ARRANGEMENTS

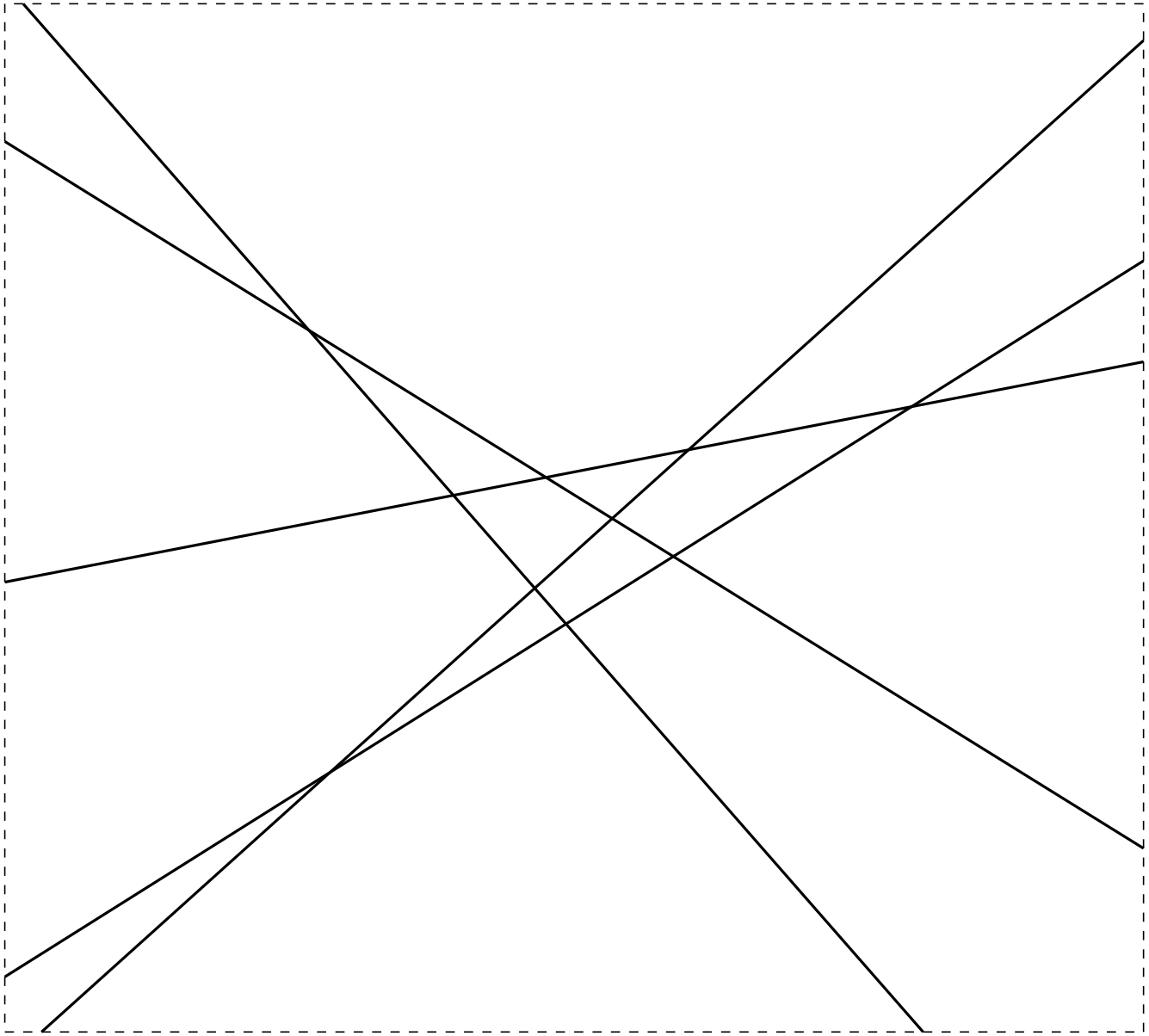
Vera Sacristán

Computational Geometry
Facultat d'Informtica de Barcelona
Universitat Politcnica de Catalunya

ARRANGEMENTS

DEFINITION

Let $L = \{l_1, \ldots, l_n\}$ be a finite set of lines in the plane.

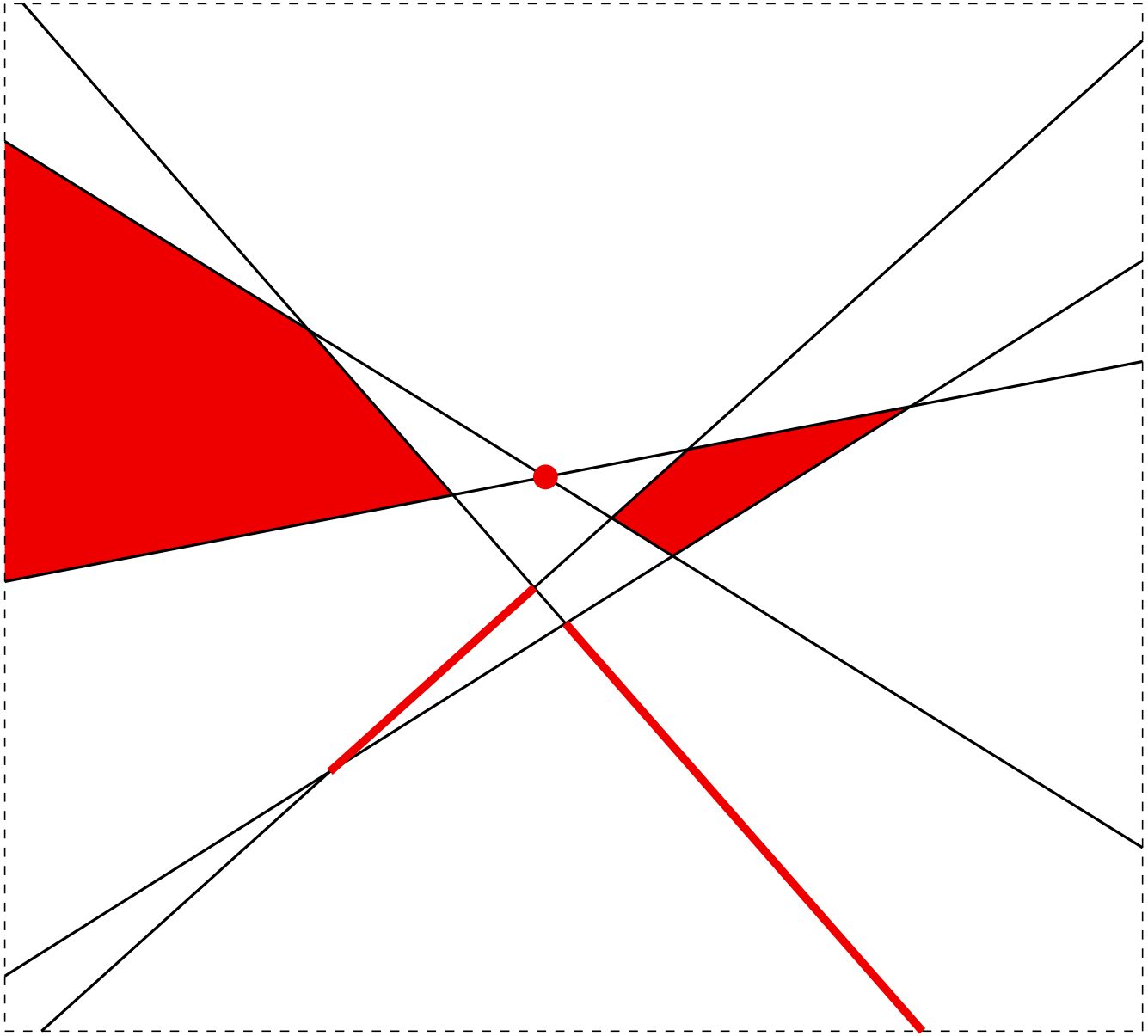


ARRANGEMENTS

DEFINITION

Let $L = \{l_1, \ldots, l_n\}$ be a finite set of lines in the plane.

The *arrangement* $\mathcal{A}(L)$ is the decomposition of the plane into *faces*, *edges* and *vertices* produced by L.



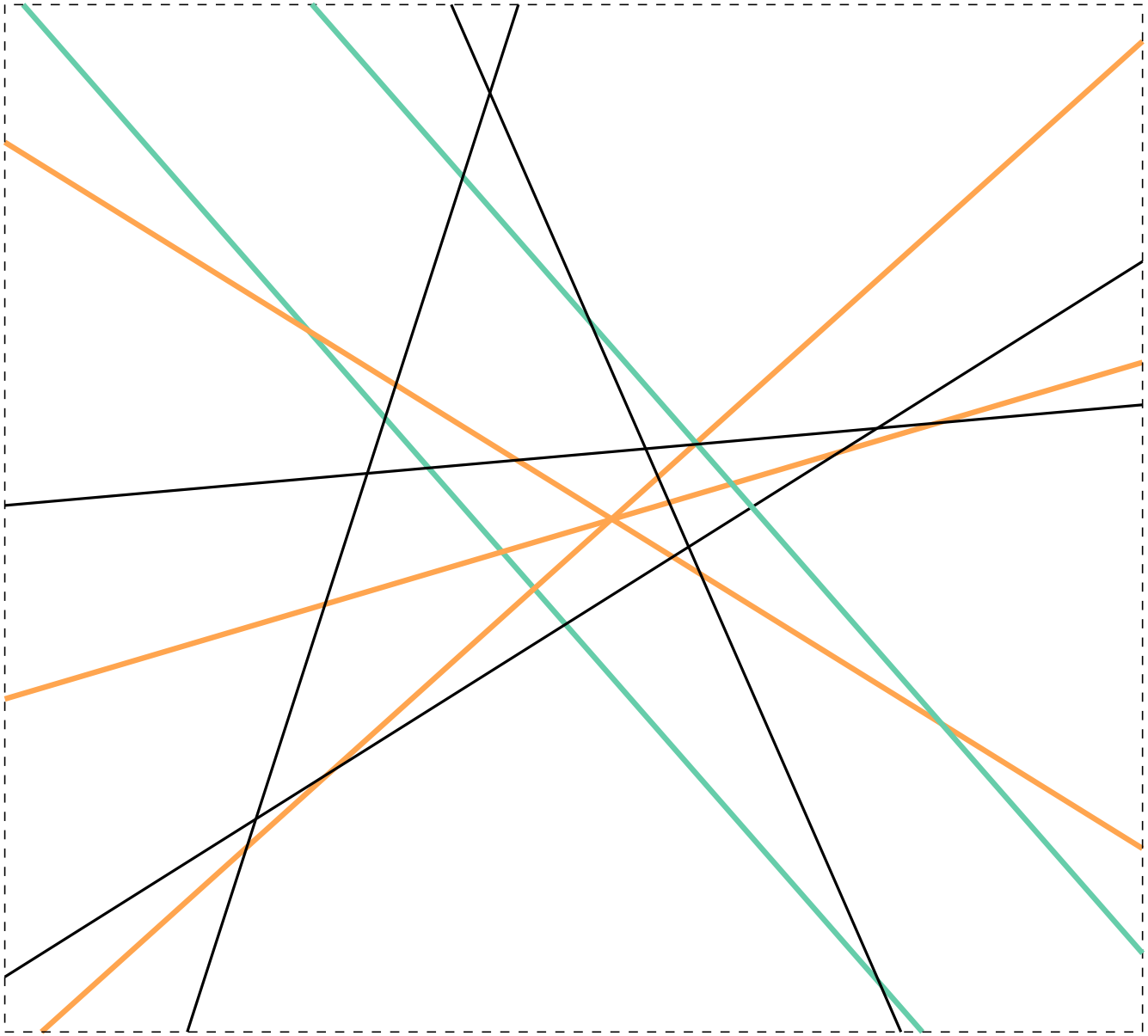
ARRANGEMENTS

DEFINITION

Let $L = \{l_1, \ldots, l_n\}$ be a finite set of lines in the plane.

The *arrangement* $\mathcal{A}(L)$ is the decomposition of the plane into *faces*, *edges* and *vertices* produced by L.

The arrangement $\mathcal{A}(L)$ is said to be *simple* if L does not contain any two parallel lines nor any three lines through the same point.



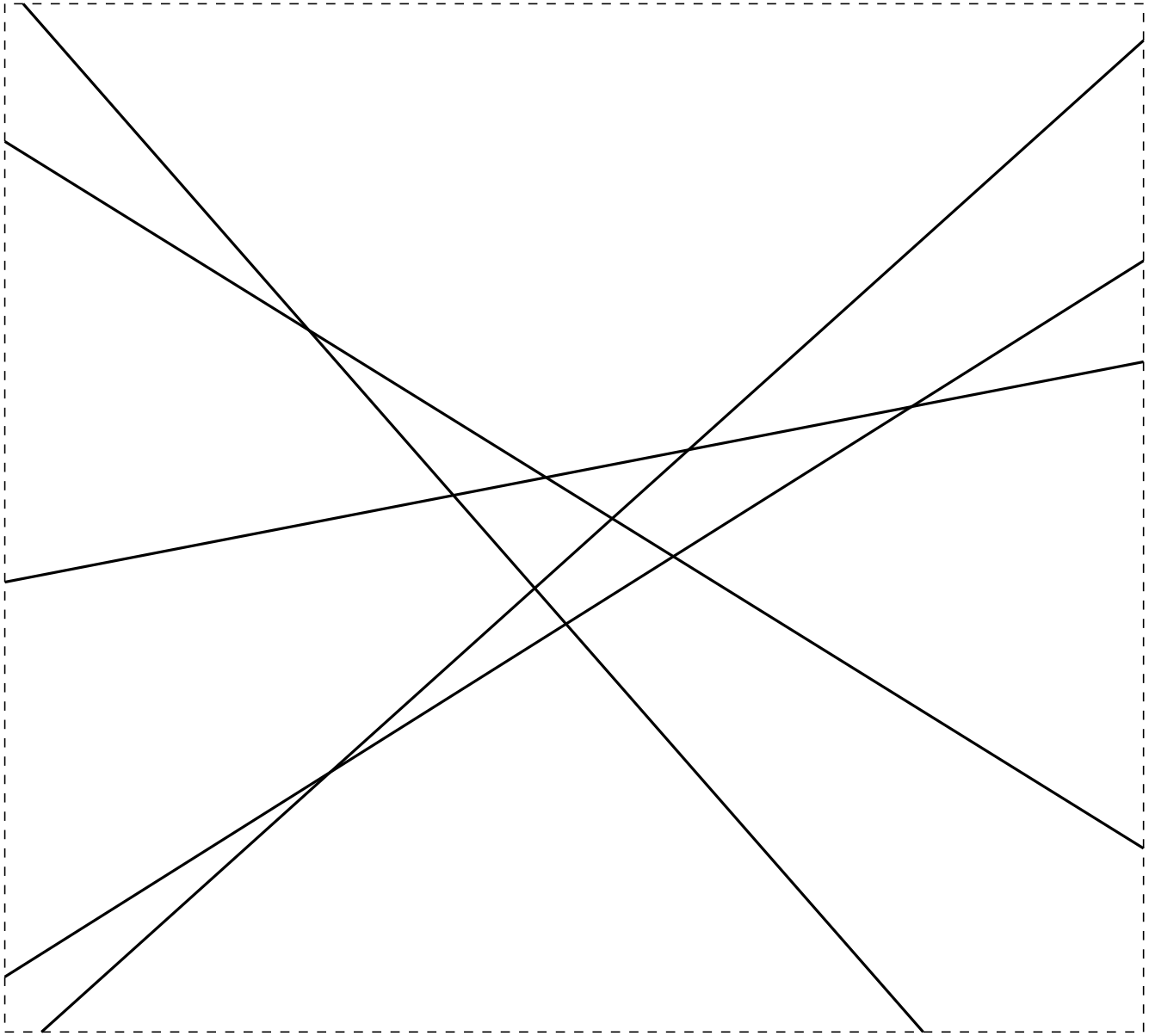
COMPLEXITY

The *(combinatorial) complexity* of an arrangement $\mathcal{A}(L)$ is its number of faces, edges and vertices.

The complexity of $\mathcal{A}(L)$ is $O(n^2)$, where $n = \#L$. Specifically:

- $v \leq \frac{n(n-1)}{2}$
- $e \leq n^2$
- $f \leq \frac{n^2}{2} + \frac{n}{2} + 1$

These bounds are achieved when the arrangement is simple.



COMPLEXITY

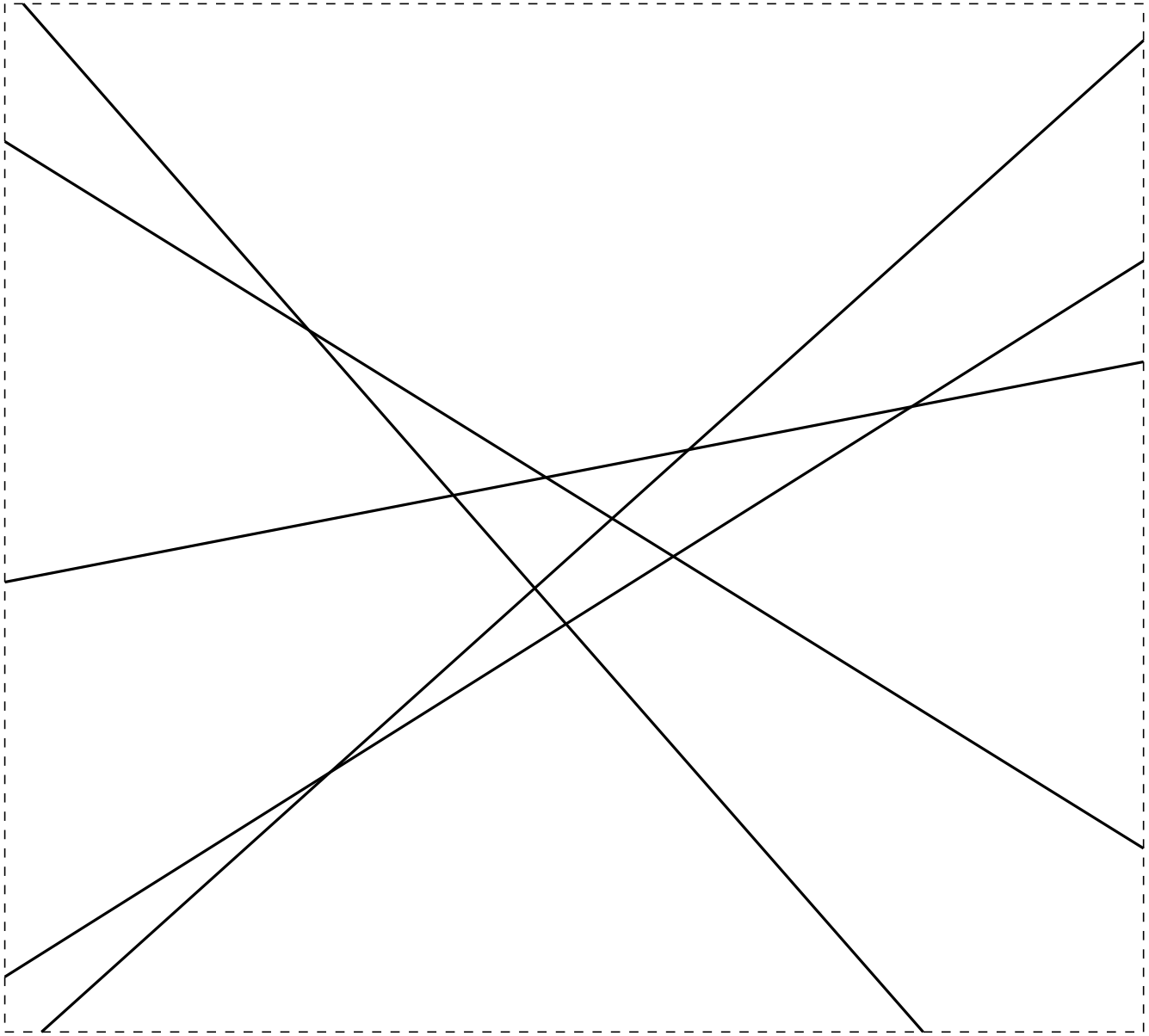
The *(combinatorial) complexity* of an arrangement $\mathcal{A}(L)$ is its number of faces, edges and vertices.

The complexity of $\mathcal{A}(L)$ is $O(n^2)$, where $n = \#L$. Specifically:

$n = 5$

- $v \leq \frac{n(n-1)}{2}$
- $e \leq n^2$
- $f \leq \frac{n^2}{2} + \frac{n}{2} + 1$

These bounds are achieved when the arrangement is simple.



