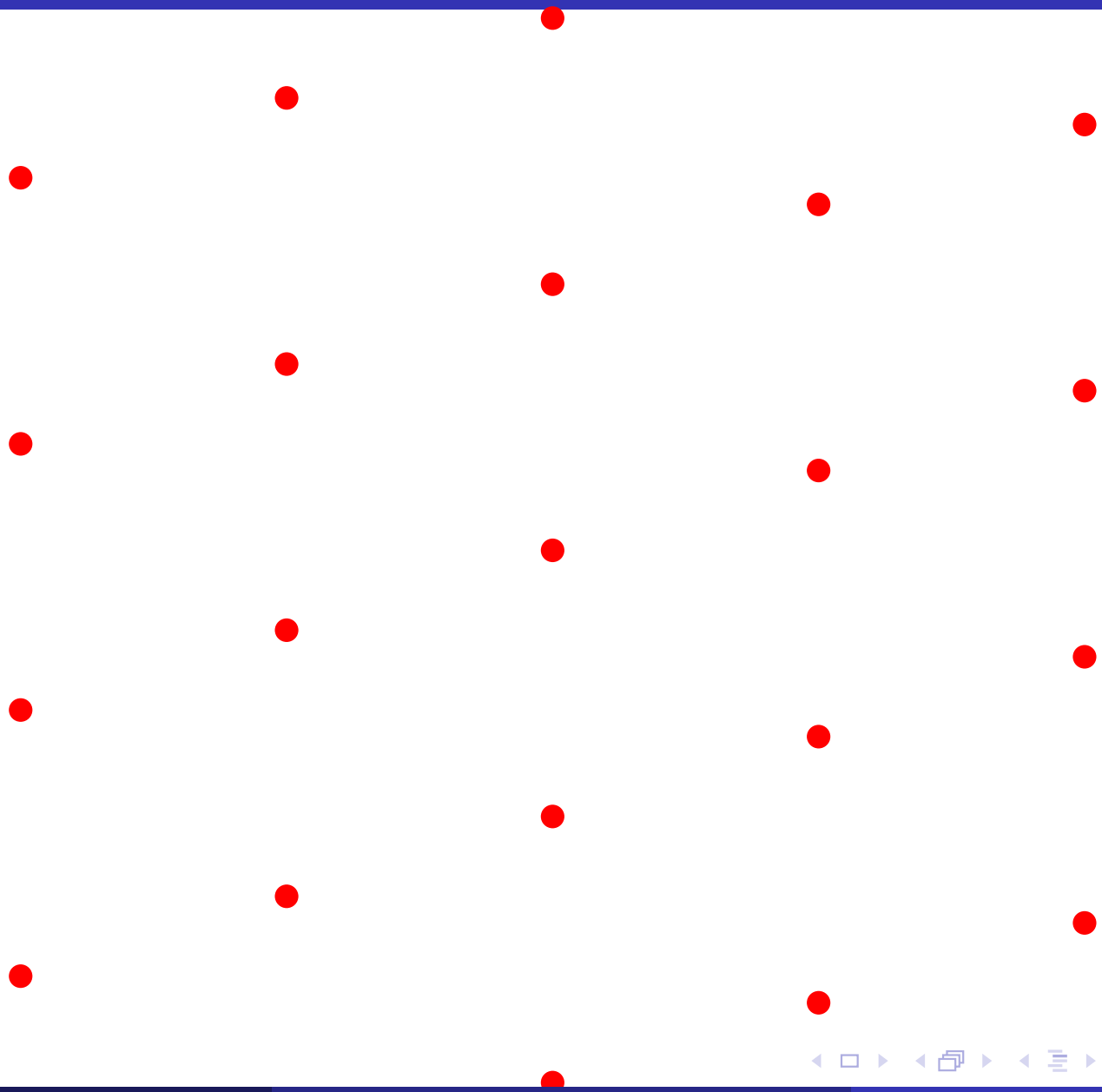
- start with algorithm based on DH97 ("phase 1")
  - Euclidean triangulation of $\Lambda X \cap \mathrm{dom}(0, 3\Lambda)$
  - i.e., incrementally insert 3^d copies of each point
  - provide interface for access to (implicit) periodic triangulation
- once aforementioned criterion fulfilled, operate akin to cubic case in CGAL ("phase 2")
  - periodic triangulation data structure

Combined approach

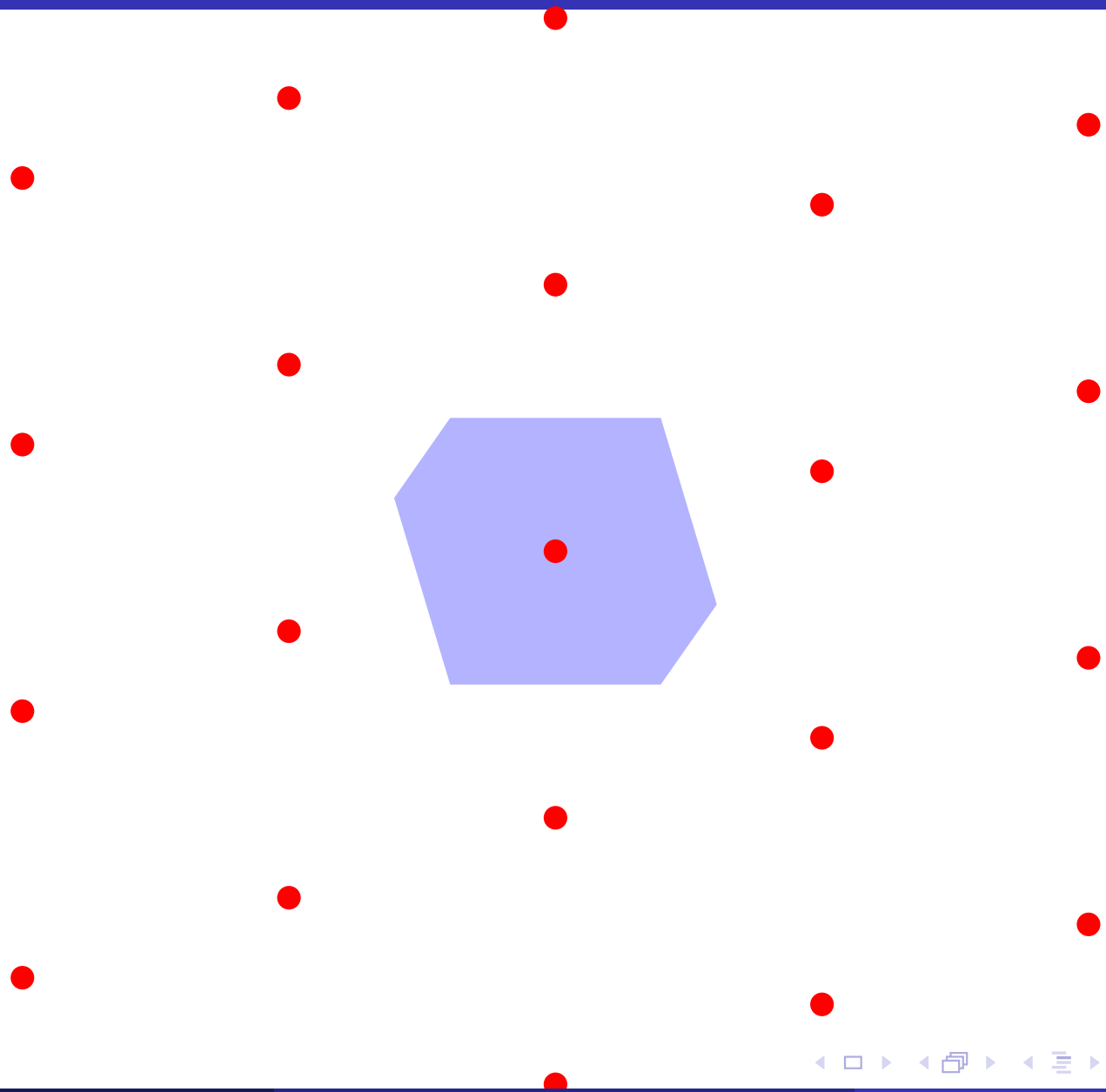
Algorithm Summary:

- start with algorithm based on DH97 ("phase 1")
  - Euclidean triangulation of $\Lambda X \cap \mathrm{dom}(0, 3\Lambda)$
  - i.e., incrementally insert 3^d copies of each point
  - provide interface for access to (implicit) periodic triangulation
- once aforementioned criterion fulfilled, operate akin to cubic case in CGAL ("phase 2")
  - periodic triangulation data structure
  - 1 copy per point

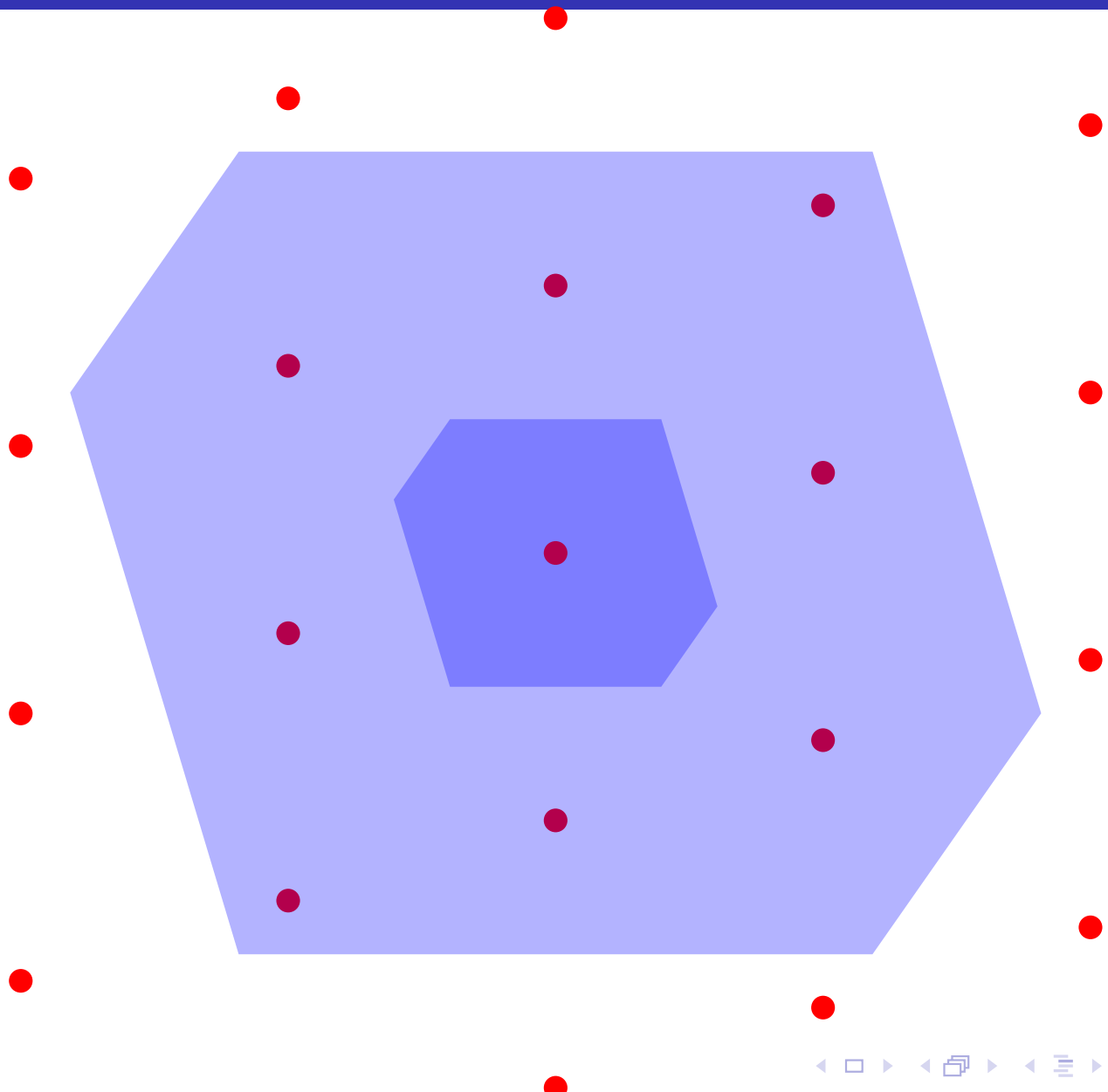
Combined approach



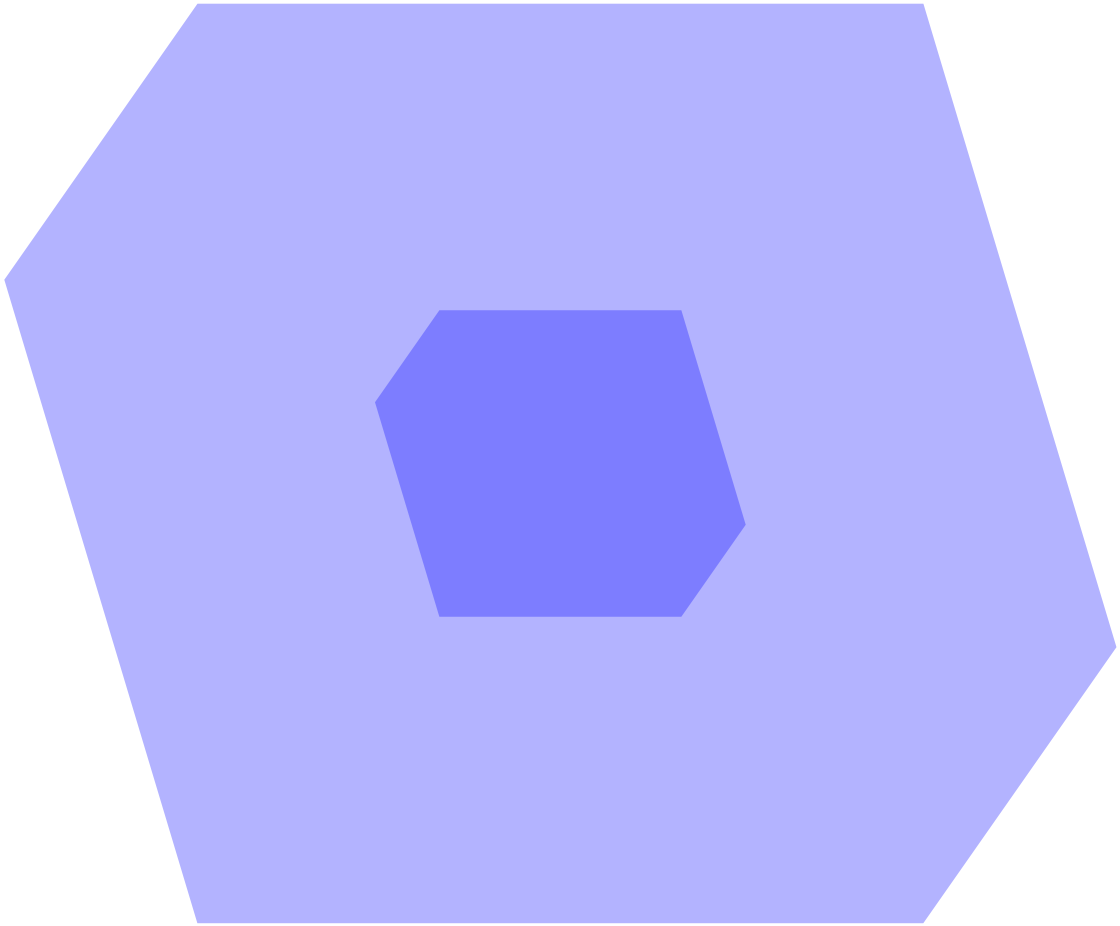
Combined approach



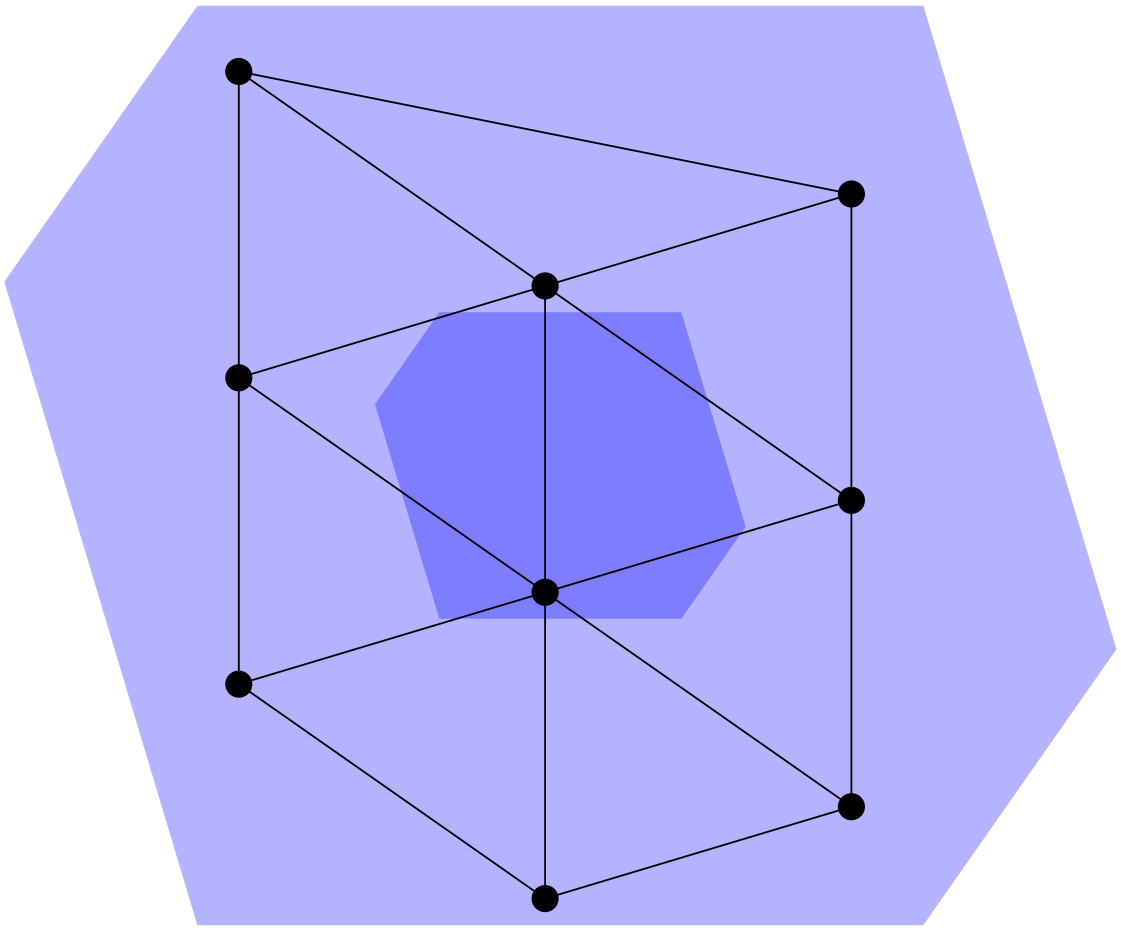
Combined approach



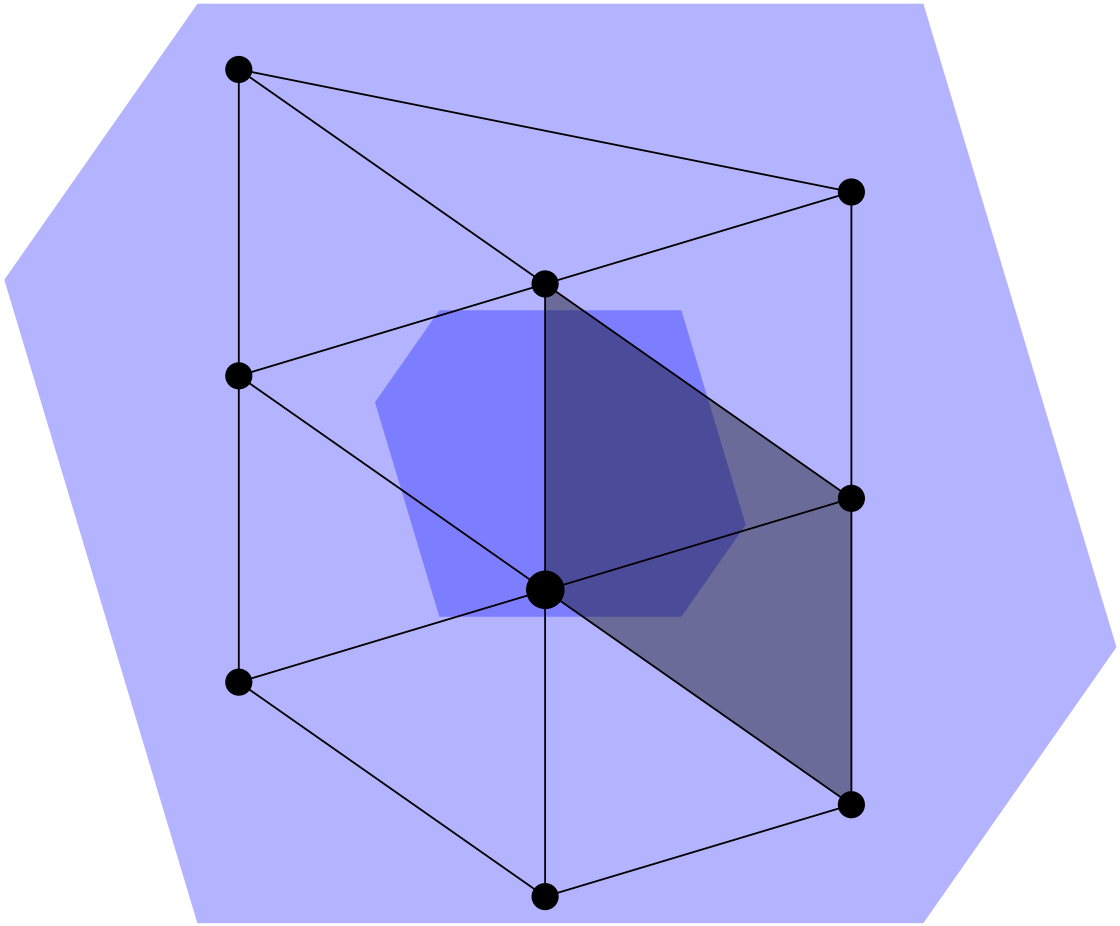
Combined approach



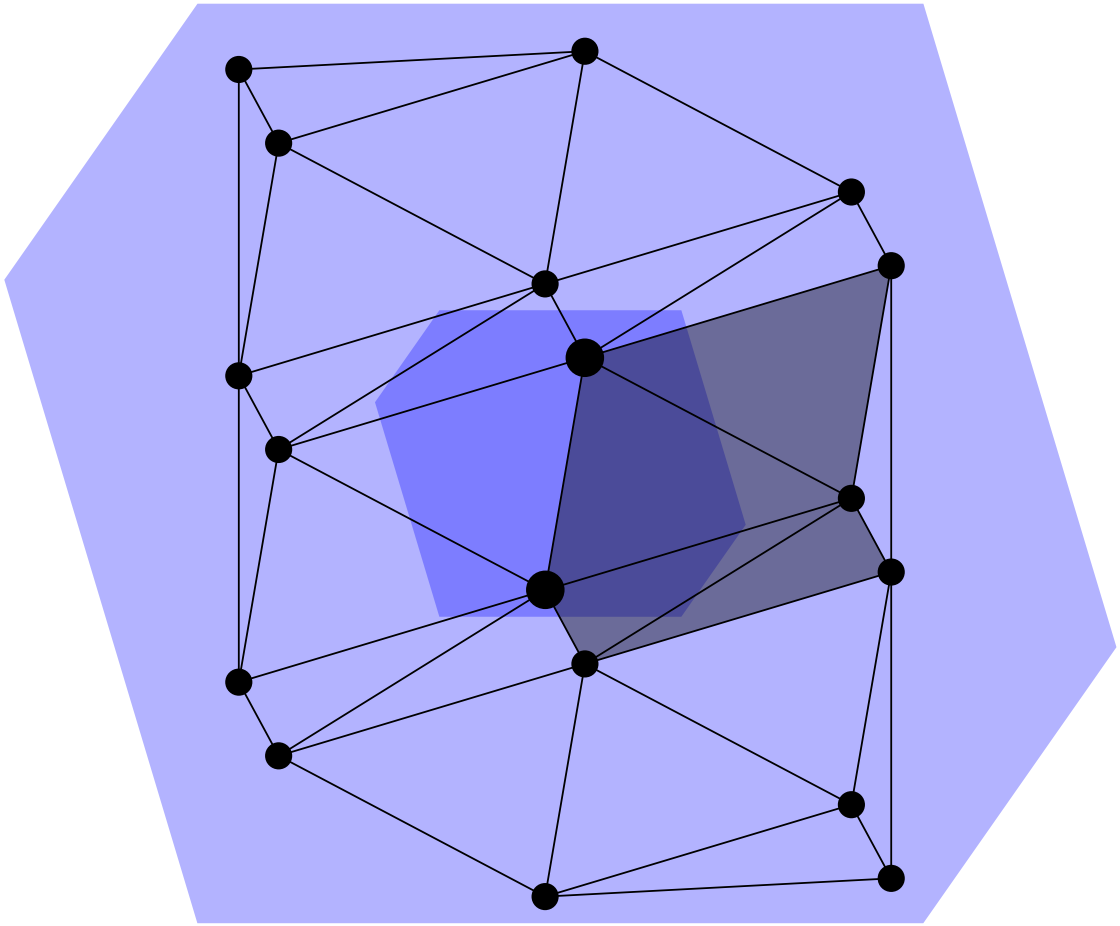
Combined approach



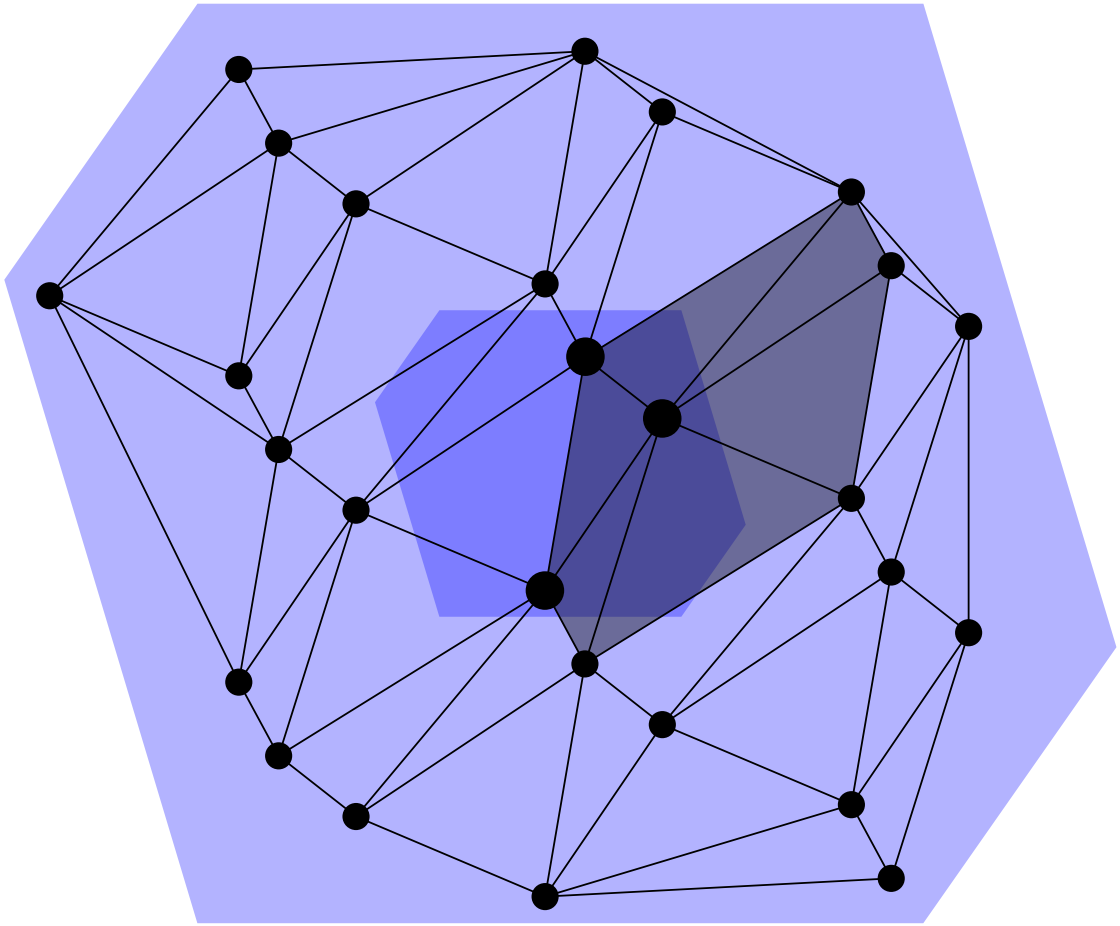
Combined approach



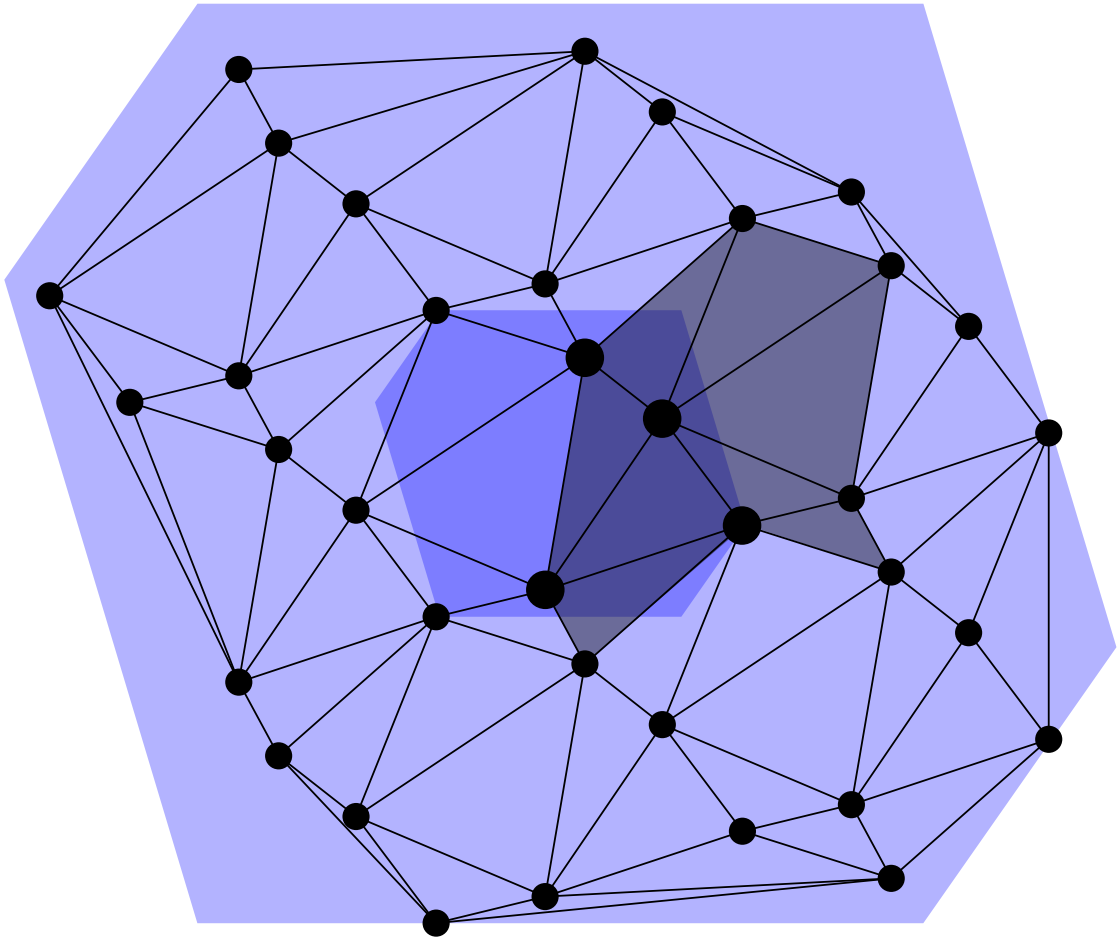
Combined approach