COMPLEXITY

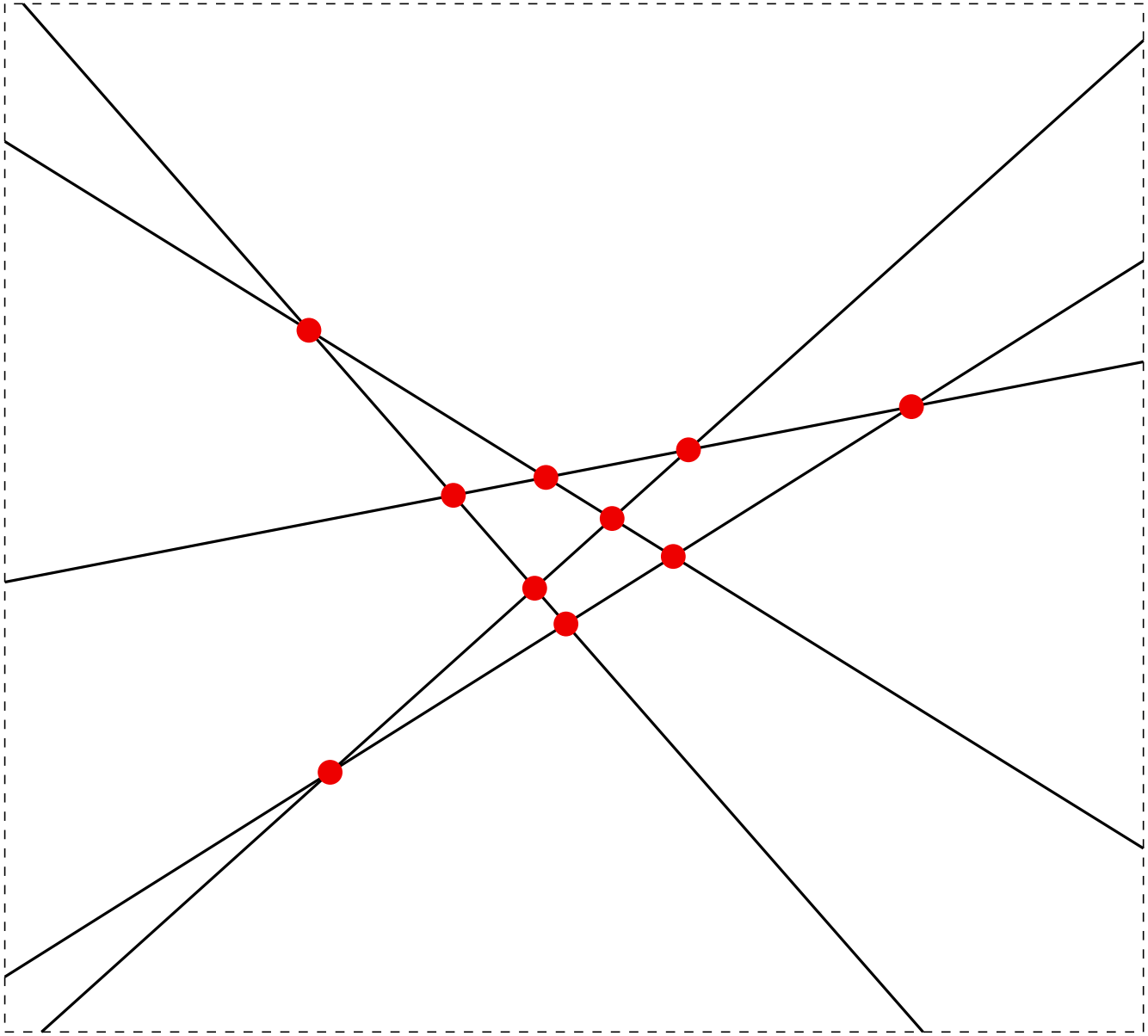
The *(combinatorial) complexity* of an arrangement $\mathcal{A}(L)$ is its number of faces, edges and vertices.

The complexity of $\mathcal{A}(L)$ is $O(n^2)$, where $n = \#L$. Specifically:

$n = 5$

- $v \leq \frac{n(n-1)}{2}$ $\quad v = 10$
- $e \leq n^2$
- $f \leq \frac{n^2}{2} + \frac{n}{2} + 1$

These bounds are achieved when the arrangement is simple.



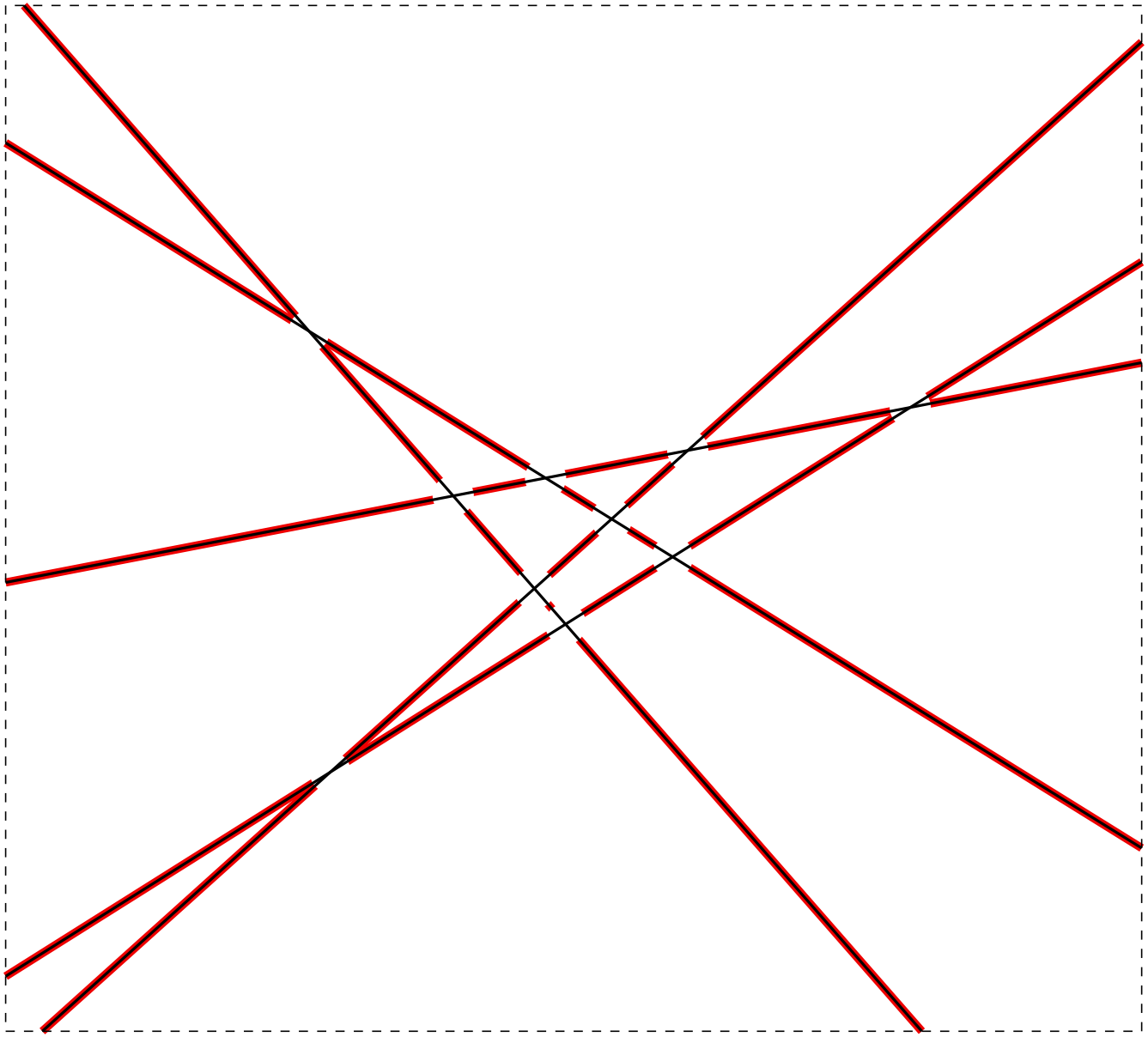
COMPLEXITY

The *(combinatorial) complexity* of an arrangement $\mathcal{A}(L)$ is its number of faces, edges and vertices.

The complexity of $\mathcal{A}(L)$ is $O(n^2)$, where $n = \#L$. Specifically: $n = 5$

- $v \leq \frac{n(n-1)}{2}$ $v = 10$
- $e \leq n^2$ $e = 25$
- $f \leq \frac{n^2}{2} + \frac{n}{2} + 1$

These bounds are achieved when the arrangement is simple.



COMPLEXITY

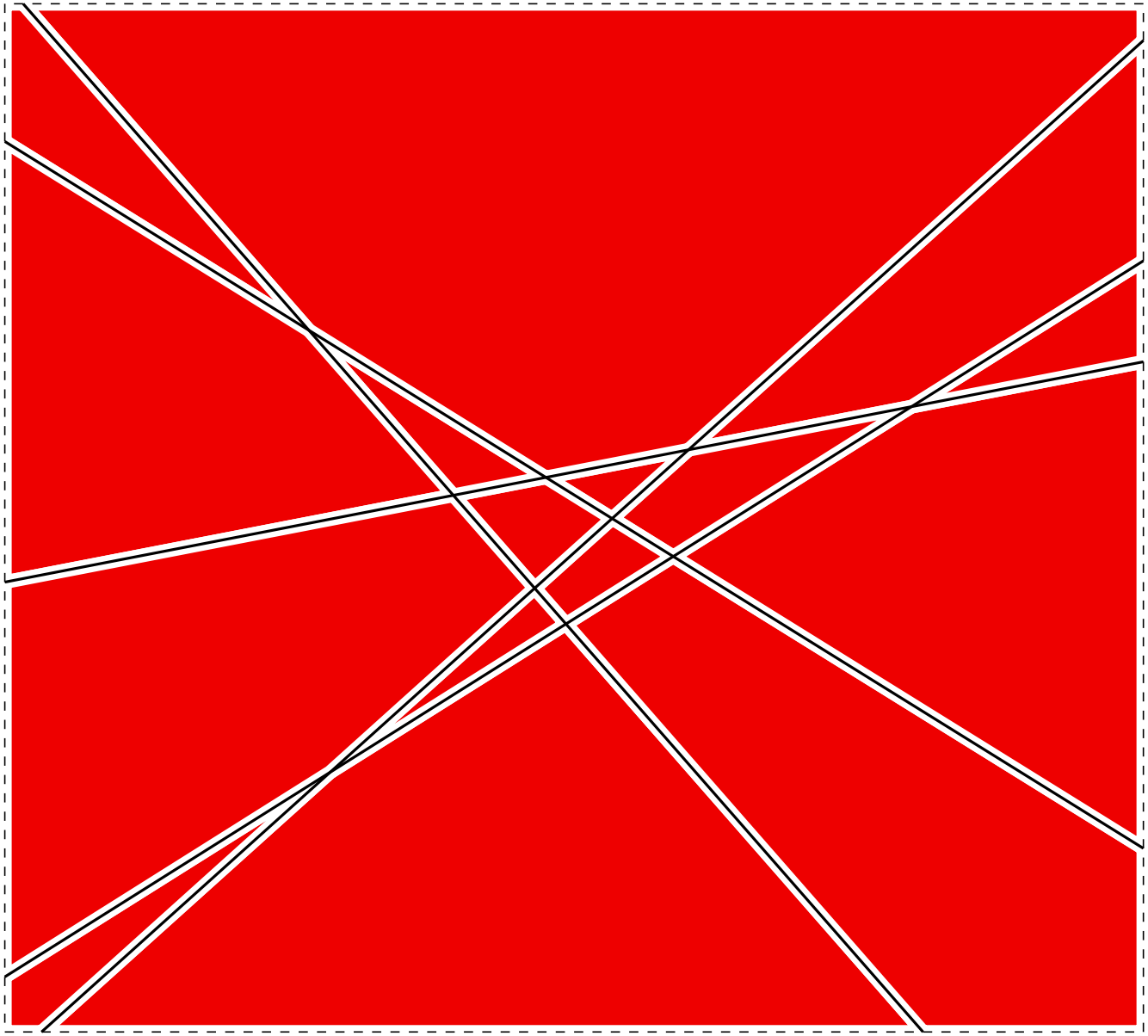
The *(combinatorial) complexity* of an arrangement $\mathcal{A}(L)$ is its number of faces, edges and vertices.

The complexity of $\mathcal{A}(L)$ is $O(n^2)$, where $n = \#L$. Specifically:

- $v \leq \frac{n(n-1)}{2}$ $\quad v = 10$
- $e \leq n^2$ $\quad e = 25$
- $f \leq \frac{n^2}{2} + \frac{n}{2} + 1$ $\quad f = 16$

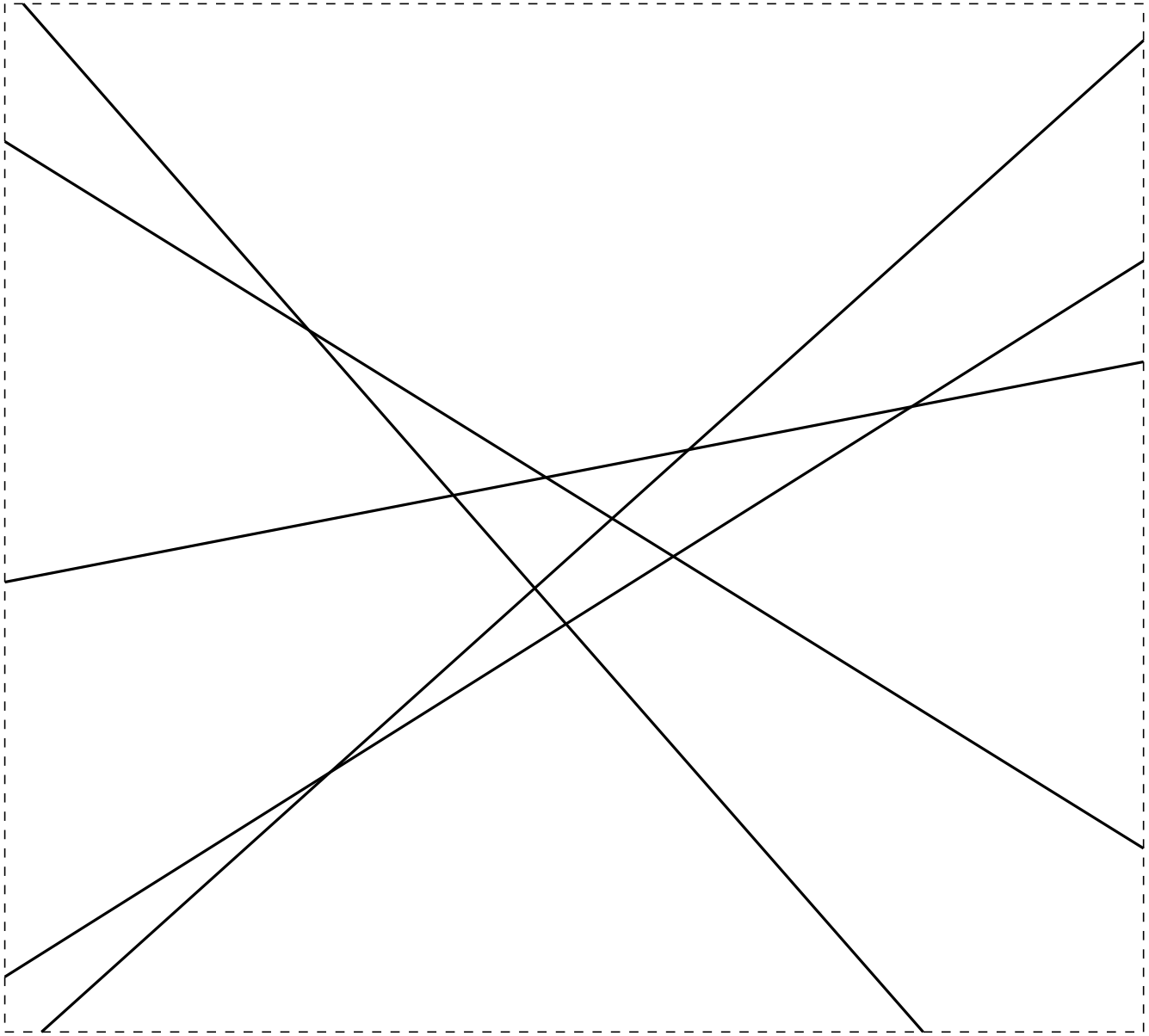
$n = 5$

These bounds are achieved when the arrangement is simple.



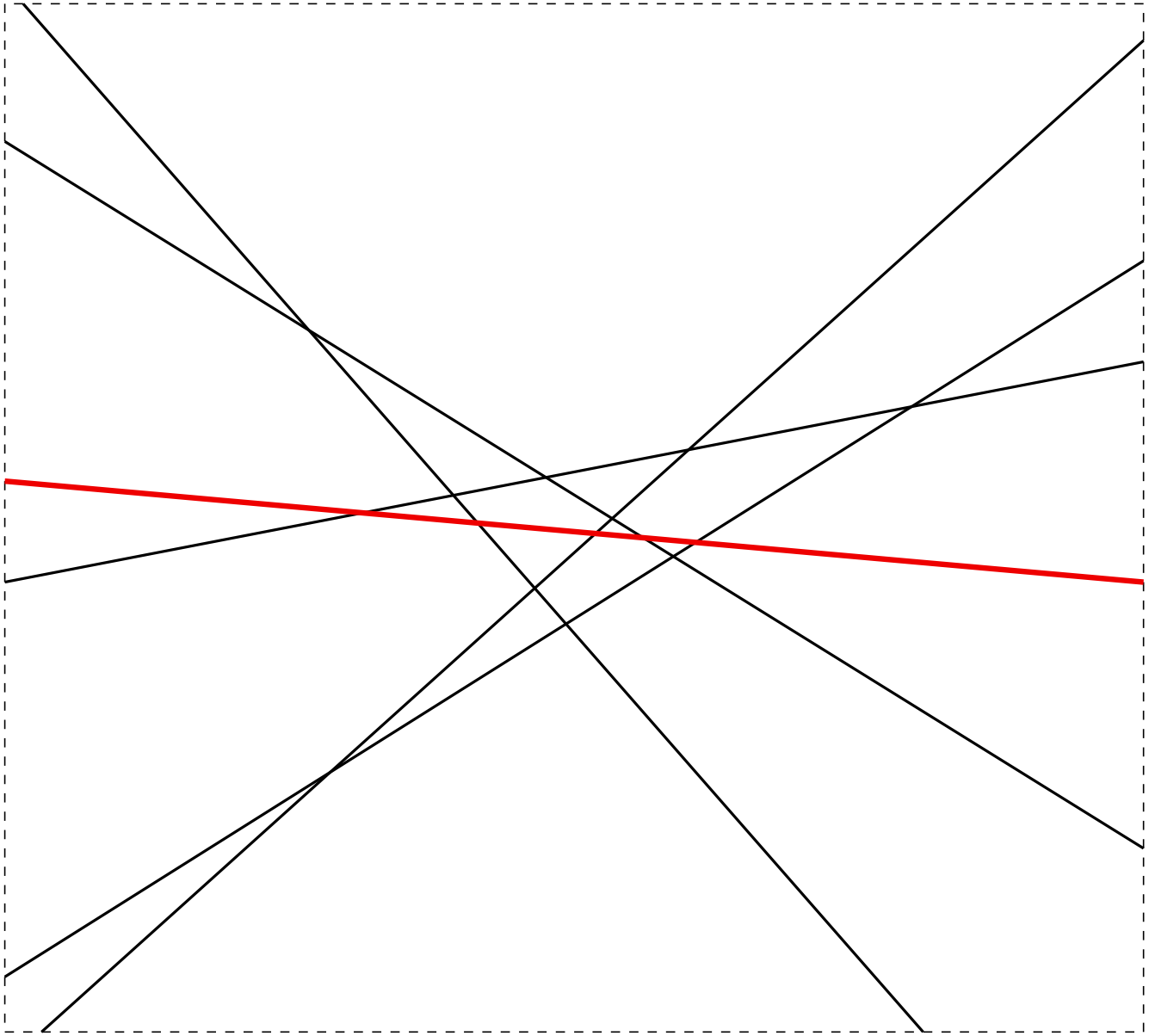
THE ZONE THEOREM

Definition: Given an arrangement $\mathcal{A}(L)$ and a line h, the *zone* of h is the set of faces of the arrangement intersected by h.



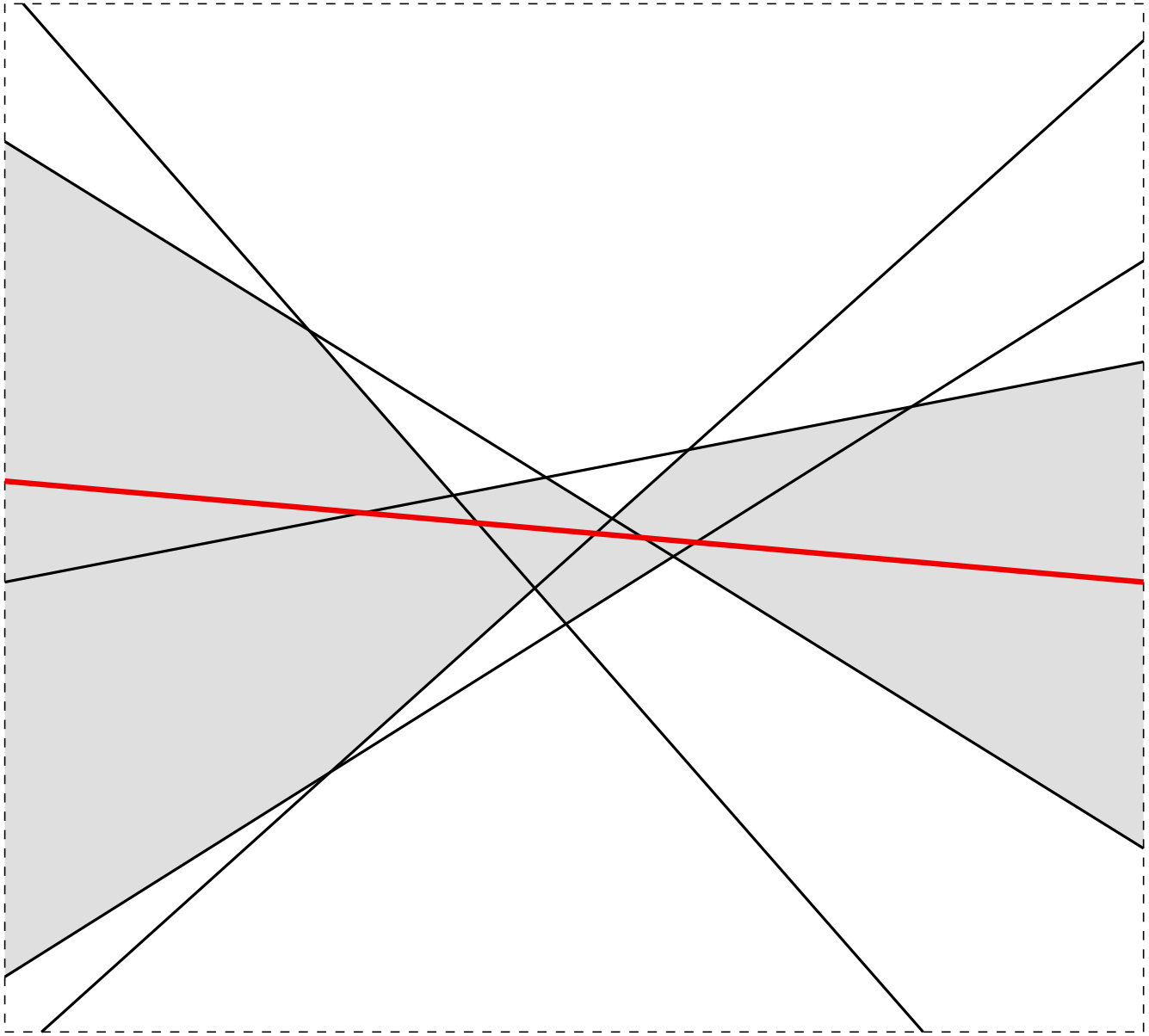
THE ZONE THEOREM

Definition: Given an arrangement $\mathcal{A}(L)$ and a line h, the *zone* of h is the set of faces of the arrangement intersected by h.



THE ZONE THEOREM

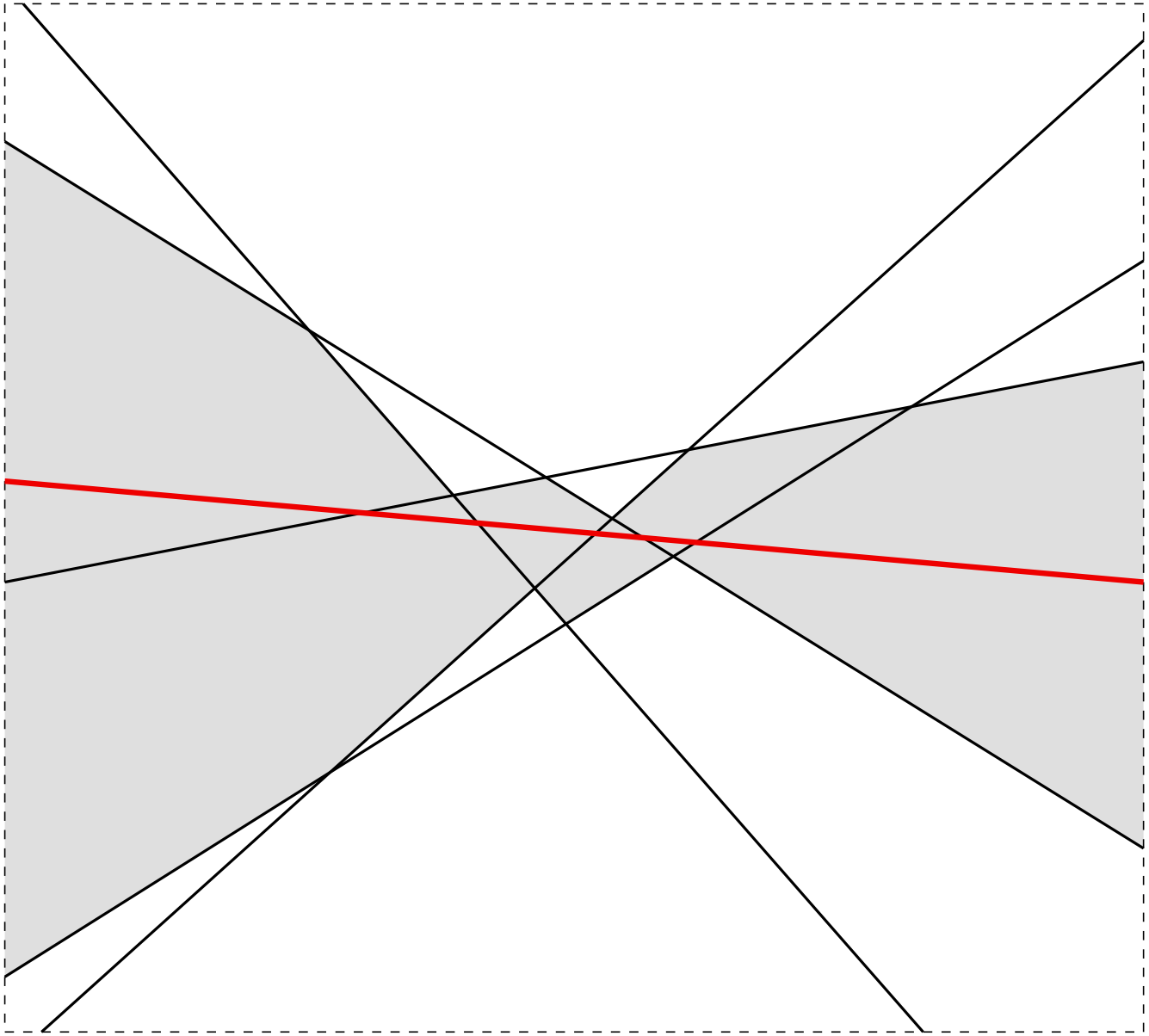
Definition: Given an arrangement $\mathcal{A}(L)$ and a line h, the *zone* of h is the set of faces of the arrangement intersected by h.



THE ZONE THEOREM

Definition: Given an arrangement $\mathcal{A}(L)$ and a line h, the *zone* of h is the set of faces of the arrangement intersected by h.

Theorem: The complexity of the zone of a line h in an arrangement of n lines is $O(n)$. More precisely, the total number of edges of the faces of the zone is $\leq 8n$.

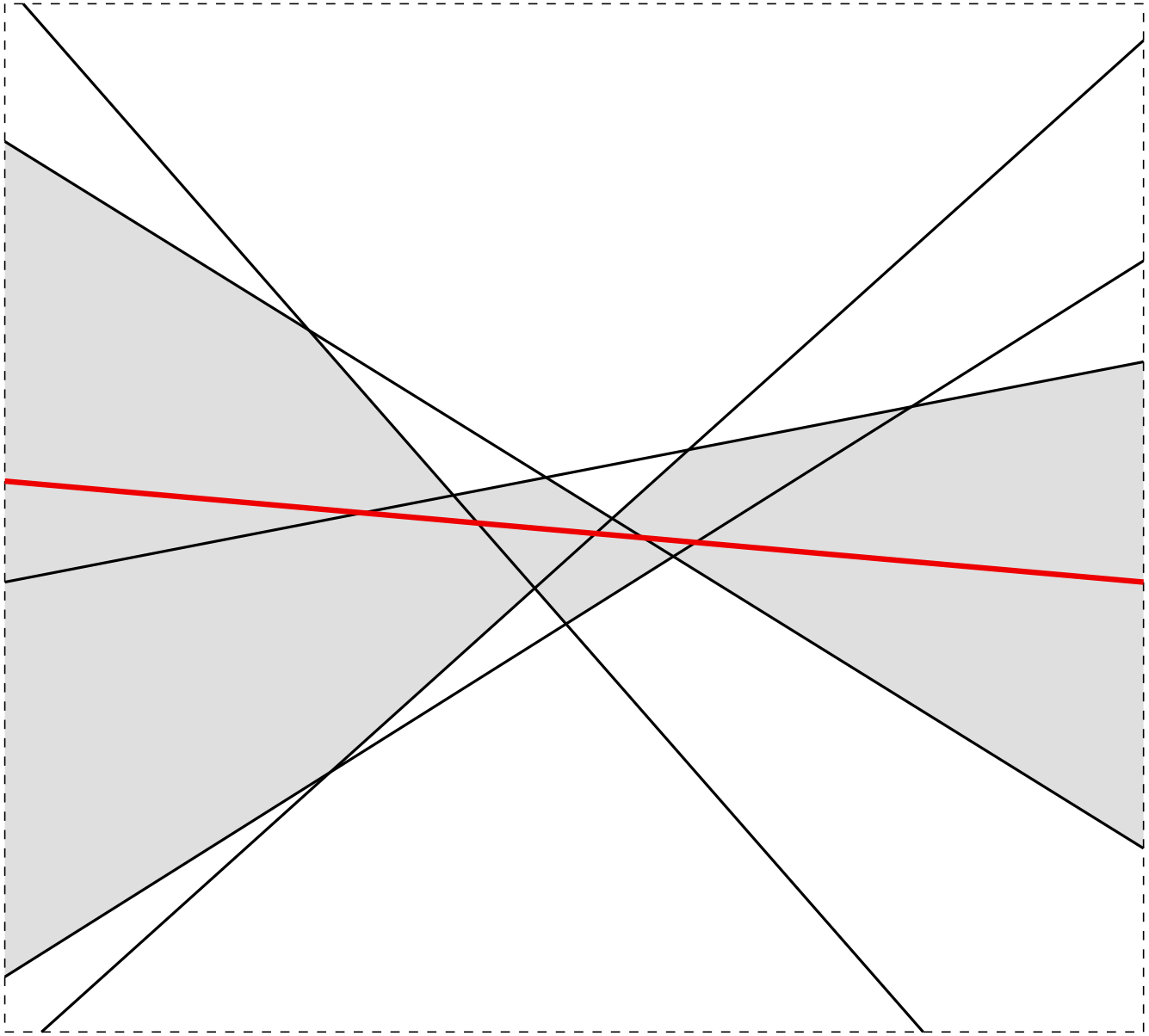


THE ZONE THEOREM

Definition: Given an arrangement $\mathcal{A}(L)$ and a line h, the *zone* of h is the set of faces of the arrangement intersected by h.

Theorem: The complexity of the zone of a line h in an arrangement of n lines is $O(n)$. More precisely, the total number of edges of the faces of the zone is $\leq 8n$.

Proof by induction, counting the left and the right edges of the faces in the zone of h (WLG, it can be assumed there are no horizontal lines in L).

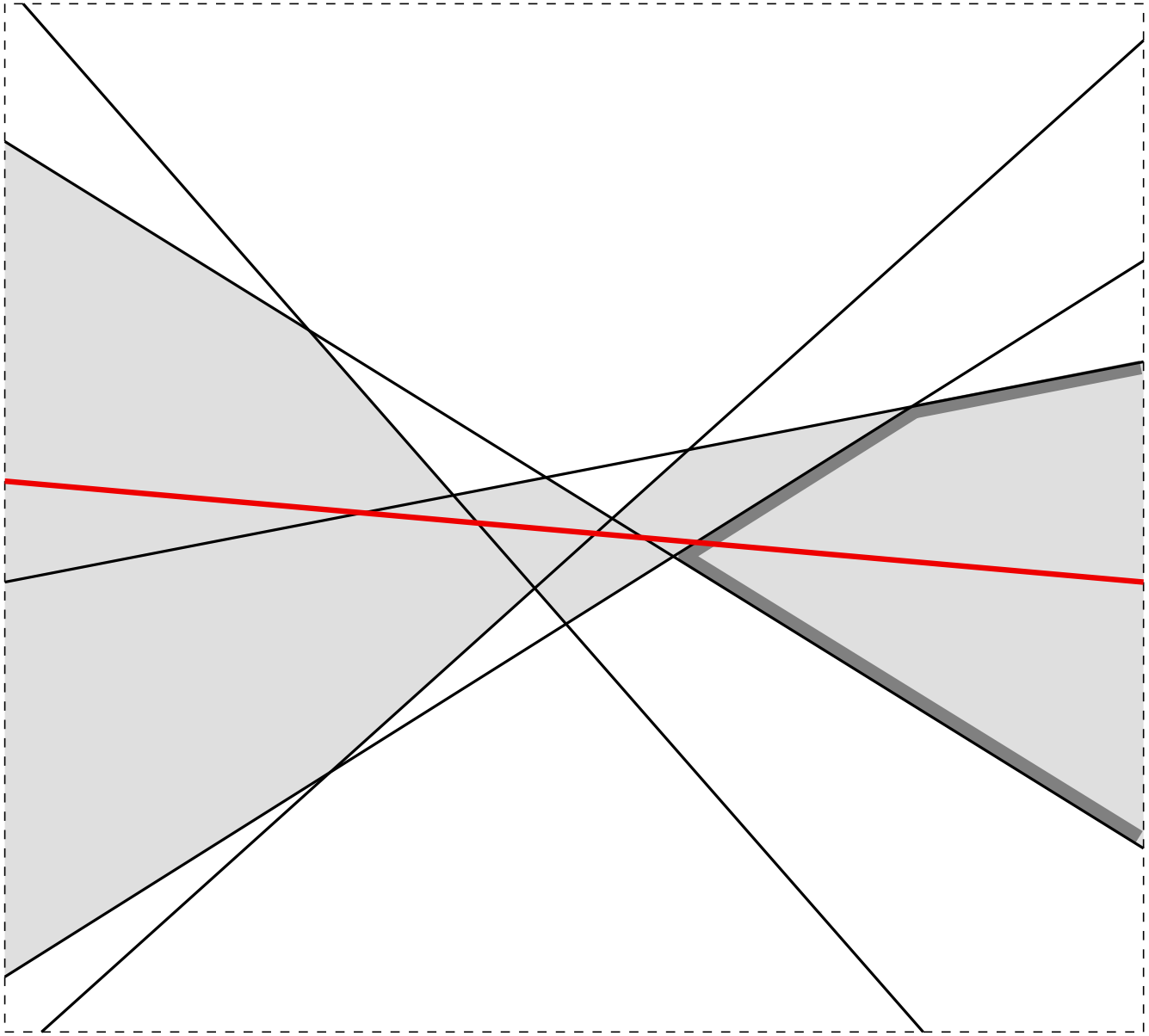


THE ZONE THEOREM

Definition: Given an arrangement $\mathcal{A}(L)$ and a line h, the *zone* of h is the set of faces of the arrangement intersected by h.

Theorem: The complexity of the zone of a line h in an arrangement of n lines is $O(n)$. More precisely, the total number of edges of the faces of the zone is $\leq 8n$.

Proof by induction, counting the left and the right edges of the faces in the zone of h (WLG, it can be assumed there are no horizontal lines in L).

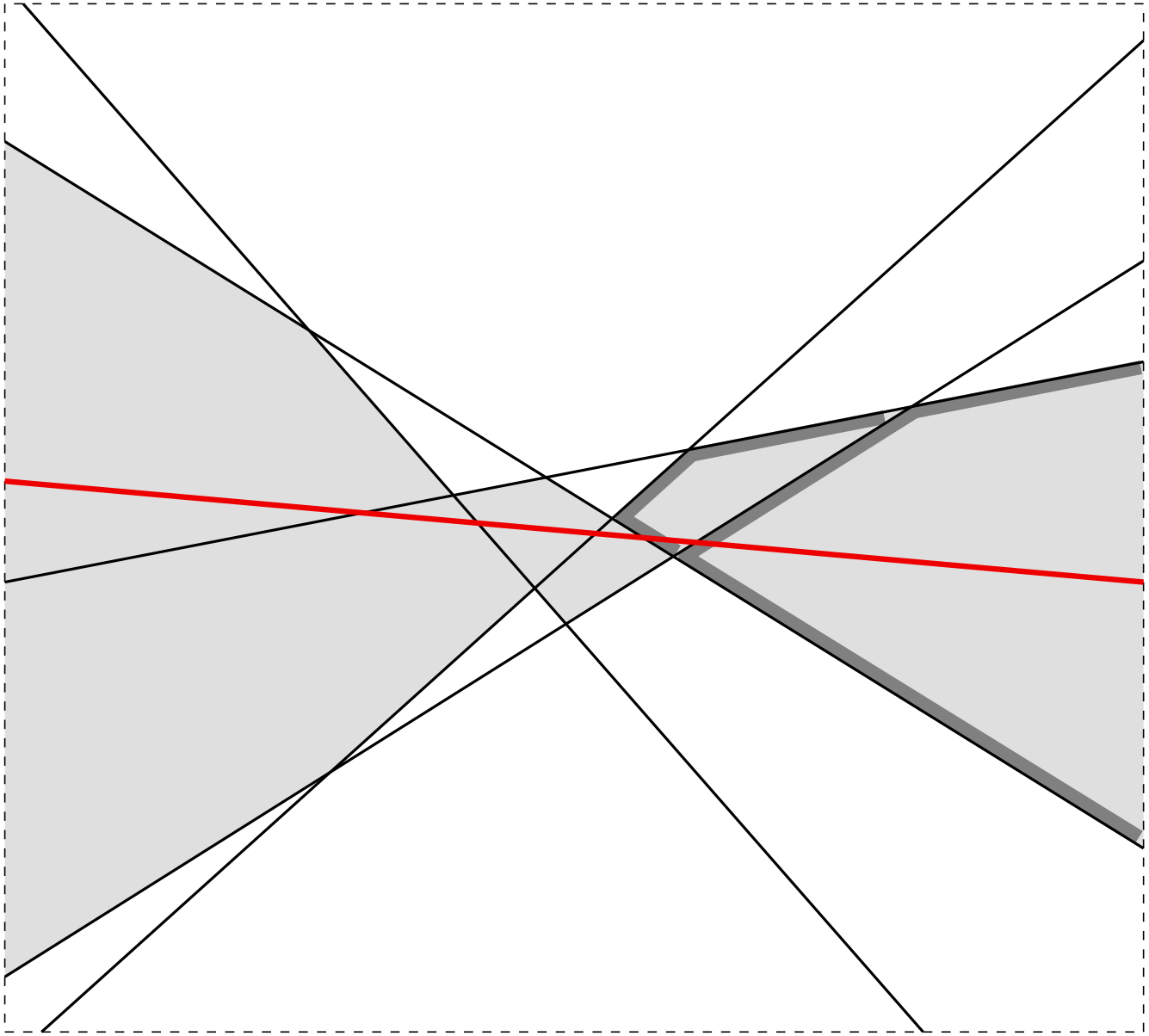


THE ZONE THEOREM

Definition: Given an arrangement $\mathcal{A}(L)$ and a line h, the *zone* of h is the set of faces of the arrangement intersected by h.

Theorem: The complexity of the zone of a line h in an arrangement of n lines is $O(n)$. More precisely, the total number of edges of the faces of the zone is $\leq 8n$.

Proof by induction, counting the left and the right edges of the faces in the zone of h (WLG, it can be assumed there are no horizontal lines in L).

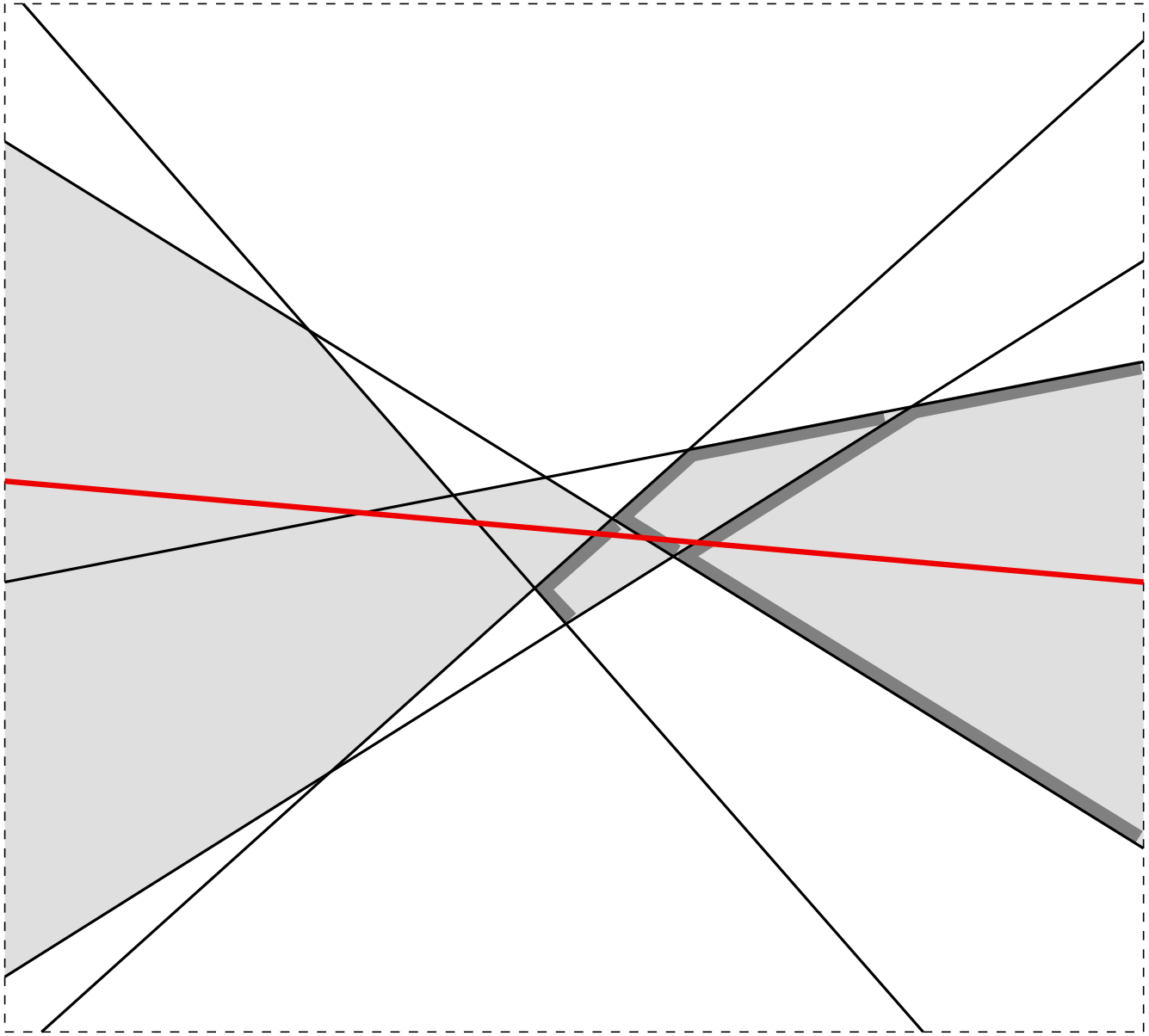


THE ZONE THEOREM

Definition: Given an arrangement $\mathcal{A}(L)$ and a line h, the *zone* of h is the set of faces of the arrangement intersected by h.

Theorem: The complexity of the zone of a line h in an arrangement of n lines is $O(n)$. More precisely, the total number of edges of the faces of the zone is $\leq 8n$.

Proof by induction, counting the left and the right edges of the faces in the zone of h (WLG, it can be assumed there are no horizontal lines in L).