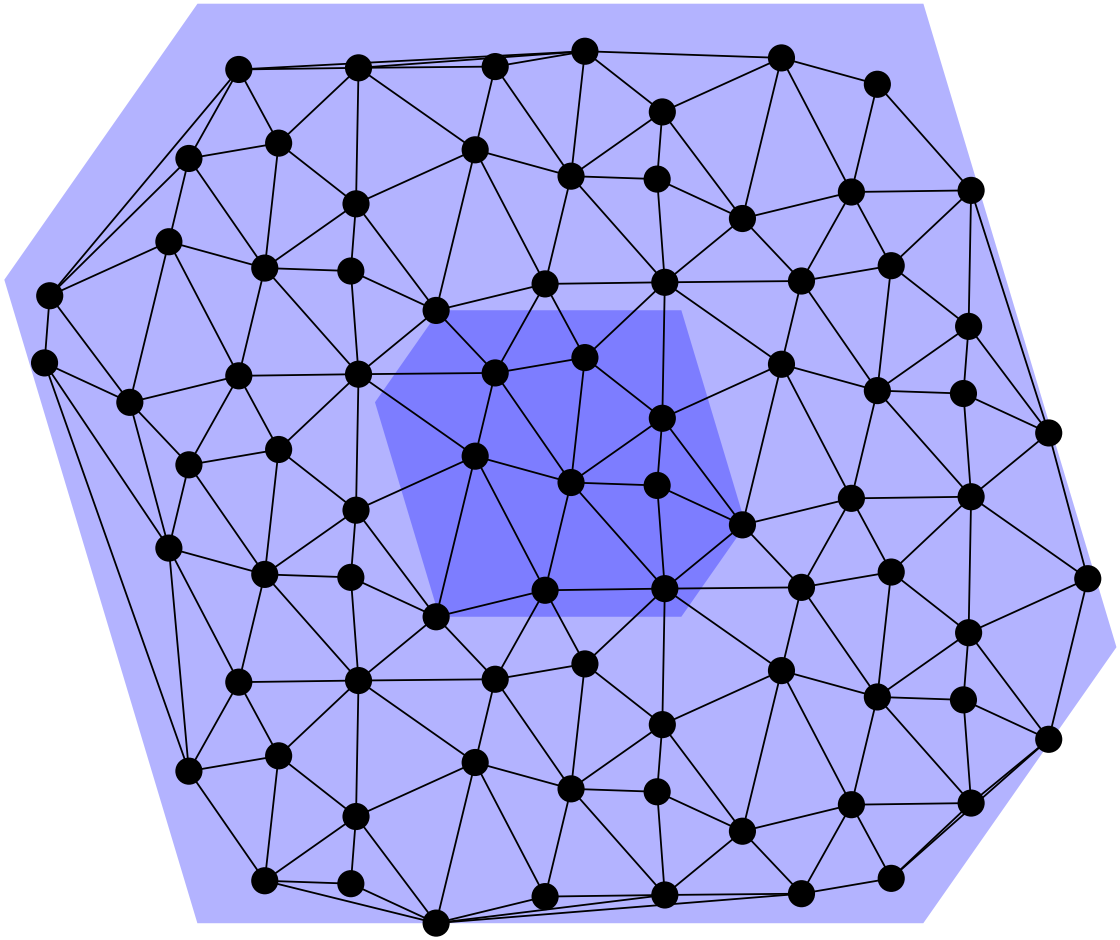
Combined approach



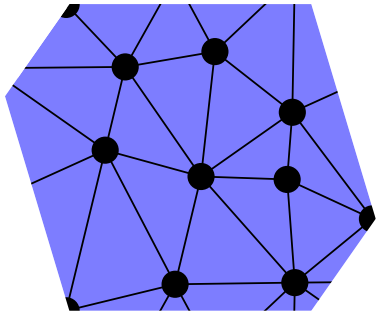
Combined approach



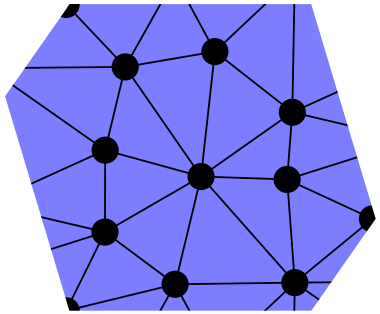
Combined approach



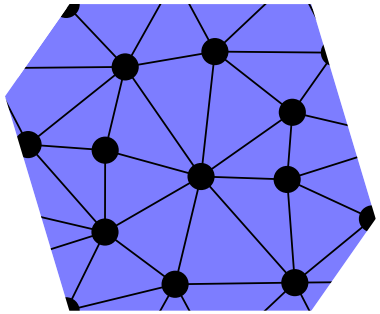
Combined approach



Combined approach



Combined approach



Voronoi domain

Computing Voronoi domain $\mathrm{dom}(0, \Lambda)$:

Voronoi domain

Computing Voronoi domain $\mathrm{dom}(0, \Lambda)$:

- *reduce* lattice basis

Voronoi domain

Computing Voronoi domain $\mathrm{dom}(0, \Lambda)$:

- *reduce* lattice basis
  - obtain faces of Voronoi domain from reduced basis vectors

Voronoi domain

Computing Voronoi domain $\mathrm{dom}(0, \Lambda)$:

- *reduce* lattice basis
  - obtain faces of Voronoi domain from reduced basis vectors
  - Remark: not as straightforward in dimension ≥ 4

Voronoi domain

Computing Voronoi domain $\mathrm{dom}(0, \Lambda)$:

- *reduce* lattice basis
  - obtain faces of Voronoi domain from reduced basis vectors
  - Remark: not as straightforward in dimension ≥ 4

Computing the *canonical* point copy in $\mathrm{dom}(0, \Lambda)$:

Voronoi domain

Computing Voronoi domain $\mathrm{dom}(0, \Lambda)$:

- *reduce* lattice basis
  - obtain faces of Voronoi domain from reduced basis vectors
  - Remark: not as straightforward in dimension ≥ 4

Computing the *canonical* point copy in $\mathrm{dom}(0, \Lambda)$:

- equivalent to closest vector problem (CVP)

Voronoi domain

Computing Voronoi domain $\mathrm{dom}(0, \Lambda)$:

- *reduce* lattice basis
  - obtain faces of Voronoi domain from reduced basis vectors
  - Remark: not as straightforward in dimension ≥ 4

Computing the *canonical* point copy in $\mathrm{dom}(0, \Lambda)$:

- equivalent to closest vector problem (CVP)
- use existing algorithm, e.g. Sommer, Feder, Shalvi '09

Voronoi domain

Computing Voronoi domain $\mathrm{dom}(0, \Lambda)$:

- *reduce* lattice basis
  - obtain faces of Voronoi domain from reduced basis vectors
  - Remark: not as straightforward in dimension ≥ 4

Computing the *canonical* point copy in $\mathrm{dom}(0, \Lambda)$:

- equivalent to closest vector problem (CVP)
- use existing algorithm, e.g. Sommer, Feder, Shalvi '09

Computing all point copies in scaled domain $\mathrm{dom}(0, 3\Lambda)$:

Voronoi domain

Computing Voronoi domain $\mathrm{dom}(0, \Lambda)$:

- *reduce* lattice basis
  - obtain faces of Voronoi domain from reduced basis vectors
  - Remark: not as straightforward in dimension ≥ 4

Computing the *canonical* point copy in $\mathrm{dom}(0, \Lambda)$:

- equivalent to closest vector problem (CVP)
- use existing algorithm, e.g. Sommer, Feder, Shalvi '09

Computing all point copies in scaled domain $\mathrm{dom}(0, 3\Lambda)$:

- translate point by fixed set of integer combinations of basis vectors

Voronoi domain

Computing Voronoi domain $\mathrm{dom}(0, \Lambda)$:

- *reduce* lattice basis
  - obtain faces of Voronoi domain from reduced basis vectors
  - Remark: not as straightforward in dimension ≥ 4

Computing the *canonical* point copy in $\mathrm{dom}(0, \Lambda)$:

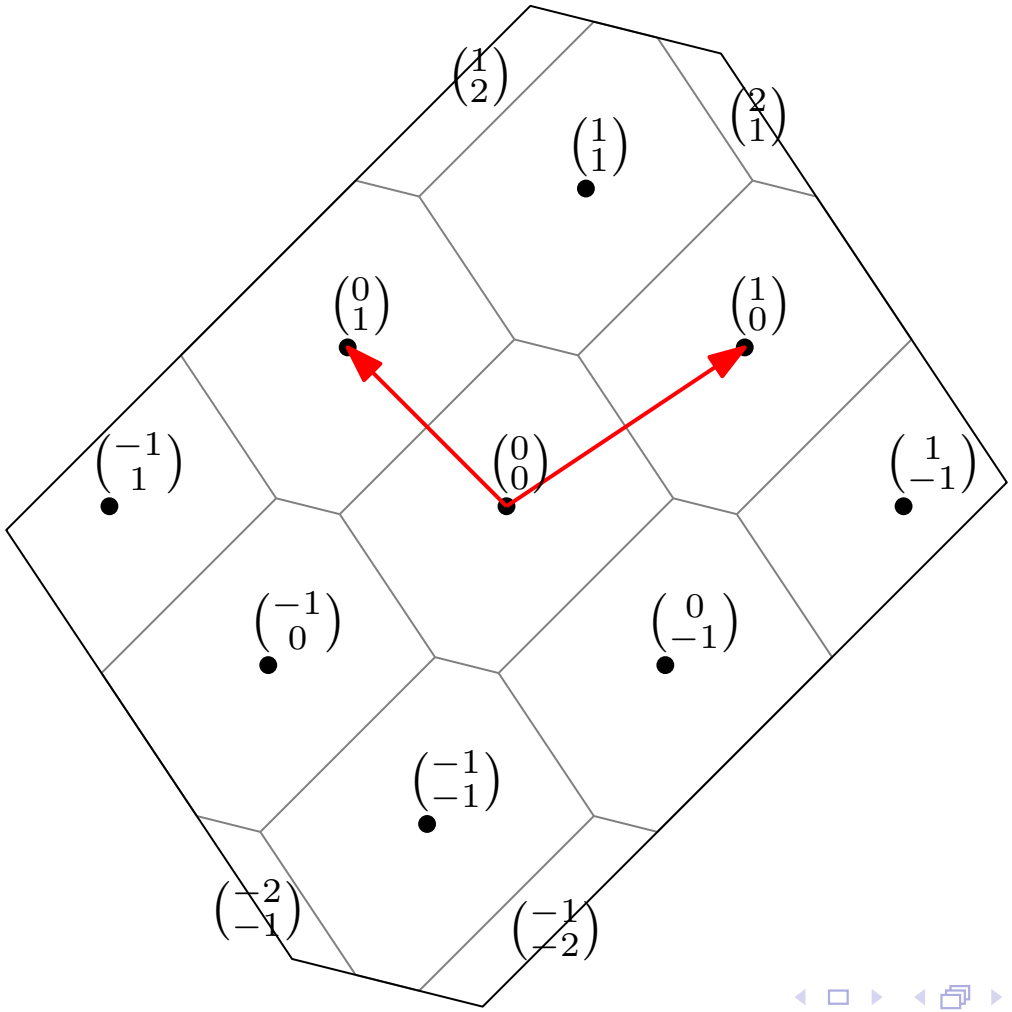
- equivalent to closest vector problem (CVP)
- use existing algorithm, e.g. Sommer, Feder, Shalvi '09

Computing all point copies in scaled domain $\mathrm{dom}(0, 3\Lambda)$:

- translate point by fixed set of integer combinations of basis vectors
- check translated point for containment in $\mathrm{dom}(0, 3\Lambda)$