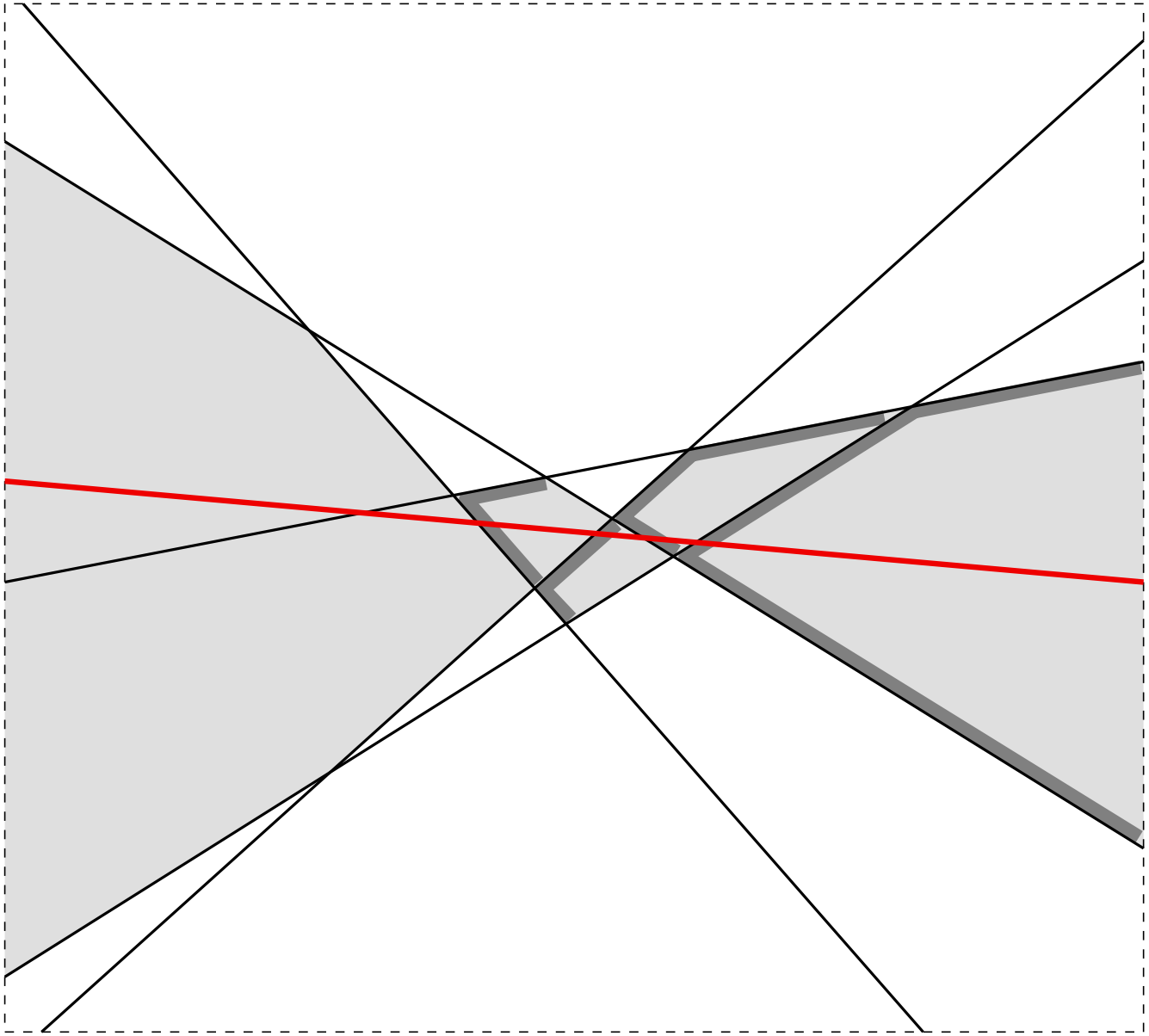


THE ZONE THEOREM

Definition: Given an arrangement $\mathcal{A}(L)$ and a line h, the *zone* of h is the set of faces of the arrangement intersected by h.

Theorem: The complexity of the zone of a line h in an arrangement of n lines is $O(n)$. More precisely, the total number of edges of the faces of the zone is $\leq 8n$.

Proof by induction, counting the left and the right edges of the faces in the zone of h (WLG, it can be assumed there are no horizontal lines in L).

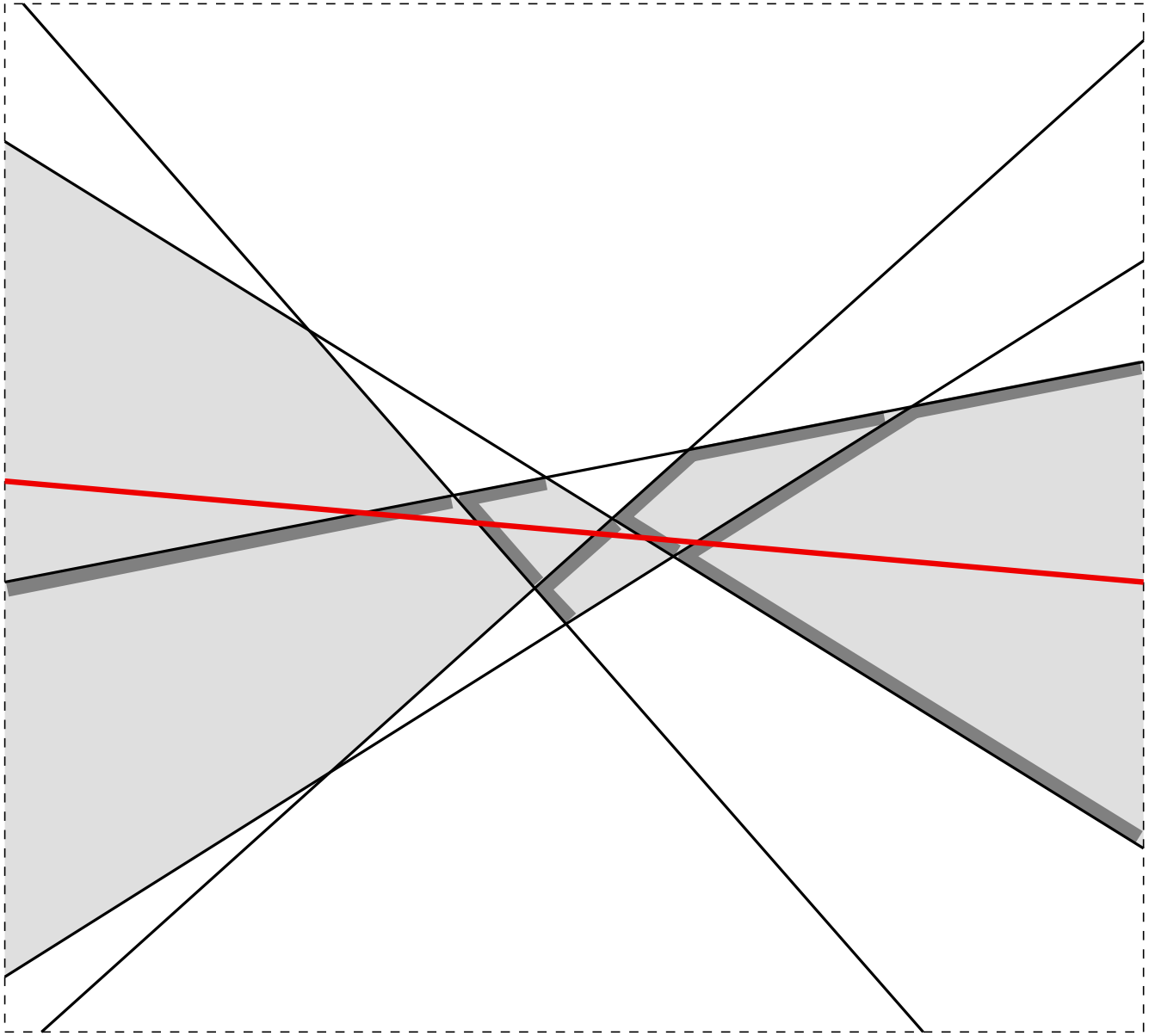


THE ZONE THEOREM

Definition: Given an arrangement $\mathcal{A}(L)$ and a line h, the *zone* of h is the set of faces of the arrangement intersected by h.

Theorem: The complexity of the zone of a line h in an arrangement of n lines is $O(n)$. More precisely, the total number of edges of the faces of the zone is $\leq 8n$.

Proof by induction, counting the left and the right edges of the faces in the zone of h (WLG, it can be assumed there are no horizontal lines in L).

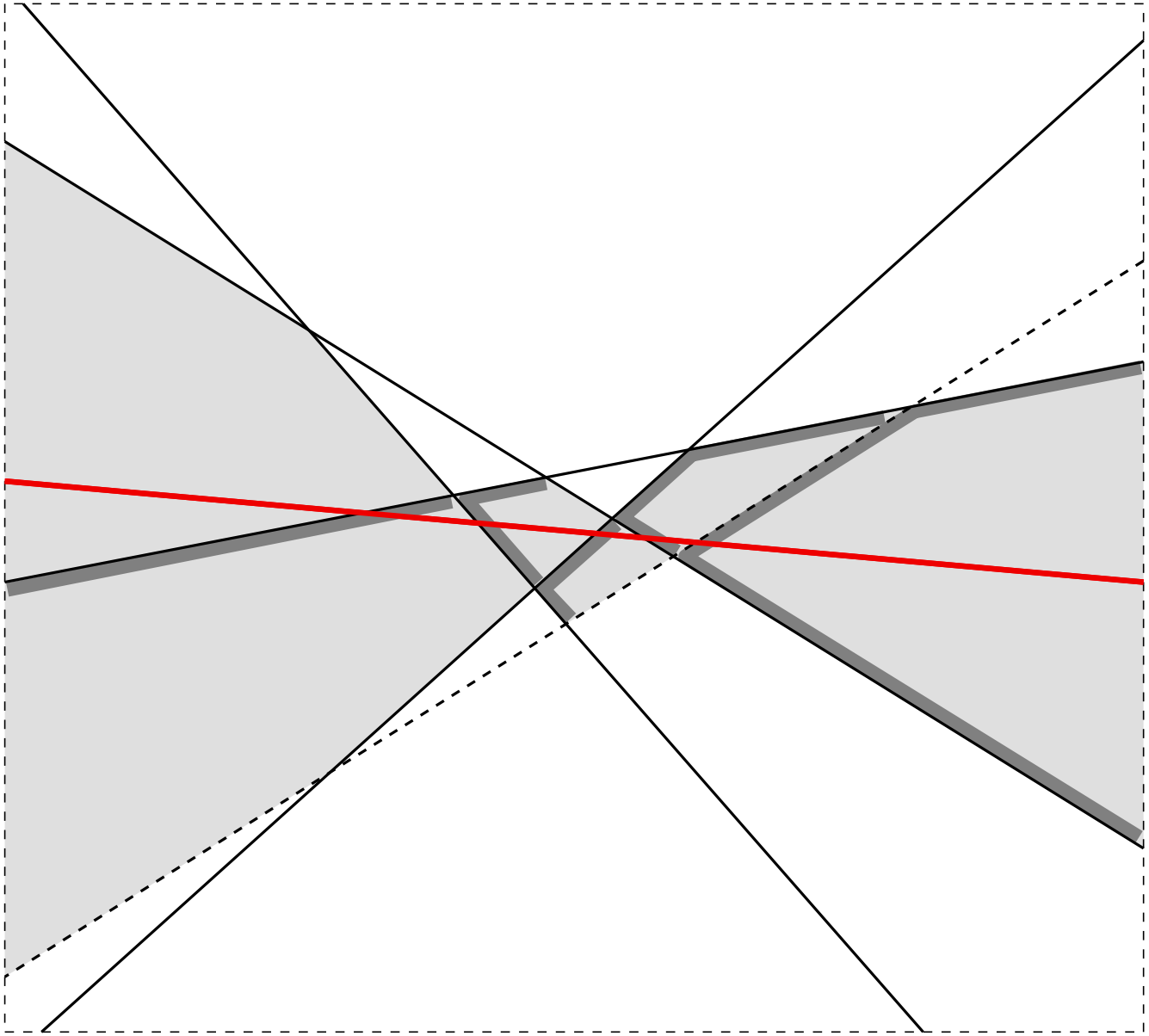


THE ZONE THEOREM

Definition: Given an arrangement $\mathcal{A}(L)$ and a line h, the *zone* of h is the set of faces of the arrangement intersected by h.

Theorem: The complexity of the zone of a line h in an arrangement of n lines is $O(n)$. More precisely, the total number of edges of the faces of the zone is $\leq 8n$.

Proof by induction, counting the left and the right edges of the faces in the zone of h (WLG, it can be assumed there are no horizontal lines in L).

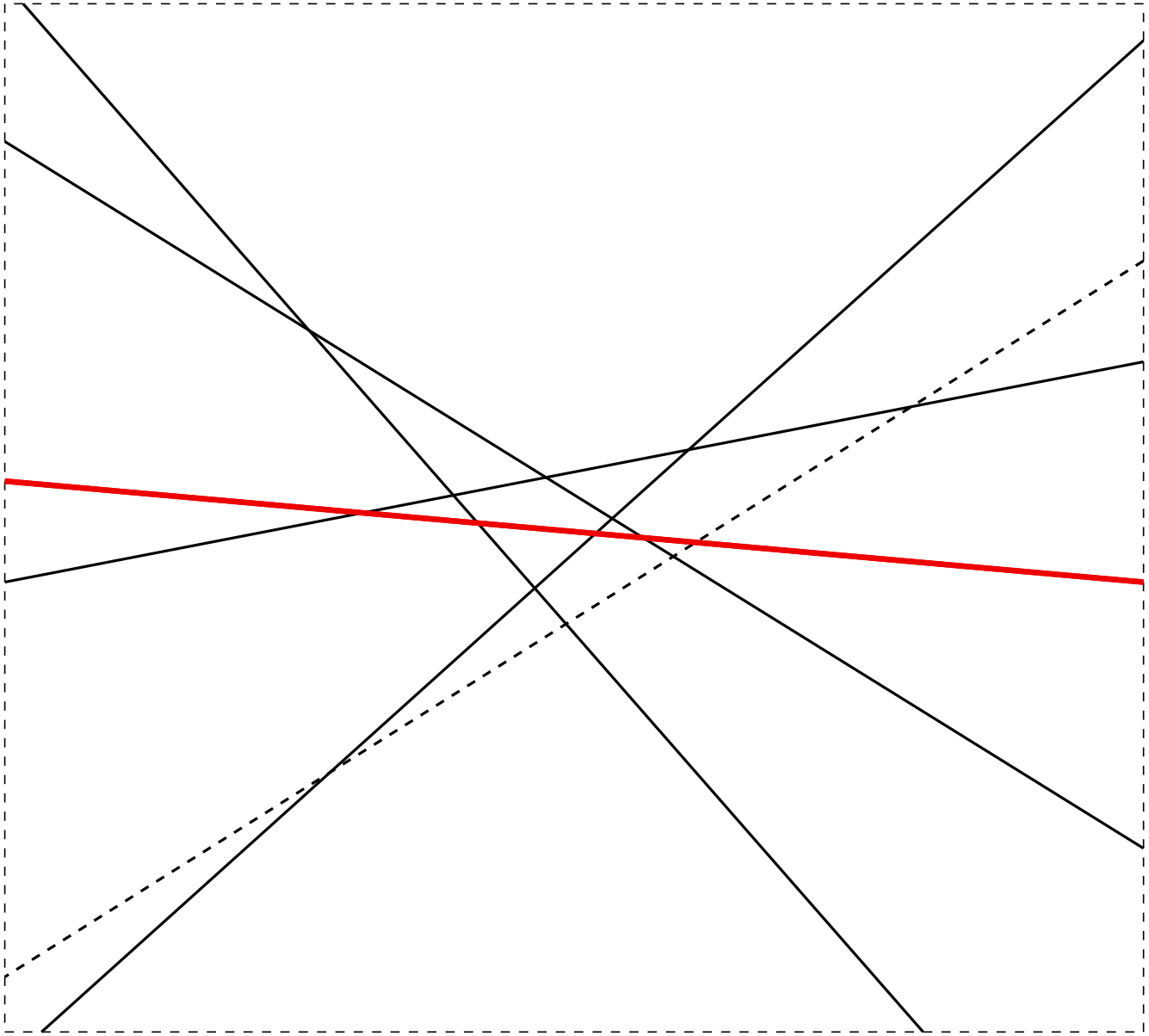


THE ZONE THEOREM

Definition: Given an arrangement $\mathcal{A}(L)$ and a line h, the *zone* of h is the set of faces of the arrangement intersected by h.

Theorem: The complexity of the zone of a line h in an arrangement of n lines is $O(n)$. More precisely, the total number of edges of the faces of the zone is $\leq 8n$.

Proof by induction, counting the left and the right edges of the faces in the zone of h (WLG, it can be assumed there are no horizontal lines in L).

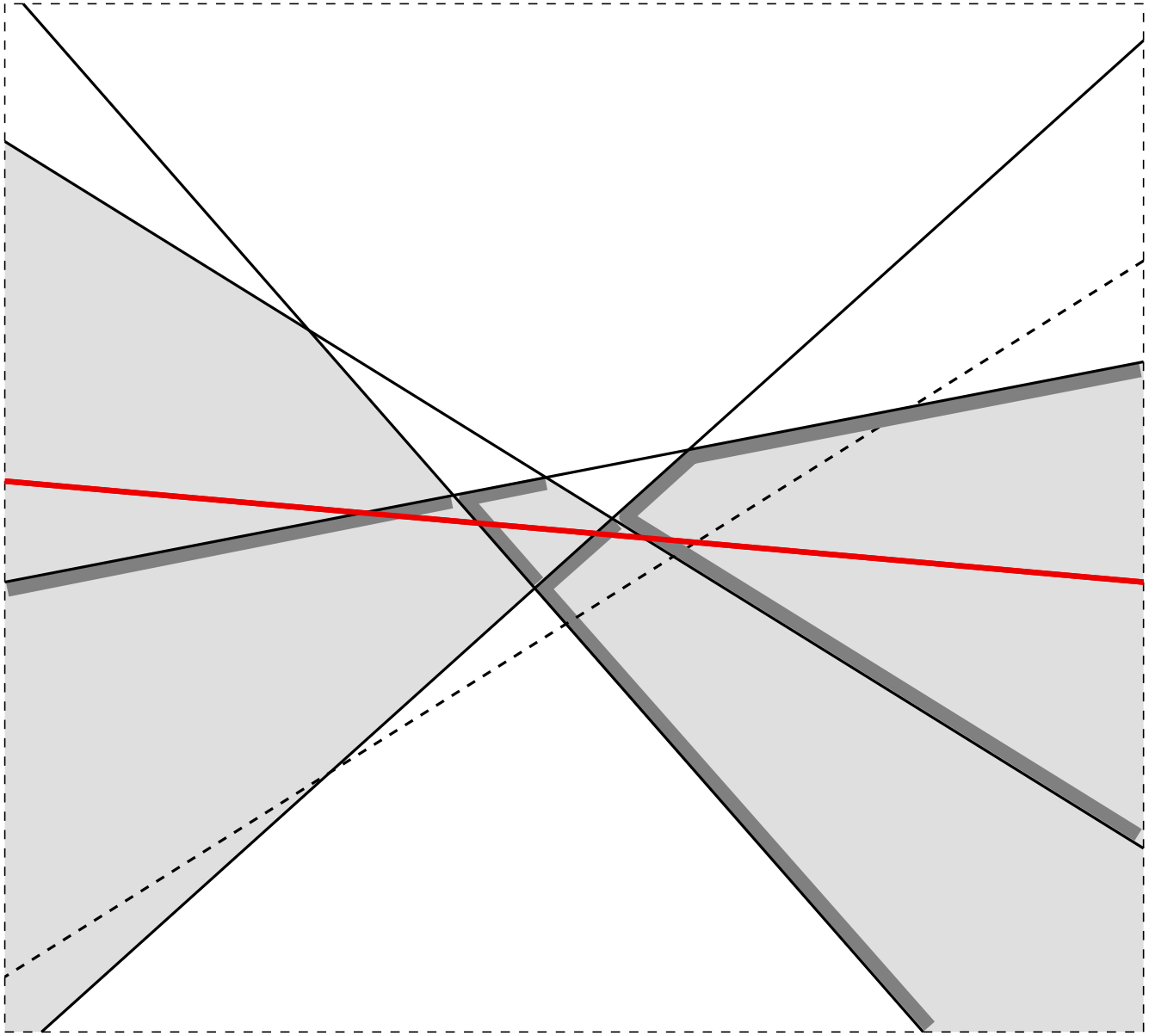


THE ZONE THEOREM

Definition: Given an arrangement $\mathcal{A}(L)$ and a line h, the *zone* of h is the set of faces of the arrangement intersected by h.

Theorem: The complexity of the zone of a line h in an arrangement of n lines is $O(n)$. More precisely, the total number of edges of the faces of the zone is $\leq 8n$.

Proof by induction, counting the left and the right edges of the faces in the zone of h (WLG, it can be assumed there are no horizontal lines in L).