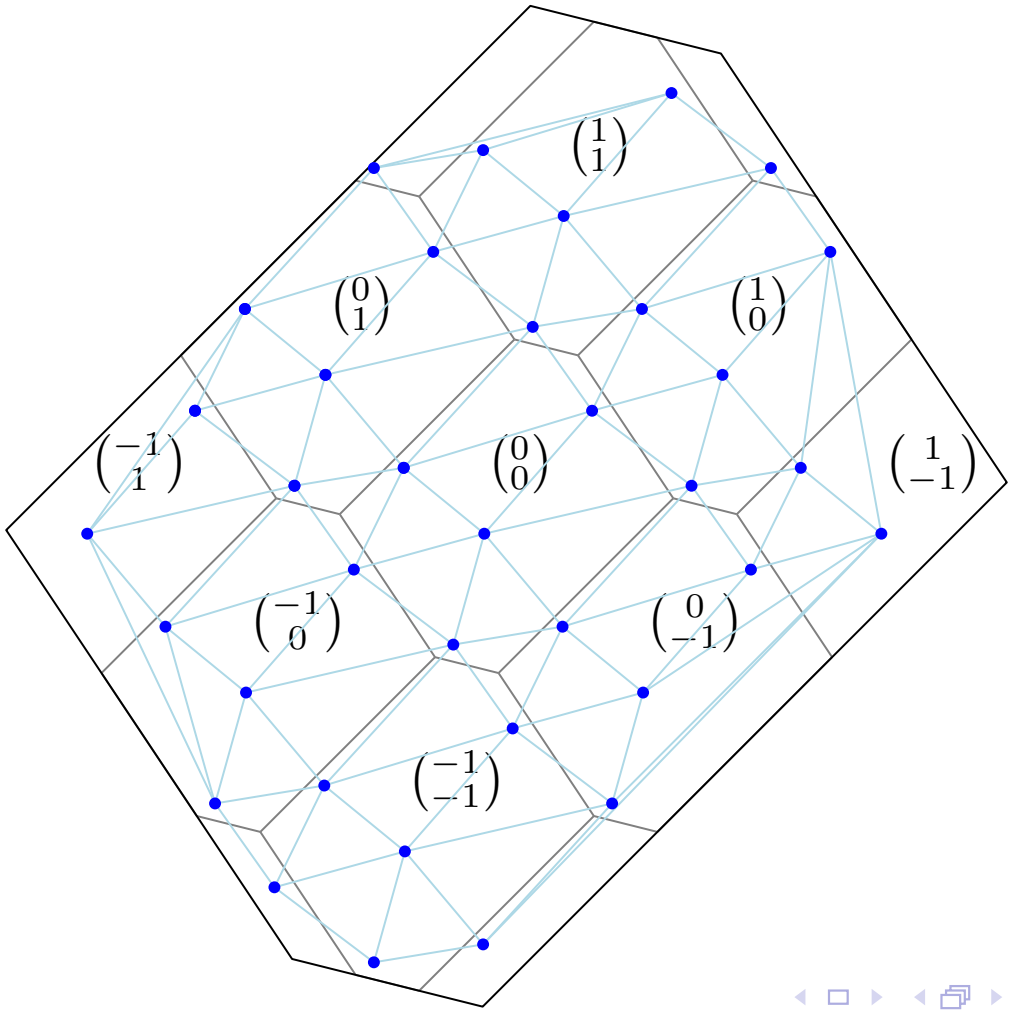
Interface in phase 1

- Output: associate *offsets* to vertices of a cell



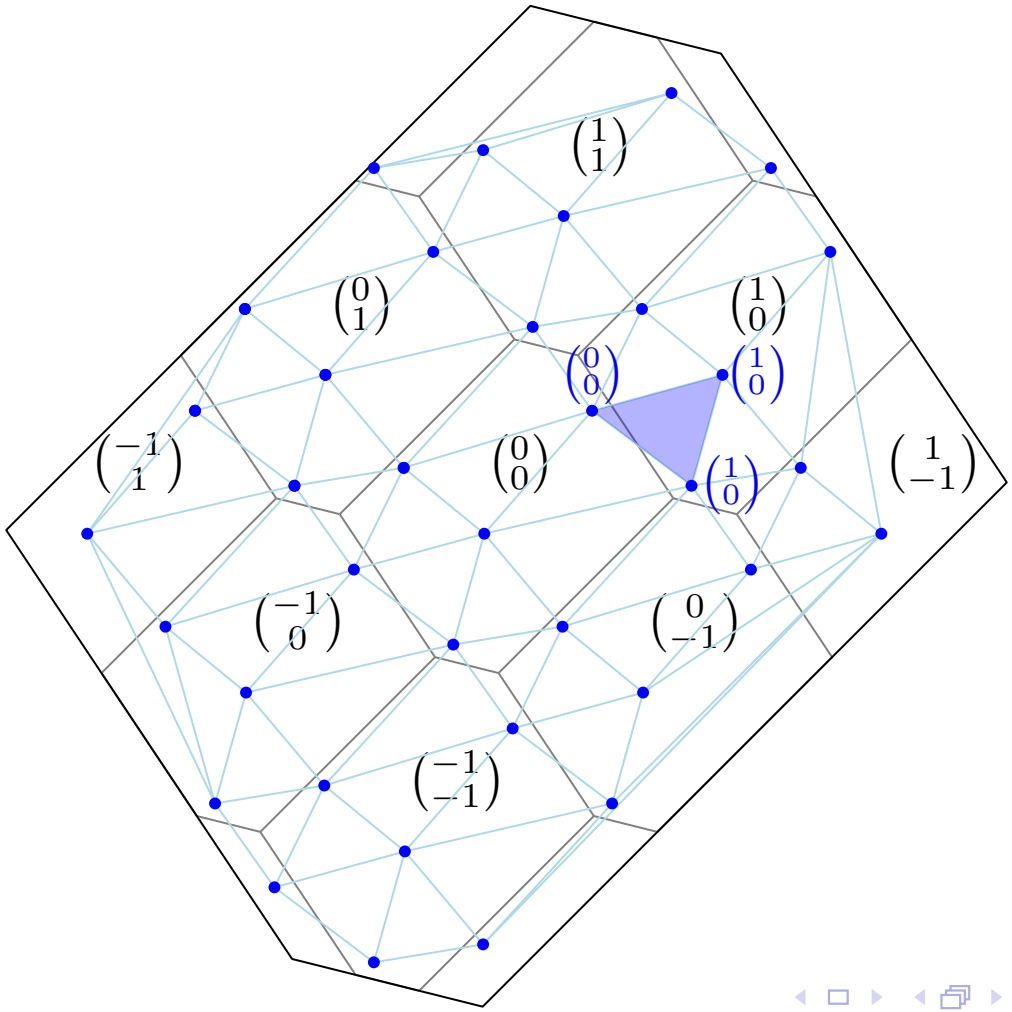
Interface in phase 1

- Output: associate *offsets* to vertices of a cell



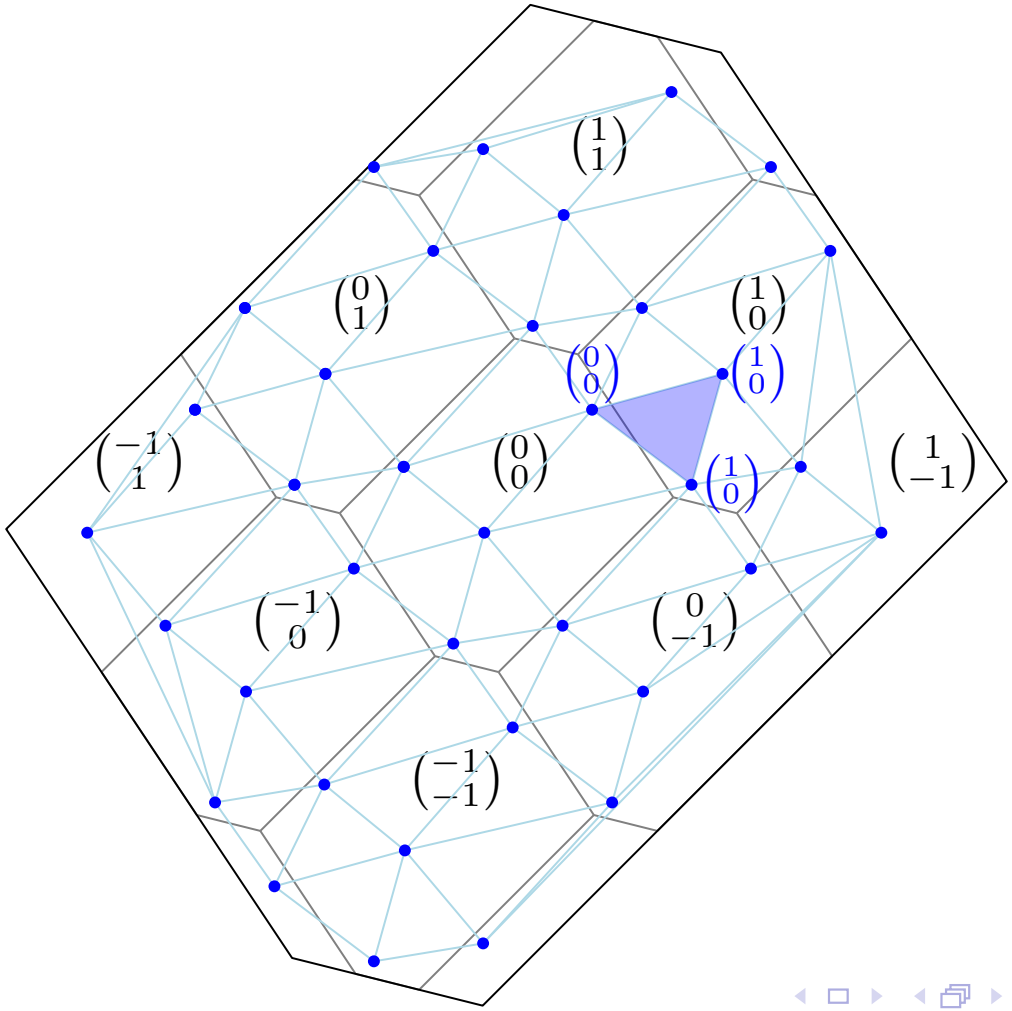
Interface in phase 1

- Output: associate *offsets* to vertices of a cell



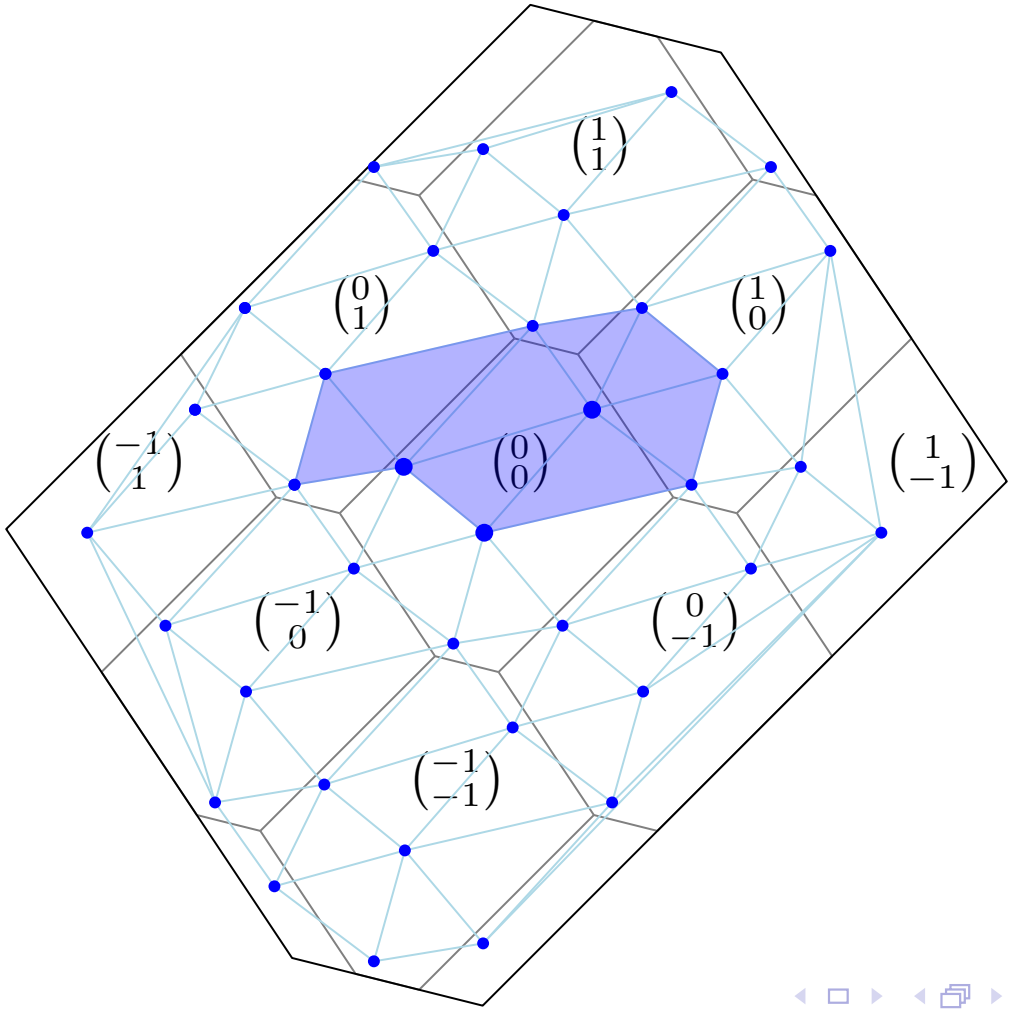
Interface in phase 1

- Output: associate *offsets* to vertices of a cell
- Filter iterators for *(canonical)* cells, vertices, . . .