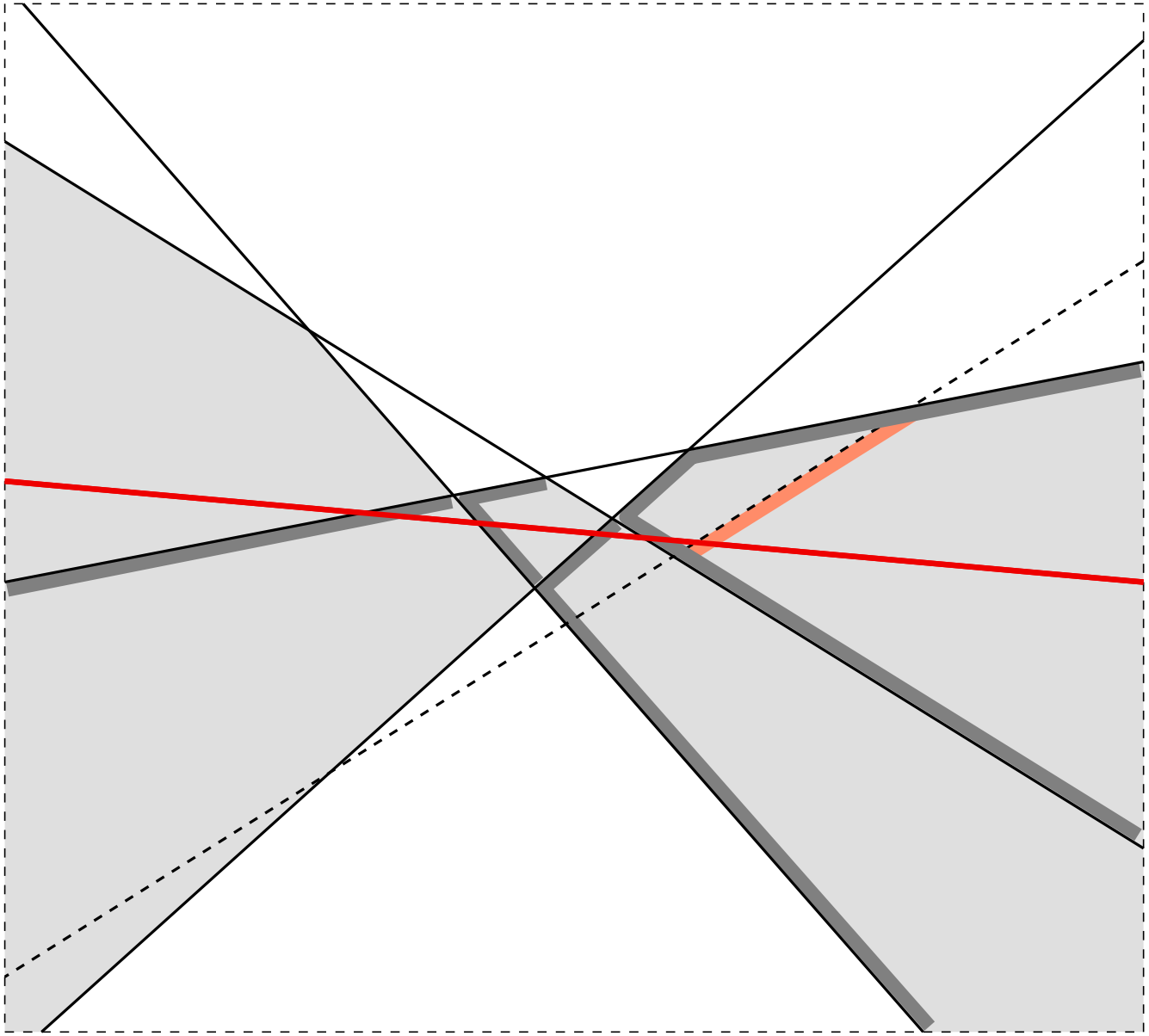


THE ZONE THEOREM

Definition: Given an arrangement $\mathcal{A}(L)$ and a line h, the *zone* of h is the set of faces of the arrangement intersected by h.

Theorem: The complexity of the zone of a line h in an arrangement of n lines is $O(n)$. More precisely, the total number of edges of the faces of the zone is $\leq 8n$.

Proof by induction, counting the left and the right edges of the faces in the zone of h (WLG, it can be assumed there are no horizontal lines in L).

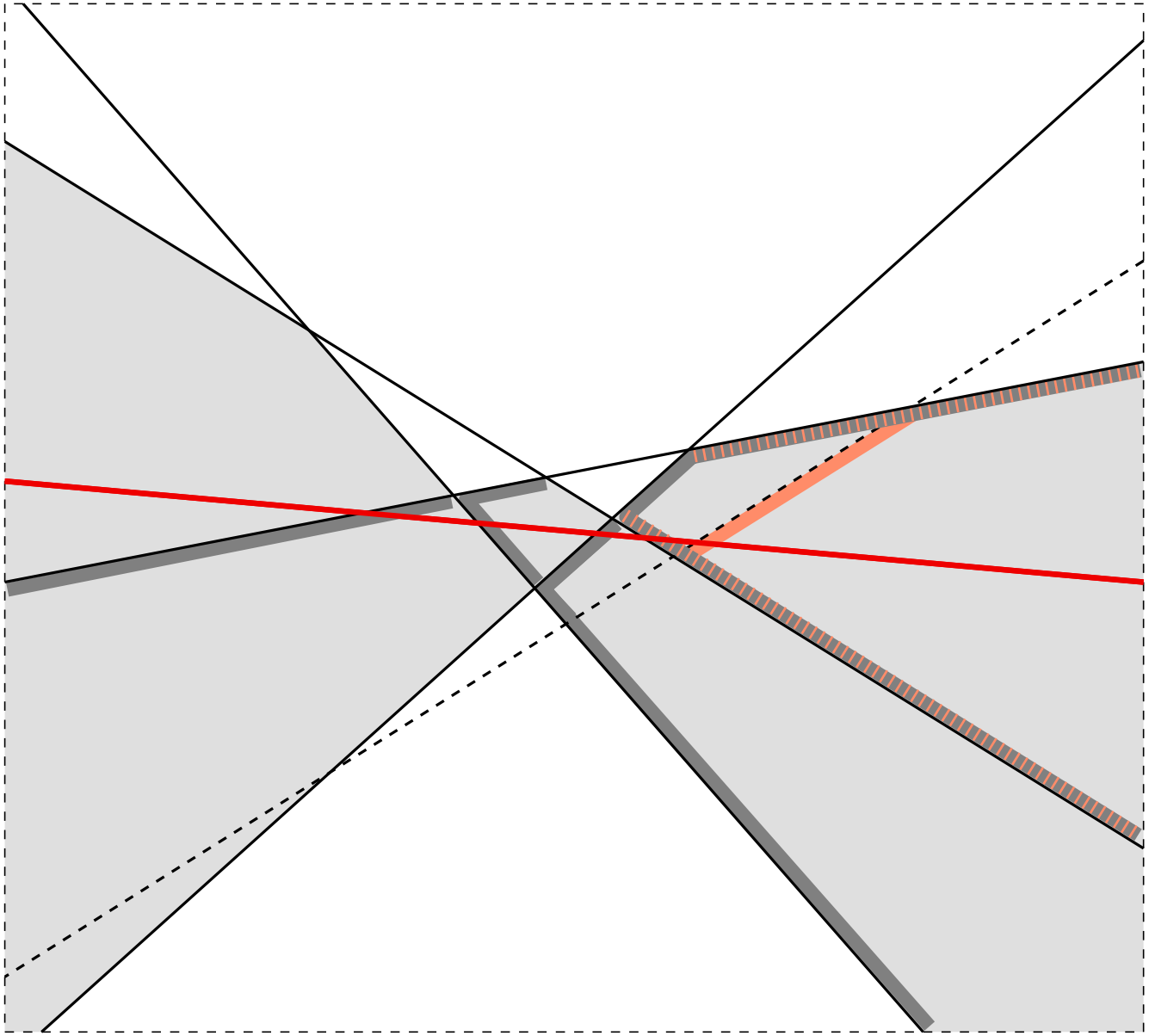


THE ZONE THEOREM

Definition: Given an arrangement $\mathcal{A}(L)$ and a line h, the *zone* of h is the set of faces of the arrangement intersected by h.

Theorem: The complexity of the zone of a line h in an arrangement of n lines is $O(n)$. More precisely, the total number of edges of the faces of the zone is $\leq 8n$.

Proof by induction, counting the left and the right edges of the faces in the zone of h (WLG, it can be assumed there are no horizontal lines in L).

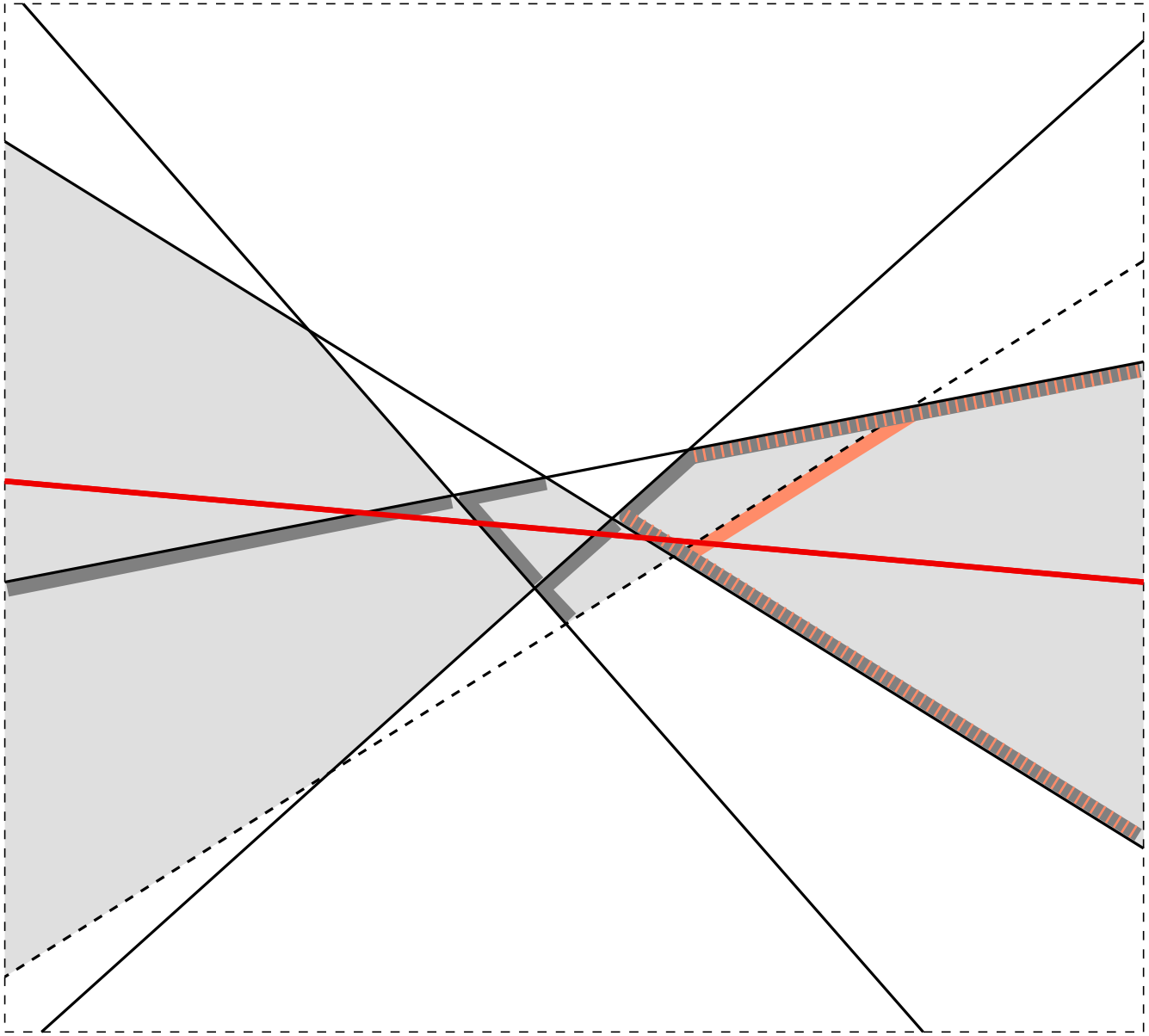


THE ZONE THEOREM

Definition: Given an arrangement $\mathcal{A}(L)$ and a line h, the *zone* of h is the set of faces of the arrangement intersected by h.

Theorem: The complexity of the zone of a line h in an arrangement of n lines is $O(n)$. More precisely, the total number of edges of the faces of the zone is $\leq 8n$.

Proof by induction, counting the left and the right edges of the faces in the zone of h (WLG, it can be assumed there are no horizontal lines in L).



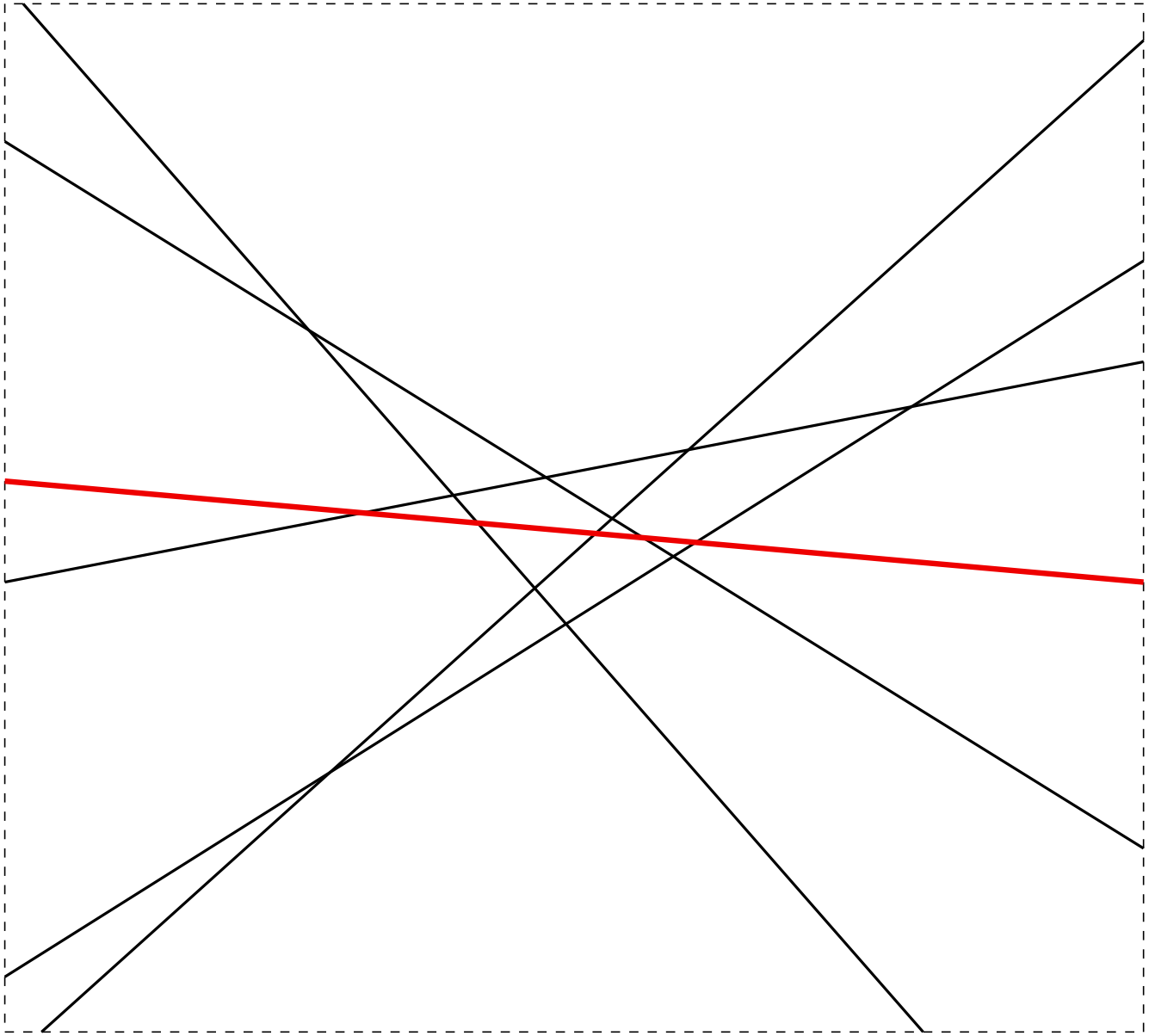
THE ZONE THEOREM

Definition: Given an arrangement $\mathcal{A}(L)$ and a line h, the *zone* of h is the set of faces of the arrangement intersected by h.

Theorem: The complexity of the zone of a line h in an arrangement of n lines is $O(n)$. More precisely, the total number of edges of the faces of the zone is $\leq 8n$.

Proof by induction, counting the left and the right edges of the faces in the zone of h (WLG, it can be assumed there are no horizontal lines in L).

(It is important to consider the rightmost intersecting line).



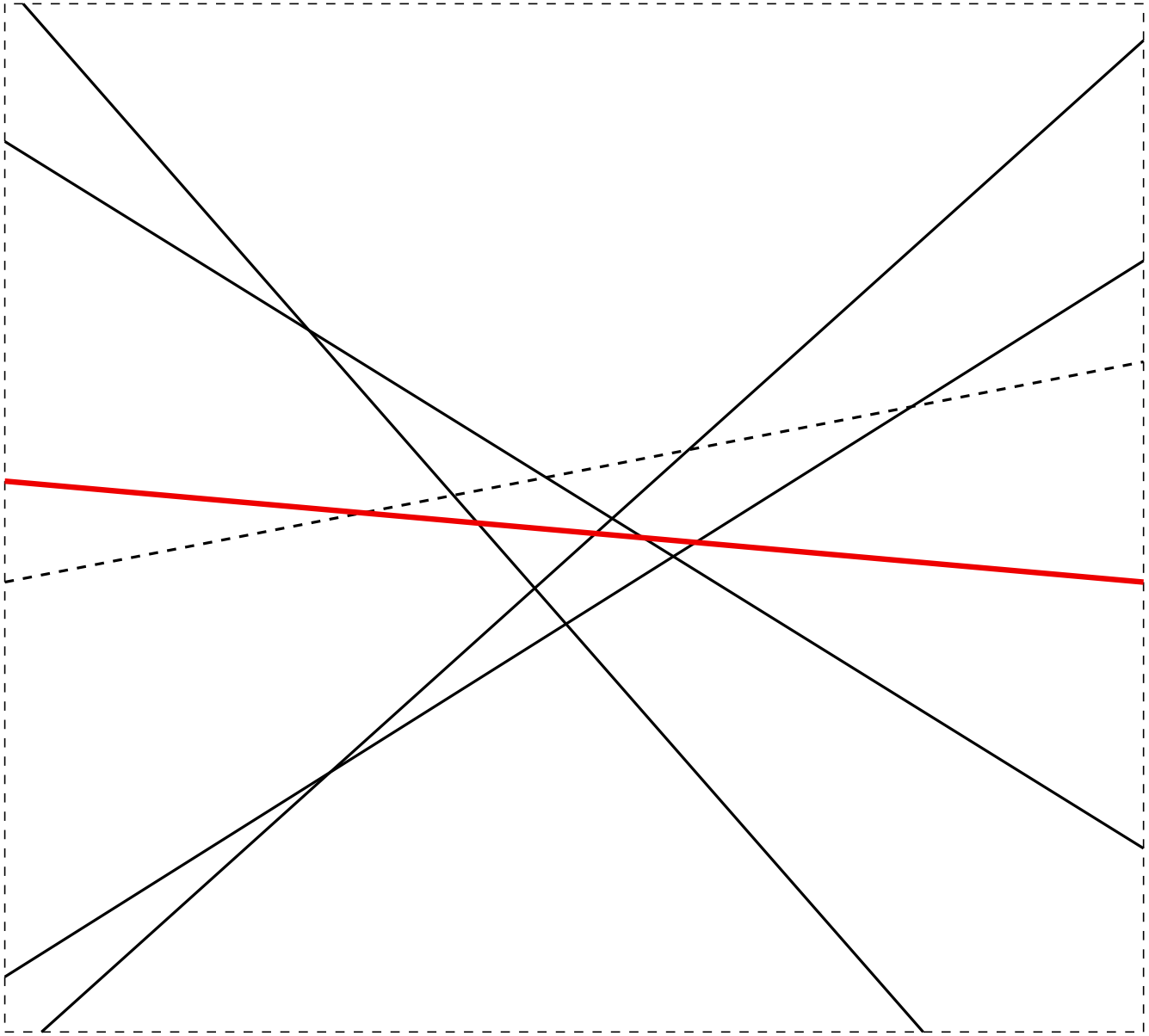
THE ZONE THEOREM

Definition: Given an arrangement $\mathcal{A}(L)$ and a line h, the *zone* of h is the set of faces of the arrangement intersected by h.

Theorem: The complexity of the zone of a line h in an arrangement of n lines is $O(n)$. More precisely, the total number of edges of the faces of the zone is $\leq 8n$.

Proof by induction, counting the left and the right edges of the faces in the zone of h (WLG, it can be assumed there are no horizontal lines in L).

(It is important to consider the rightmost intersecting line).