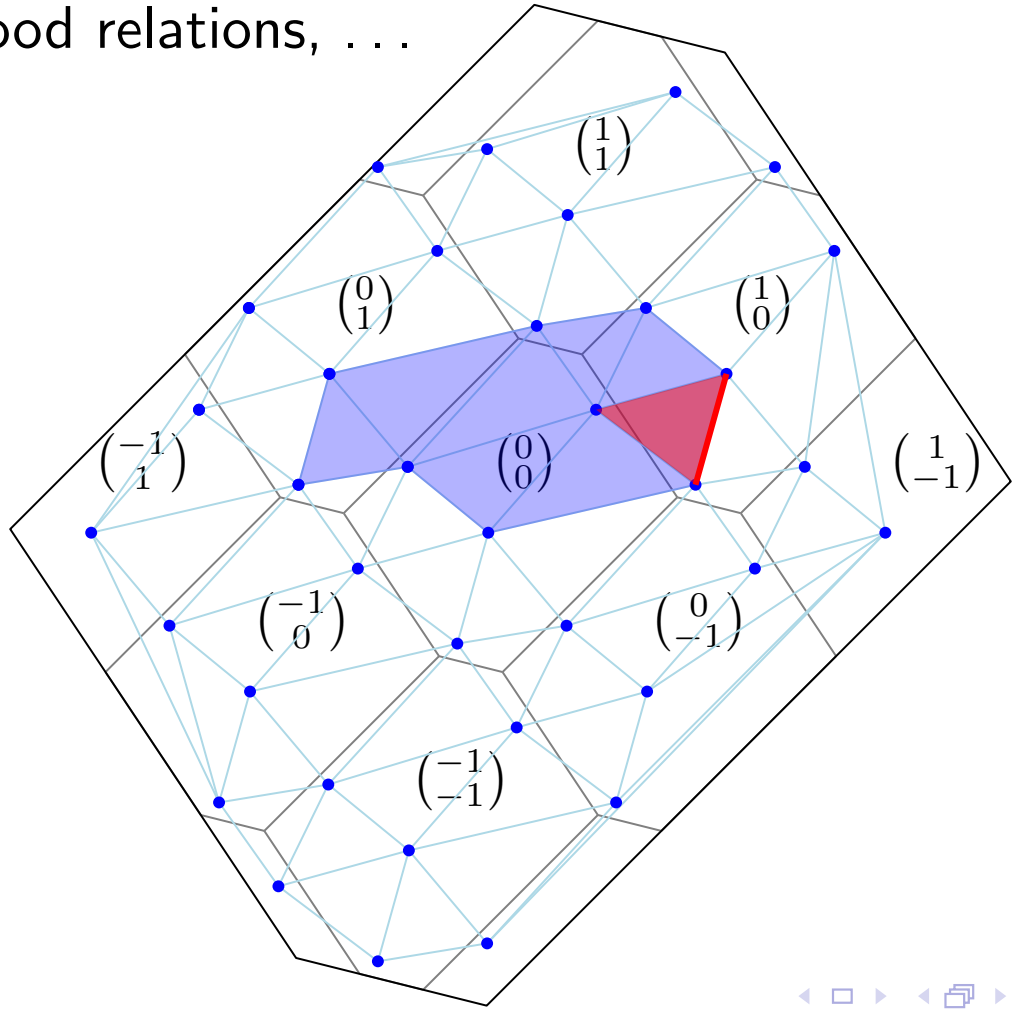
Interface in phase 1

- Output: associate *offsets* to vertices of a cell
- Filter iterators for *(canonical)* cells, vertices, . . .



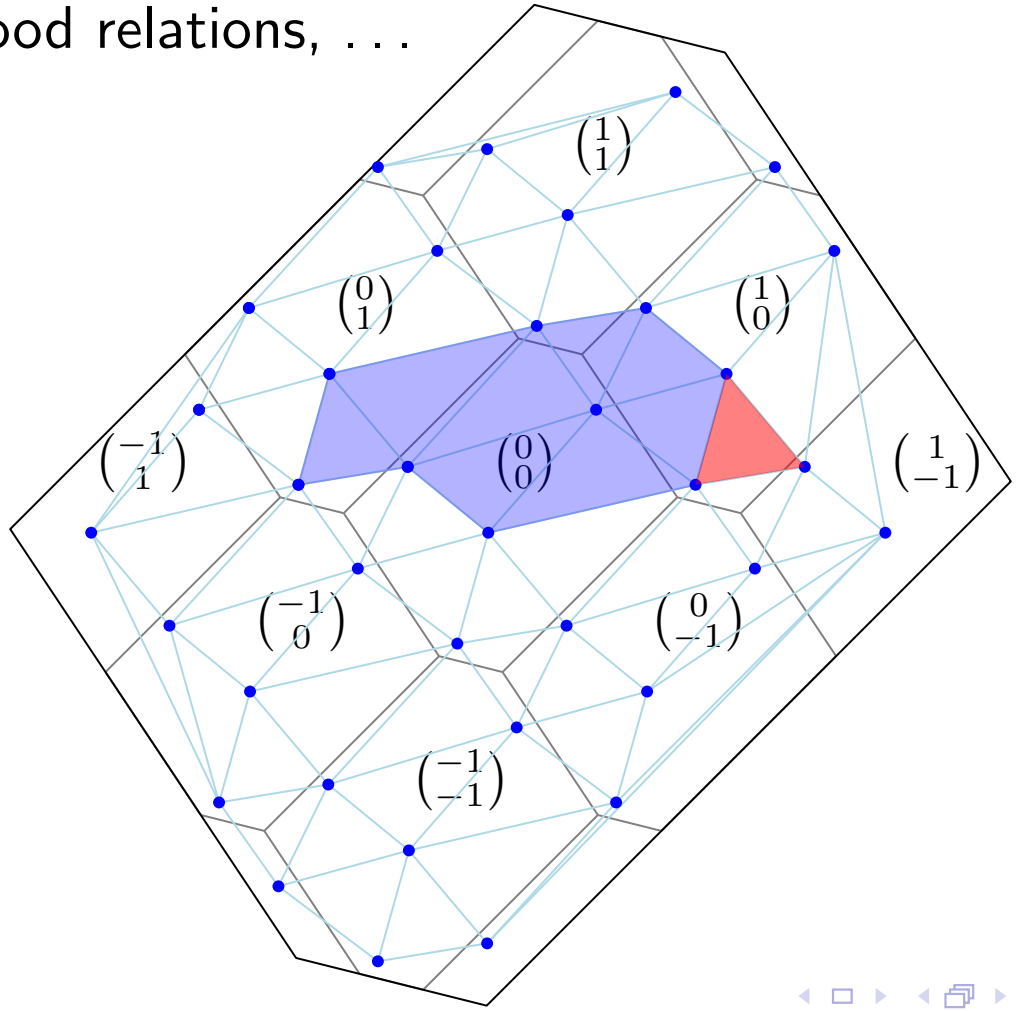
Interface in phase 1

- Output: associate *offsets* to vertices of a cell
- Filter iterators for *(canonical)* cells, vertices, ...
- Neighbourhood relations, ...