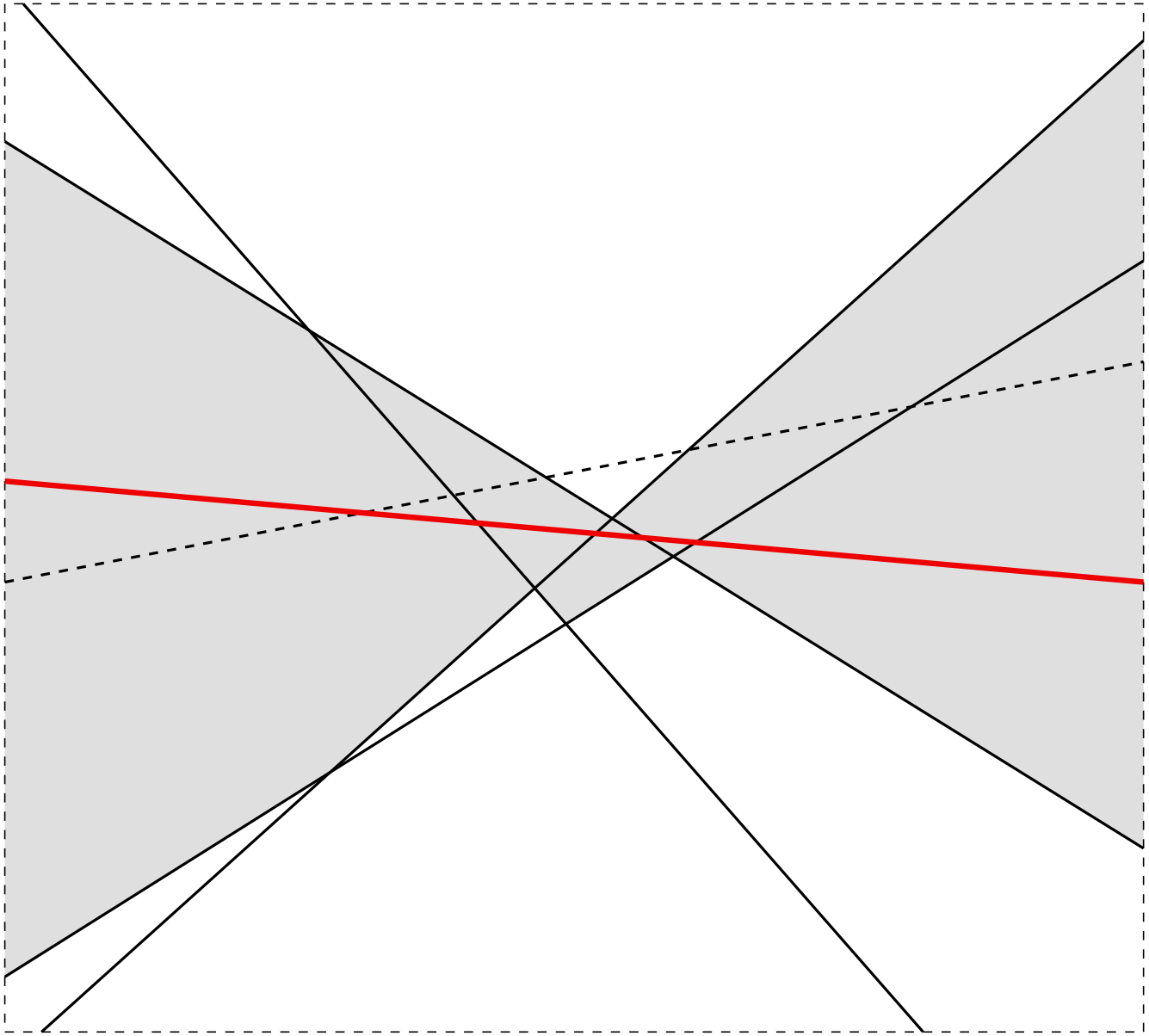
THE ZONE THEOREM

Definition: Given an arrangement $\mathcal{A}(L)$ and a line h, the *zone* of h is the set of faces of the arrangement intersected by h.

Theorem: The complexity of the zone of a line h in an arrangement of n lines is $O(n)$. More precisely, the total number of edges of the faces of the zone is $\leq 8n$.

Proof by induction, counting the left and the right edges of the faces in the zone of h (WLG, it can be assumed there are no horizontal lines in L).

(It is important to consider the rightmost intersecting line).



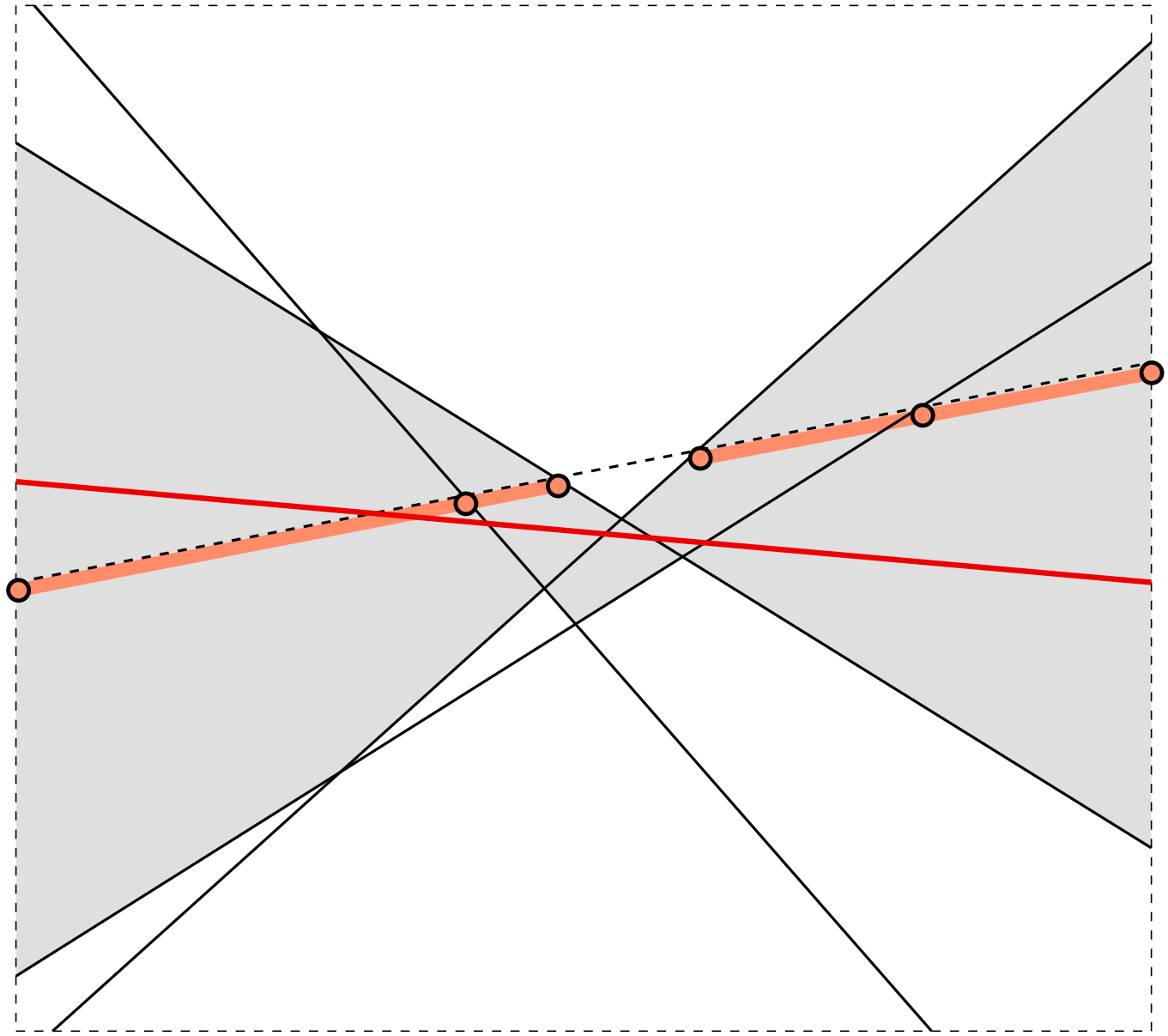
THE ZONE THEOREM

Definition: Given an arrangement $\mathcal{A}(L)$ and a line h, the *zone* of h is the set of faces of the arrangement intersected by h.

Theorem: The complexity of the zone of a line h in an arrangement of n lines is $O(n)$. More precisely, the total number of edges of the faces of the zone is $\leq 8n$.

Proof by induction, counting the left and the right edges of the faces in the zone of h (WLG, it can be assumed there are no horizontal lines in L).

(It is important to consider the rightmost intersecting line).



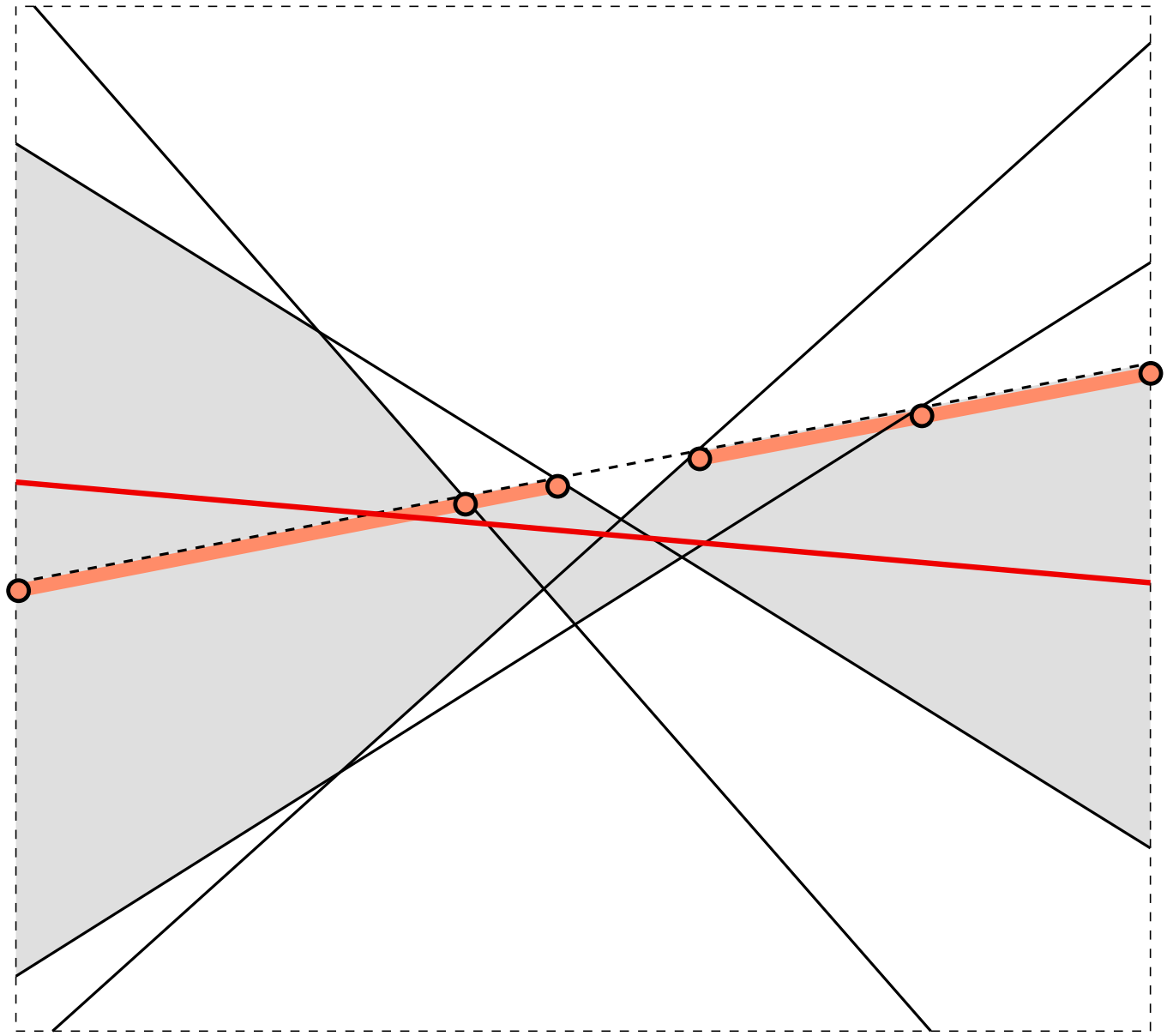
THE ZONE THEOREM

Definition: Given an arrangement $\mathcal{A}(L)$ and a line h, the *zone* of h is the set of faces of the arrangement intersected by h.

Theorem: The complexity of the zone of a line h in an arrangement of n lines is $O(n)$. More precisely, the total number of edges of the faces of the zone is $\leq 8n$.

Proof by induction, counting the left and the right edges of the faces in the zone of h (WLG, it can be assumed there are no horizontal lines in L).

(It is important to consider the rightmost intersecting line).

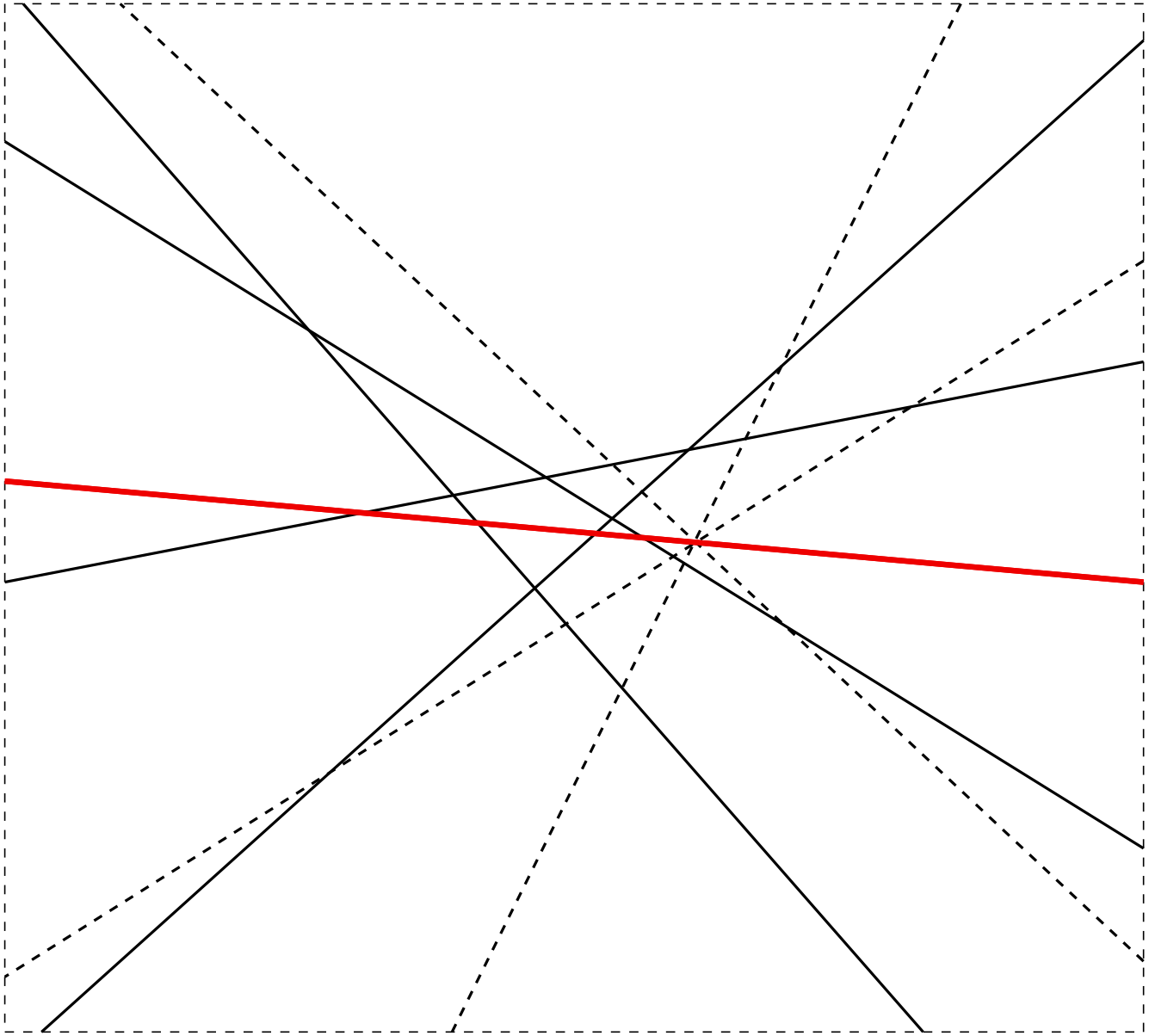


THE ZONE THEOREM

Definition: Given an arrangement $\mathcal{A}(L)$ and a line h, the *zone* of h is the set of faces of the arrangement intersected by h.

Theorem: The complexity of the zone of a line h in an arrangement of n lines is $O(n)$. More precisely, the total number of edges of the faces of the zone is $\leq 8n$.

Proof by induction, counting the left and the right edges of the faces in the zone of h (WLG, it can be assumed there are no horizontal lines in L).

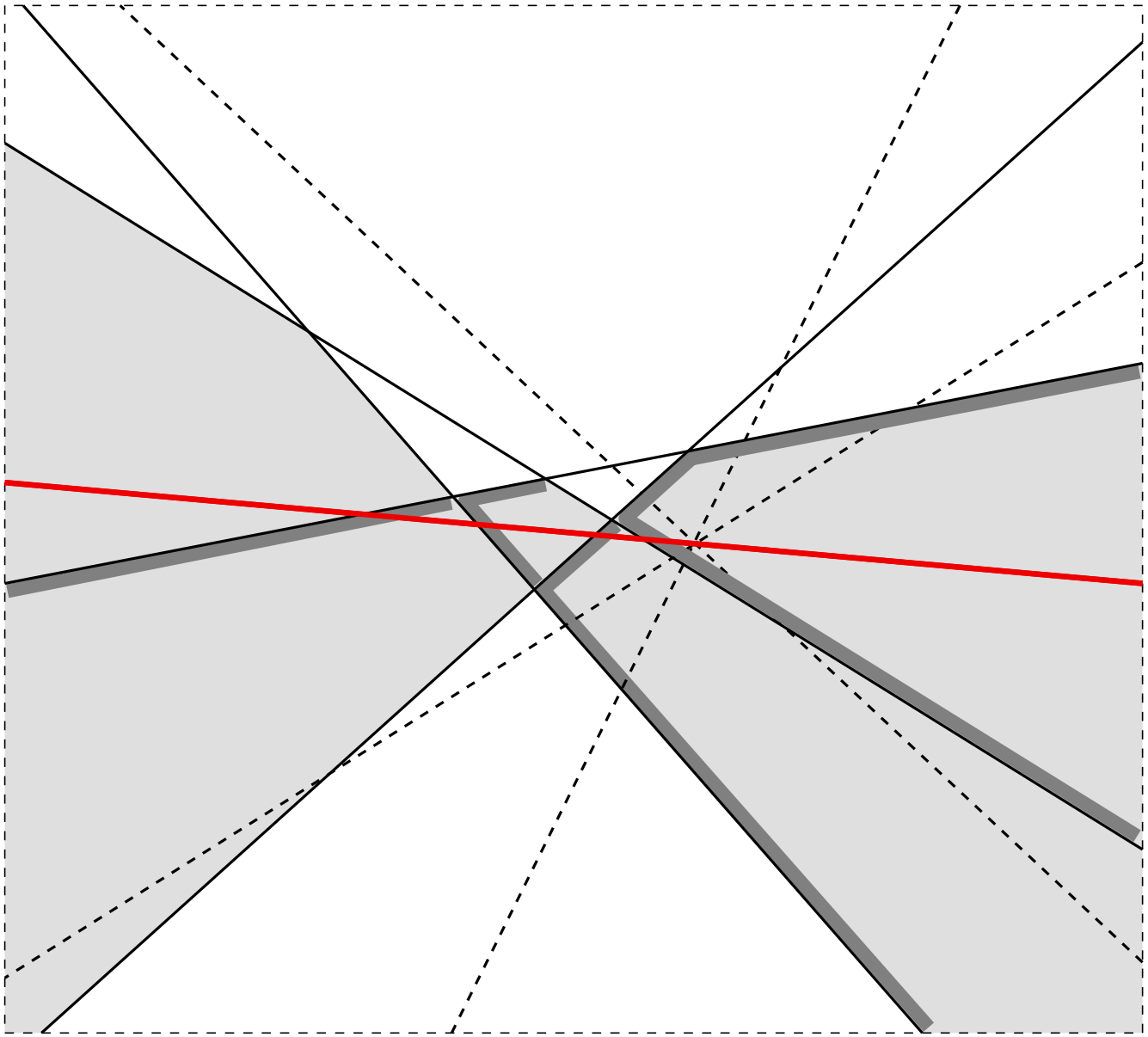


THE ZONE THEOREM

Definition: Given an arrangement $\mathcal{A}(L)$ and a line h, the *zone* of h is the set of faces of the arrangement intersected by h.

Theorem: The complexity of the zone of a line h in an arrangement of n lines is $O(n)$. More precisely, the total number of edges of the faces of the zone is $\leq 8n$.

Proof by induction, counting the left and the right edges of the faces in the zone of h (WLG, it can be assumed there are no horizontal lines in L).

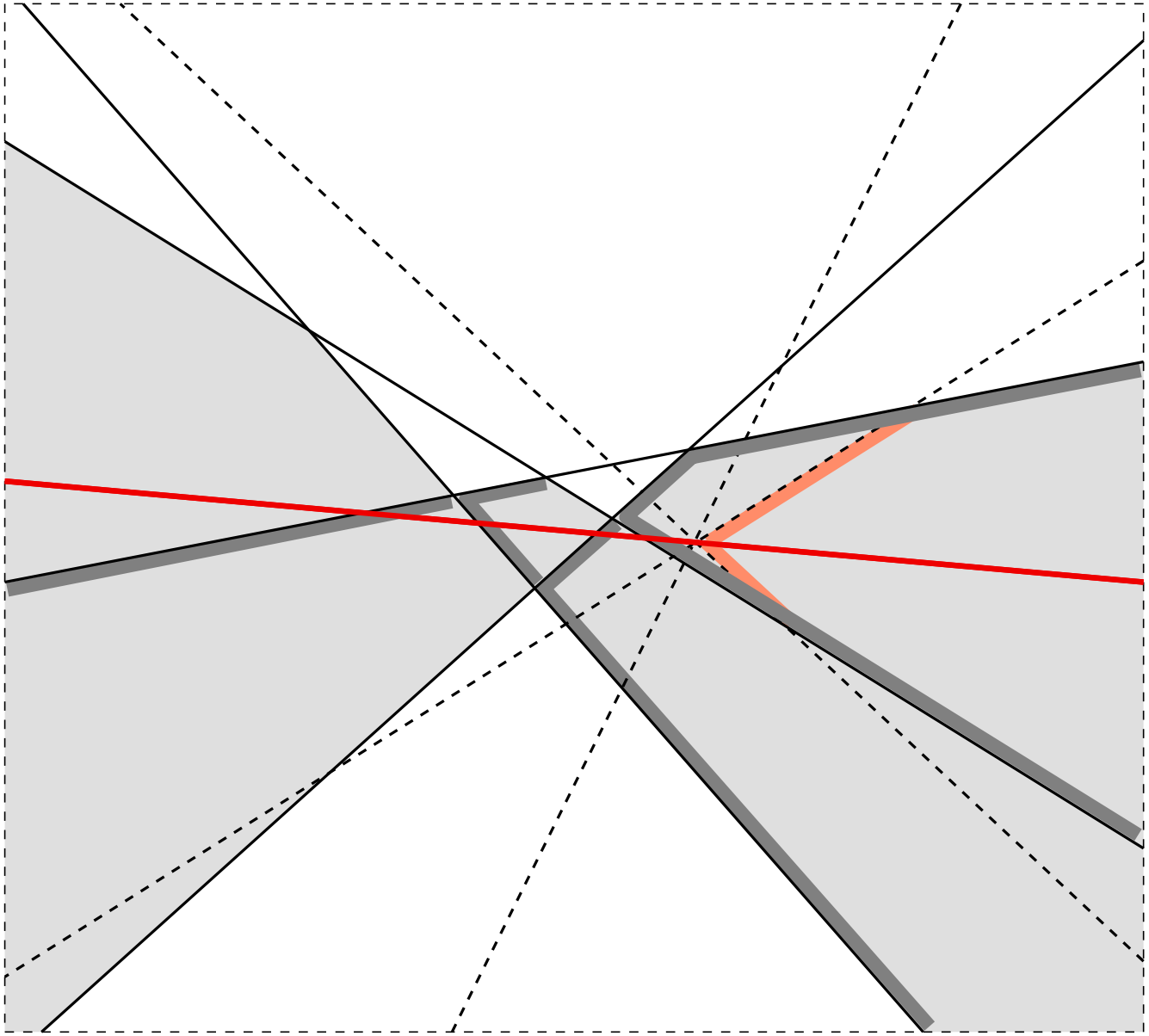


THE ZONE THEOREM

Definition: Given an arrangement $\mathcal{A}(L)$ and a line h, the *zone* of h is the set of faces of the arrangement intersected by h.

Theorem: The complexity of the zone of a line h in an arrangement of n lines is $O(n)$. More precisely, the total number of edges of the faces of the zone is $\leq 8n$.

Proof by induction, counting the left and the right edges of the faces in the zone of h (WLG, it can be assumed there are no horizontal lines in L).

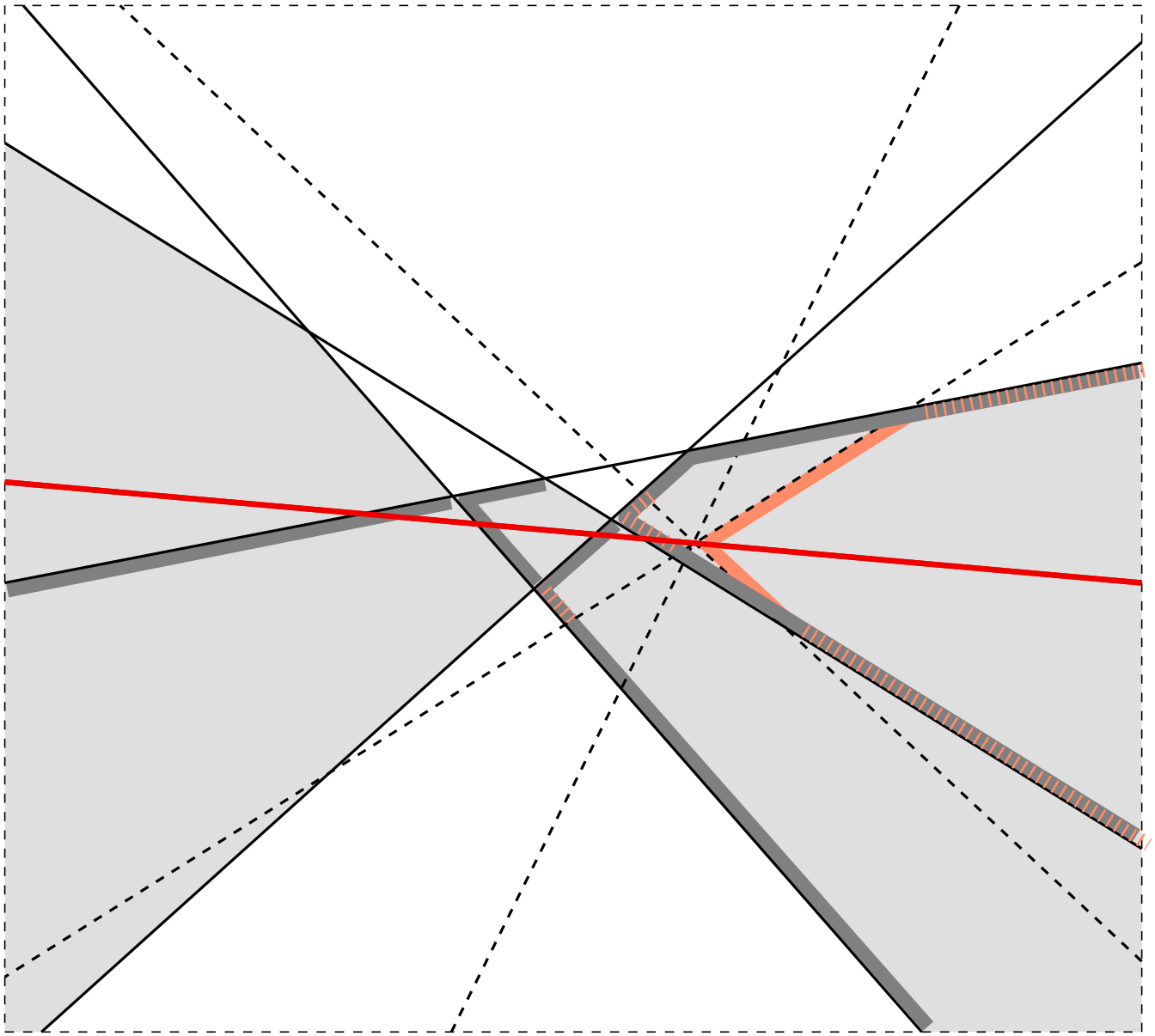


THE ZONE THEOREM

Definition: Given an arrangement $\mathcal{A}(L)$ and a line h, the *zone* of h is the set of faces of the arrangement intersected by h.

Theorem: The complexity of the zone of a line h in an arrangement of n lines is $O(n)$. More precisely, the total number of edges of the faces of the zone is $\leq 8n$.

Proof by induction, counting the left and the right edges of the faces in the zone of h (WLG, it can be assumed there are no horizontal lines in L).

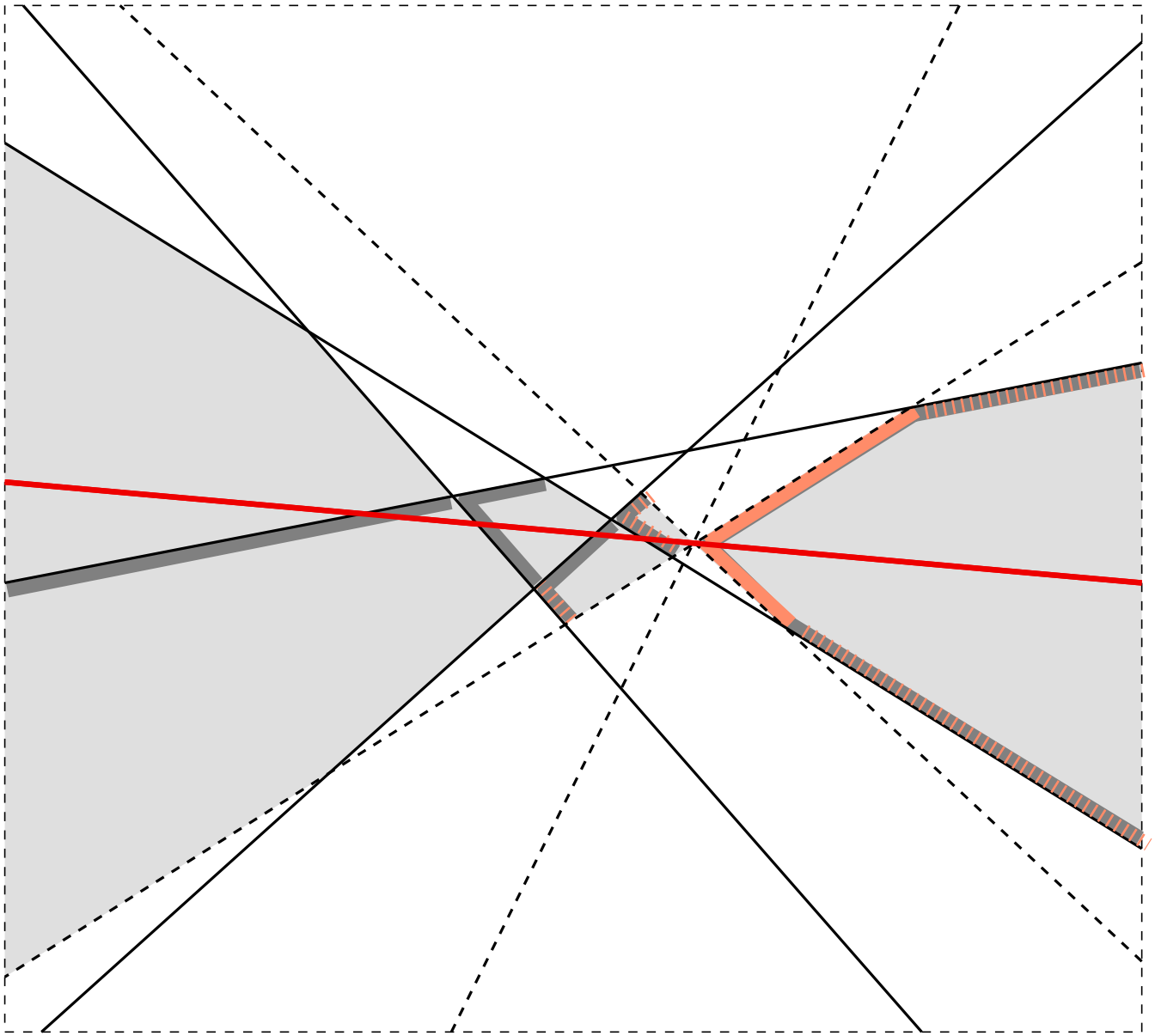


THE ZONE THEOREM

Definition: Given an arrangement $\mathcal{A}(L)$ and a line h, the *zone* of h is the set of faces of the arrangement intersected by h.

Theorem: The complexity of the zone of a line h in an arrangement of n lines is $O(n)$. More precisely, the total number of edges of the faces of the zone is $\leq 8n$.

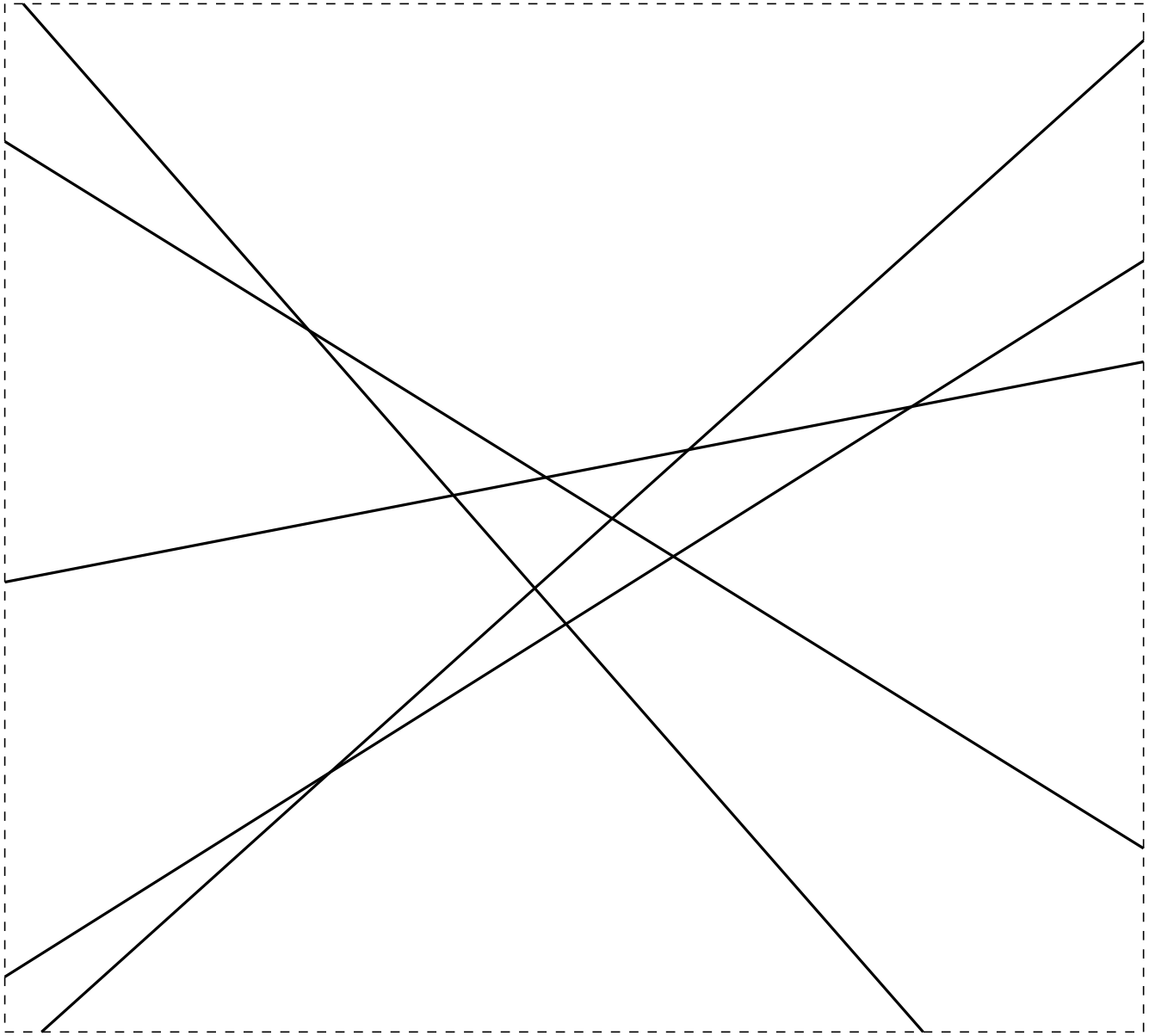
Proof by induction, counting the left and the right edges of the faces in the zone of h (WLG, it can be assumed there are no horizontal lines in L).



LEVELS

Definition: The kth *level* of an arrangement $\mathcal{A}(L)$ without vertical lines is the set of points p of the plane such that:

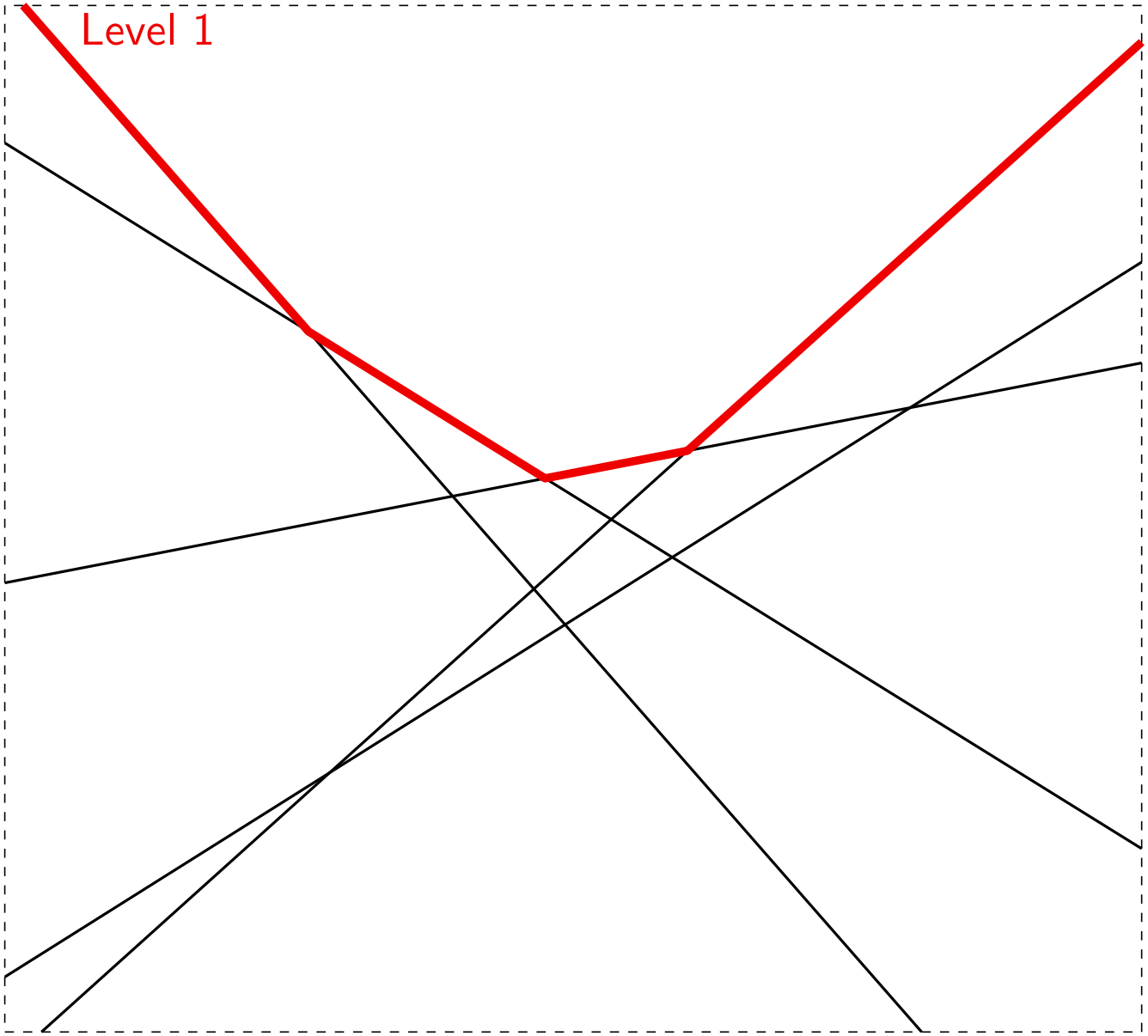
- The number of lines above p is $\leq k-1$.
- The number of lines below p is $\leq n-k$.



LEVELS

Definition: The kth *level* of an arrangement $\mathcal{A}(L)$ without vertical lines is the set of points p of the plane such that:

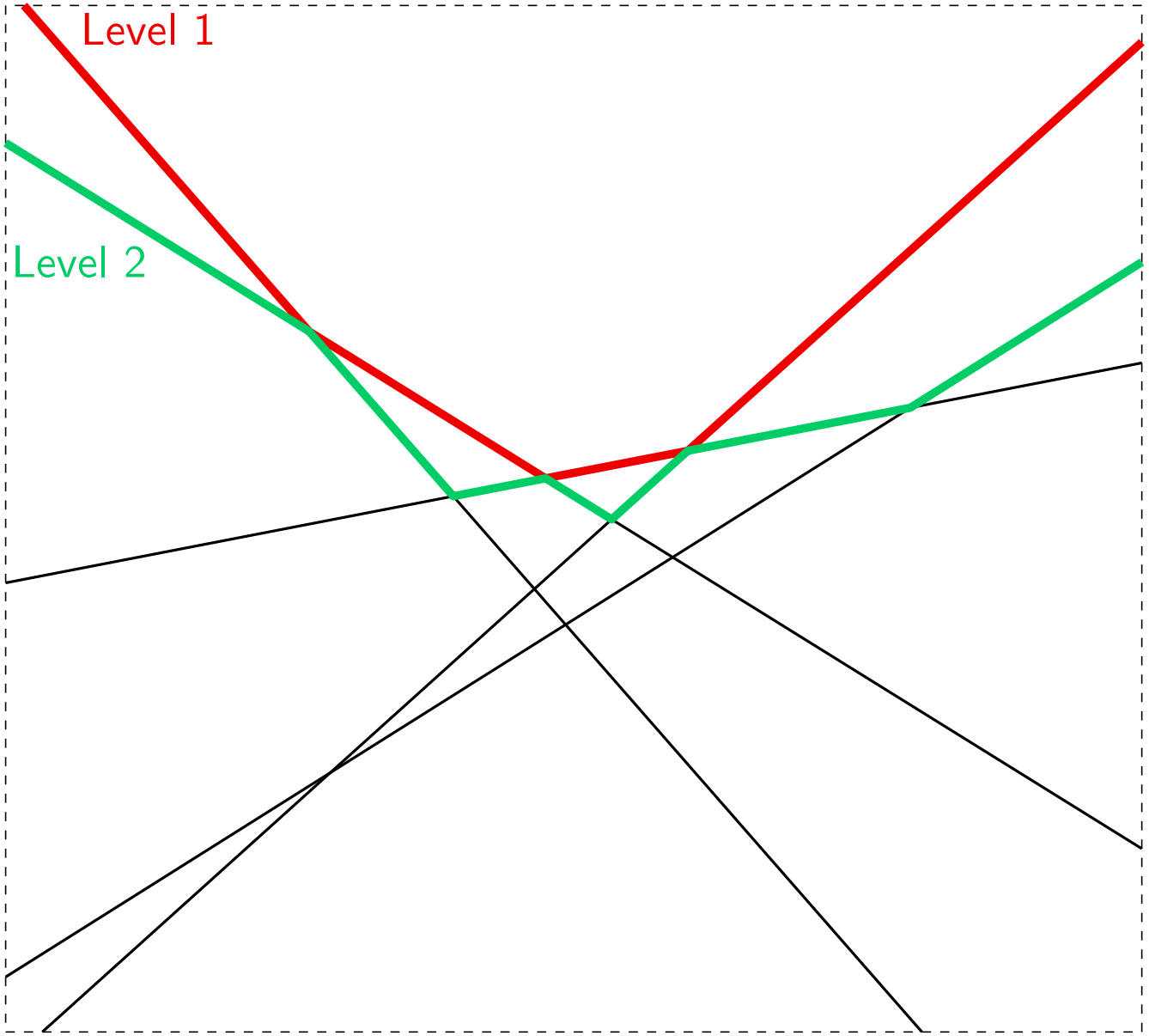
- The number of lines above p is $\leq k-1$.
- The number of lines below p is $\leq n-k$.



LEVELS

Definition: The kth *level* of an arrangement $\mathcal{A}(L)$ without vertical lines is the set of points p of the plane such that:

- The number of lines above p is $\leq k-1$.
- The number of lines below p is $\leq n-k$.