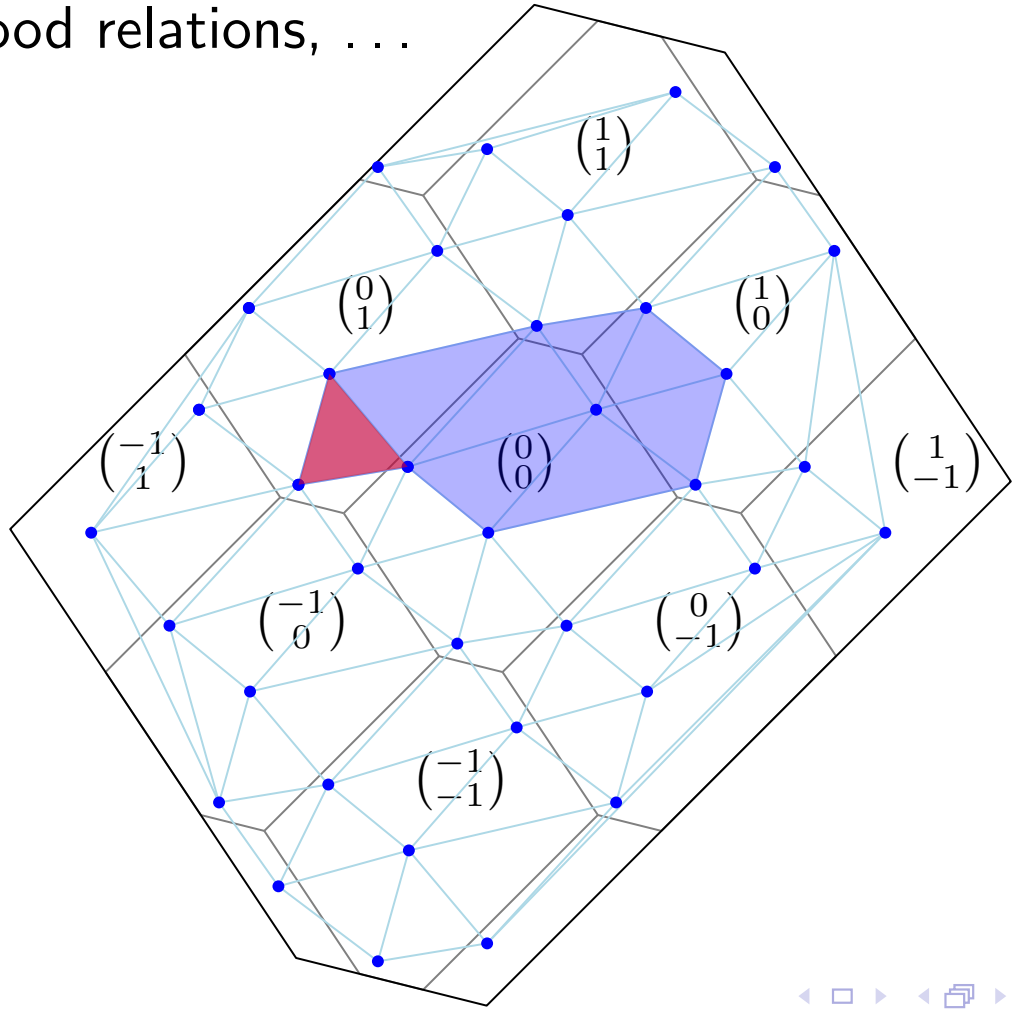
Interface in phase 1

- Output: associate *offsets* to vertices of a cell
- Filter iterators for *(canonical)* cells, vertices, ...
- Neighbourhood relations, ...



Interface in phase 1

- Output: associate *offsets* to vertices of a cell
- Filter iterators for *(canonical)* cells, vertices, ...
- Neighbourhood relations, ...



Converting to phase 2

Transition criterion:

Converting to phase 2

Transition criterion:

- circumradii of all cells are smaller than $\frac{1}{4}\mathrm{sv}(\Lambda)$
- $\mathrm{sv}(\Lambda)$: length of shortest (non-zero) lattice vector

Converting to phase 2

Transition criterion:

- circumradii of all cells are smaller than $\frac{1}{4}\mathrm{sv}(\Lambda)$
- $\mathrm{sv}(\Lambda)$: length of shortest (non-zero) lattice vector
- ensures conflict zones of all future insertions contractible

Converting to phase 2

Transition criterion:

- circumradii of all cells are smaller than $\frac{1}{4}\mathrm{sv}(\Lambda)$
- $\mathrm{sv}(\Lambda)$: length of shortest (non-zero) lattice vector
- ensures conflict zones of all future insertions contractible

Number of points until transition:

Converting to phase 2

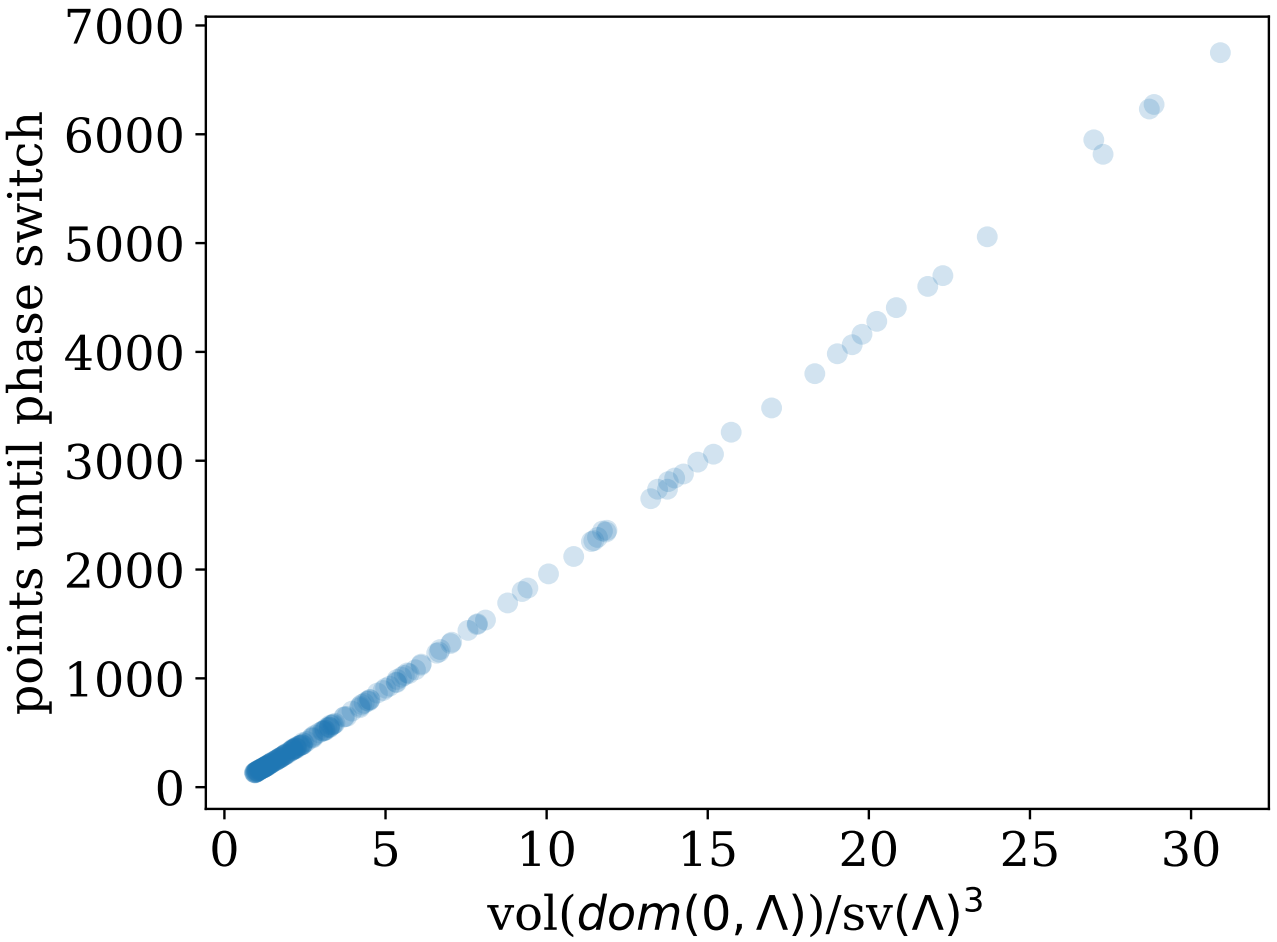
Transition criterion:

- circumradii of all cells are smaller than $\frac{1}{4}\mathrm{sv}(\Lambda)$
- $\mathrm{sv}(\Lambda)$: length of shortest (non-zero) lattice vector
- ensures conflict zones of all future insertions contractible

Number of points until transition:

- depends on shape of Voronoi domain
- in practice: earlier transition if points evenly distributed

Points until switch



Number of random points inserted until switching criterion is fulfilled, for various 3D lattices. Each point in the plot represents a lattice.

Running times (cubic case)

Implementation	CGAL Euclidean	CGAL Cubic	Lattice (new)
10^0	0.0000	0.0001	0.0002
10^1	0.0000	0.0160	0.0033
10^2	0.0004	0.1848	0.0461
10^3	0.0049	0.5957	0.0858
10^4	0.0487	0.9591	0.3832
10^5	0.5679	4.8119	4.5153
10^6	6.5974	93.9327	95.1447
10^7	59.5152	2314.3618	2317.7867

Running time (in seconds) of various algorithms on random point sets of various sizes with cubic periodicity.

Running times (other lattices)

Lattice	Cubic	FCC	Λ_1	Λ_2
$\frac{\mathrm{vol}(\mathrm{dom}(0,\Lambda))}{\mathrm{sv}(\Lambda)^3}$	1.0	0.71	12.5	346.41
n_{switch}	141	94	2519	89950
10^0	0.0002	0.0001	0.0003	0.0004
10^1	0.0033	0.0026	0.0044	0.0035
10^2	0.0461	0.0287	0.0460	0.0380
10^3	0.0858	0.0446	0.9812	0.6372
10^4	0.3832	0.1642	4.6602	16.5759
10^5	4.5153	2.7868	10.8139	362.7956
10^6	95.1447	51.5945	58.1568	517.9715
10^7	2317.7867	1215.2648	1799.4515	2983.0943

Running times (in seconds) of our algorithm for random point sets of various sizes and various lattices.

n_{switch}: average number of points until switch to phase 2

Summary

- periodic 2D and 3D Delaunay triangulations for arbitrary lattices within branch of CGAL codebase, aim to integrate into CGAL
- performance comparable to CGAL implementation for cubic lattice
- internally operates in 2 phases
  - Phase 1: Euclidean triangulation with 3^d copies of each point
  - Phase 2: Periodic triangulation with 1 copy of each point
  - Transition once sufficiently many points inserted
- provides uniform interface to periodic triangulation (e.g. iterators, adjacency relations) that hides the internal phase