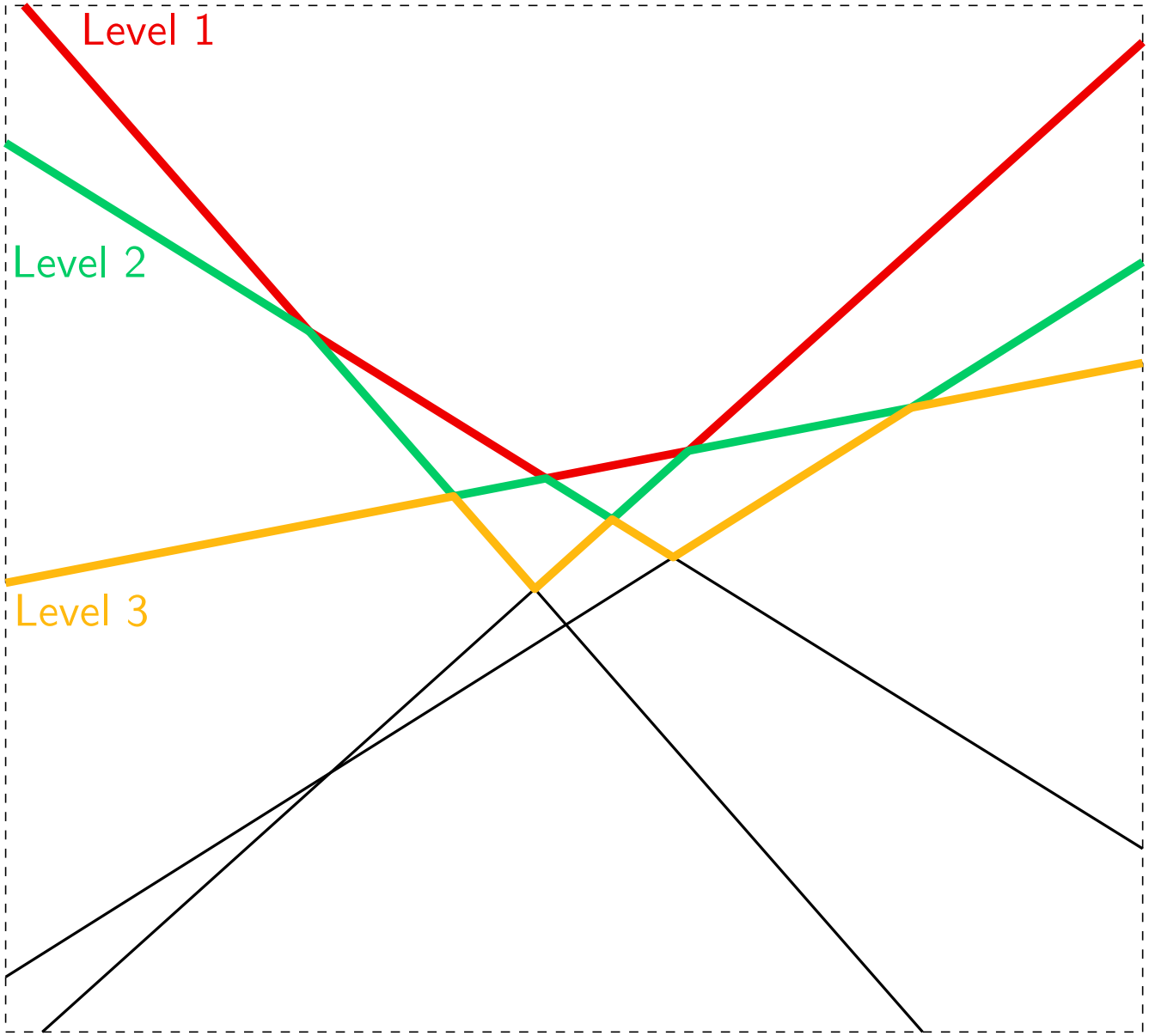


ARRANGEMENTS

LEVELS

Definition: The kth *level* of an arrangement $\mathcal{A}(L)$ without vertical lines is the set of points p of the plane such that:

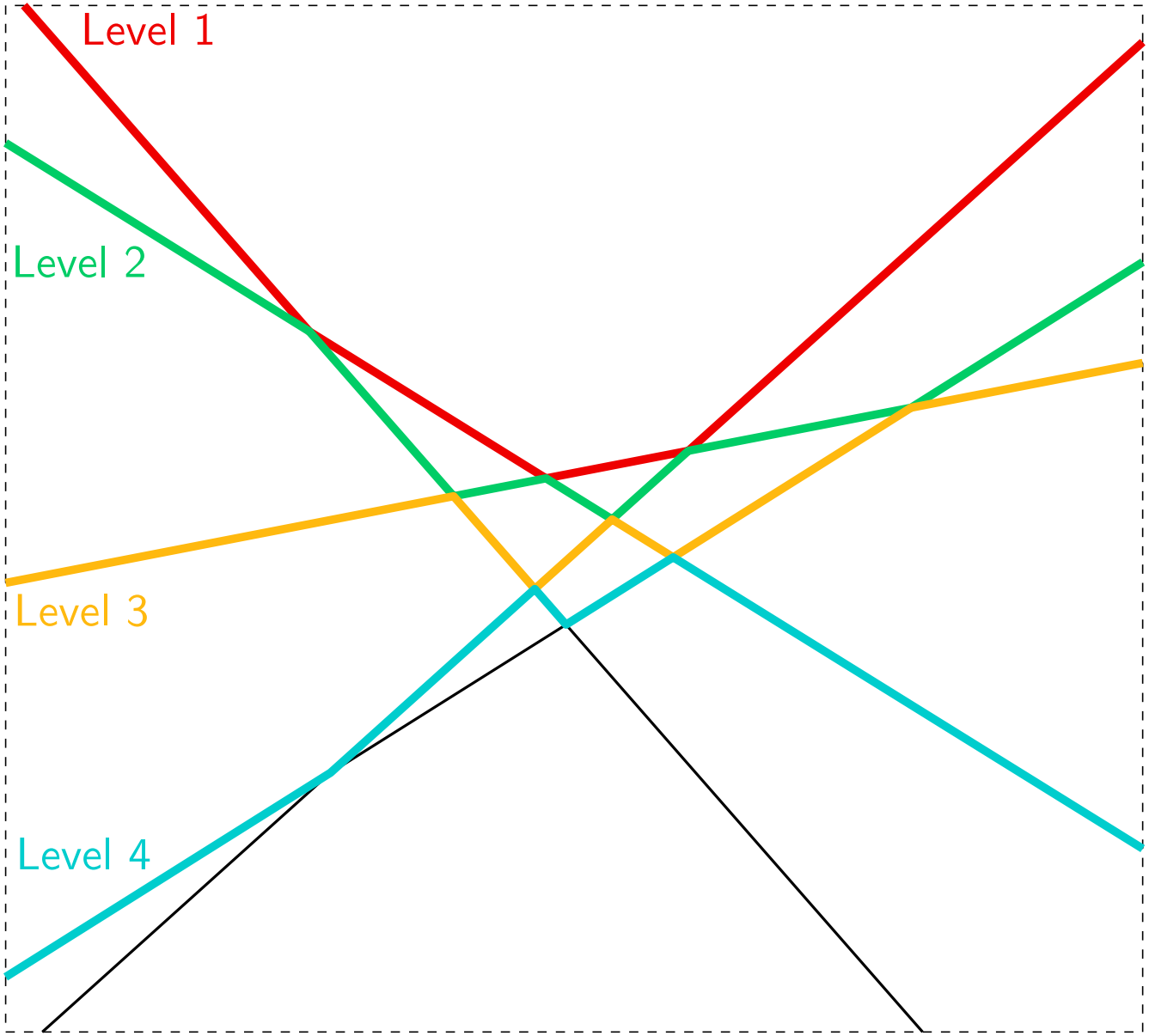
- The number of lines above p is $\leq k-1$.
- The number of lines below p is $\leq n-k$.



LEVELS

Definition: The kth *level* of an arrangement $\mathcal{A}(L)$ without vertical lines is the set of points p of the plane such that:

- The number of lines above p is $\leq k-1$.
- The number of lines below p is $\leq n-k$.

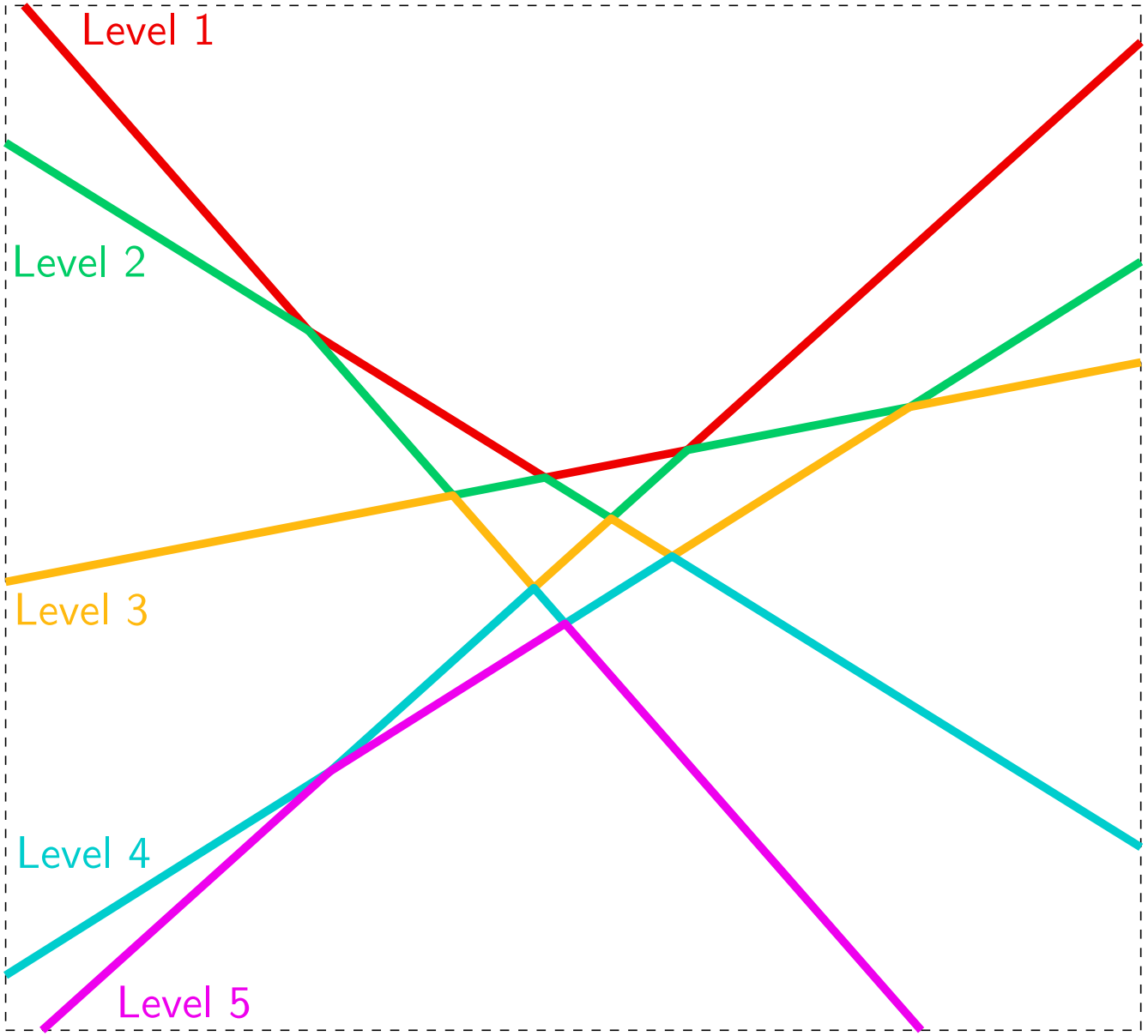


ARRANGEMENTS

LEVELS

Definition: The kth *level* of an arrangement $\mathcal{A}(L)$ without vertical lines is the set of points p of the plane such that:

- The number of lines above p is $\leq k-1$.
- The number of lines below p is $\leq n-k$.



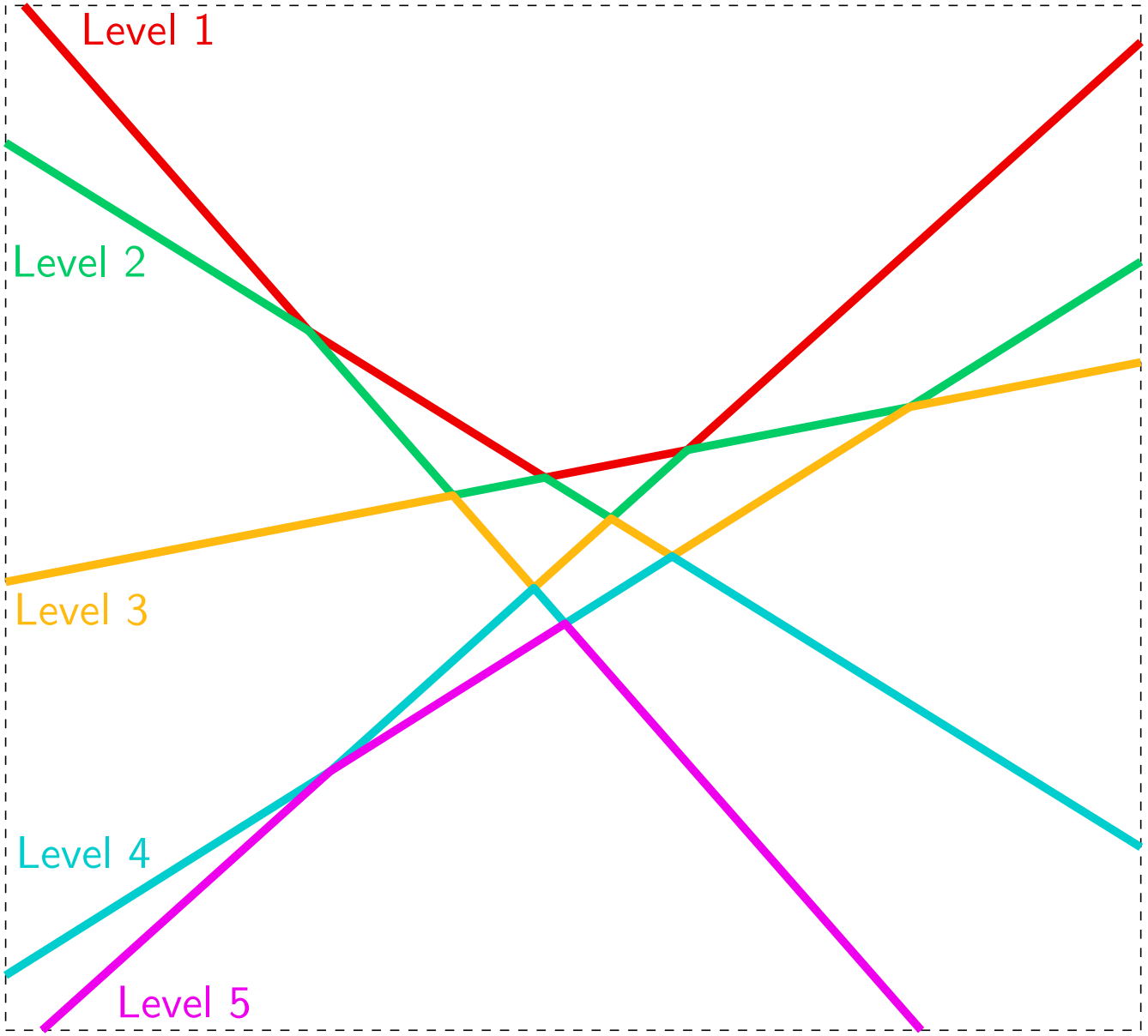
LEVELS

Properties

1. A point p belongs to one (more than one) level if and only if it belongs to one (more than one) of the lines.

2. Any vertical line intersects each level in a single point.

3. If a vertical line intersects two levels, $i \leq j$ in two points p_i and p_j, then p_i is above p_j. Consequently, the levels are totally ordered.



ARRANGEMENTS AND VORONOI DIAGRAMS

All previous definitions and properties extend to arrangements of hyperplanes in $\mathbb{R}^d$ (simplicity, combinatorics, levels,...), but the complexity of the arrangement depends on the dimension.

ARRANGEMENTS AND VORONOI DIAGRAMS

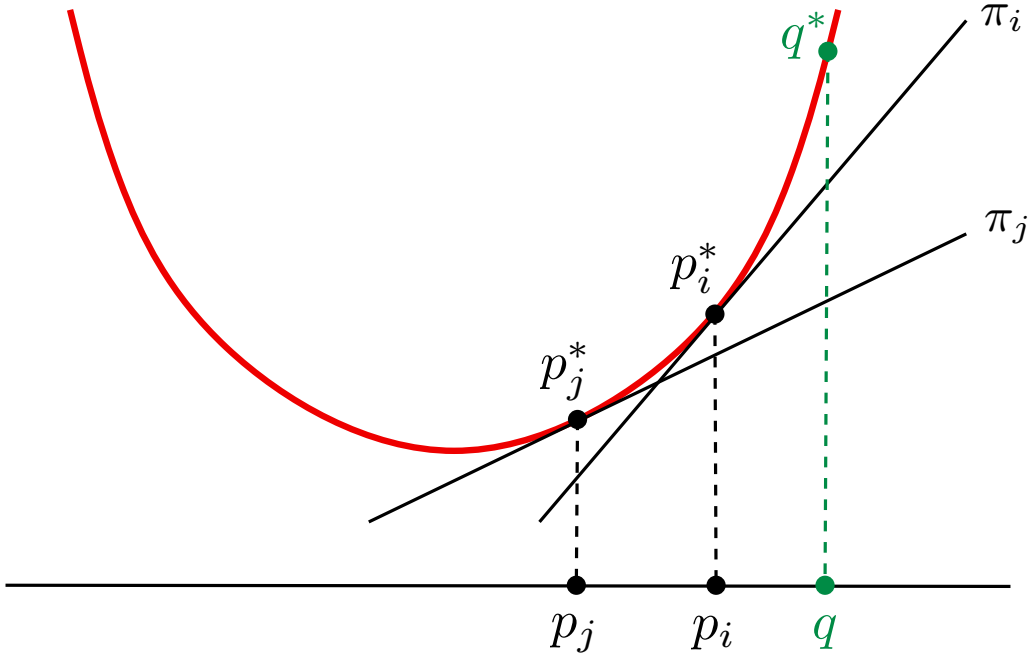
All previous definitions and properties extend to arrangements of hyperplanes in $\mathbb{R}^d$ (simplicity, combinatorics, levels,...), but the complexity of the arrangement depends on the dimension.

Proposition: Let $P = \{p_1, \ldots, p_n\}$ be a finite set of points $p_i = (a_i, b_i, 0)$. Let $H = \{h_1, \ldots, h_n\}$ be the set of planes which are tangent to the paraboloid $z = x^2 + y^2$ at the points $p_i^* = (a_i, b_i, a_i^2 + b_i^2)$. The 1-skeleton of the kth-order Voronoi diagram of P is the orthogonal projection onto the plane $z = 0$ of the intersection of the levels k and $k + 1$ of $\mathcal{A}(H)$.

ARRANGEMENTS AND VORONOI DIAGRAMS

All previous definitions and properties extend to arrangements of hyperplanes in $\mathbb{R}^d$ (simplicity, combinatorics, levels,...), but the complexity of the arrangement depends on the dimension.

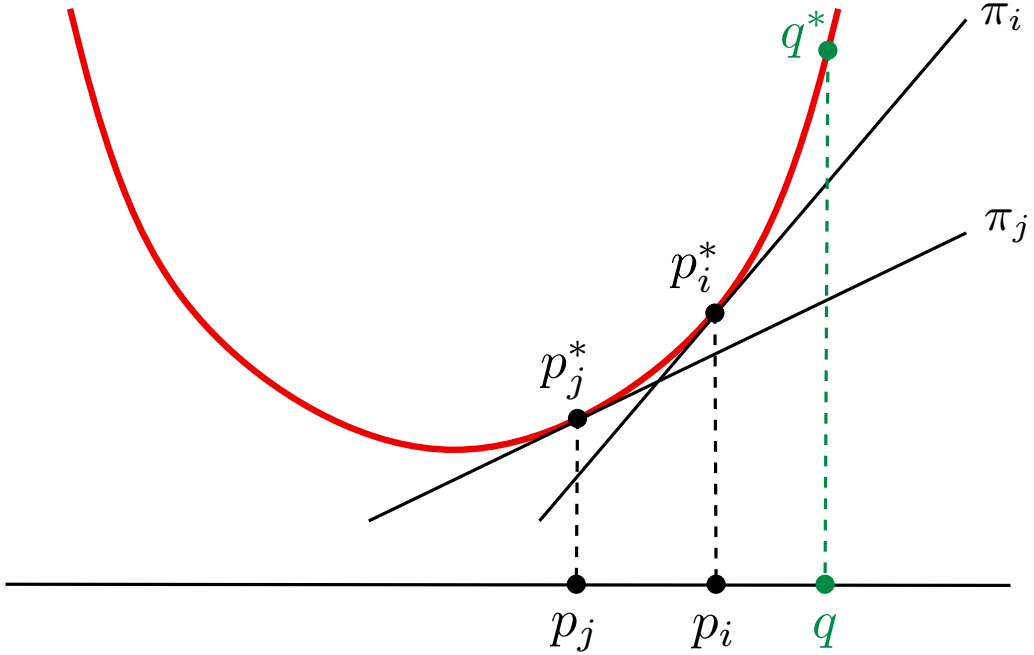
Proposition: Let $P = \{p_1, \ldots, p_n\}$ be a finite set of points $p_i = (a_i, b_i, 0)$. Let $H = \{h_1, \ldots, h_n\}$ be the set of planes which are tangent to the paraboloid $z = x^2 + y^2$ at the points $p_i^* = (a_i, b_i, a_i^2 + b_i^2)$. The 1-skeleton of the kth-order Voronoi diagram of P is the orthogonal projection onto the plane $z = 0$ of the intersection of the levels k and $k+1$ of $\mathcal{A}(H)$.

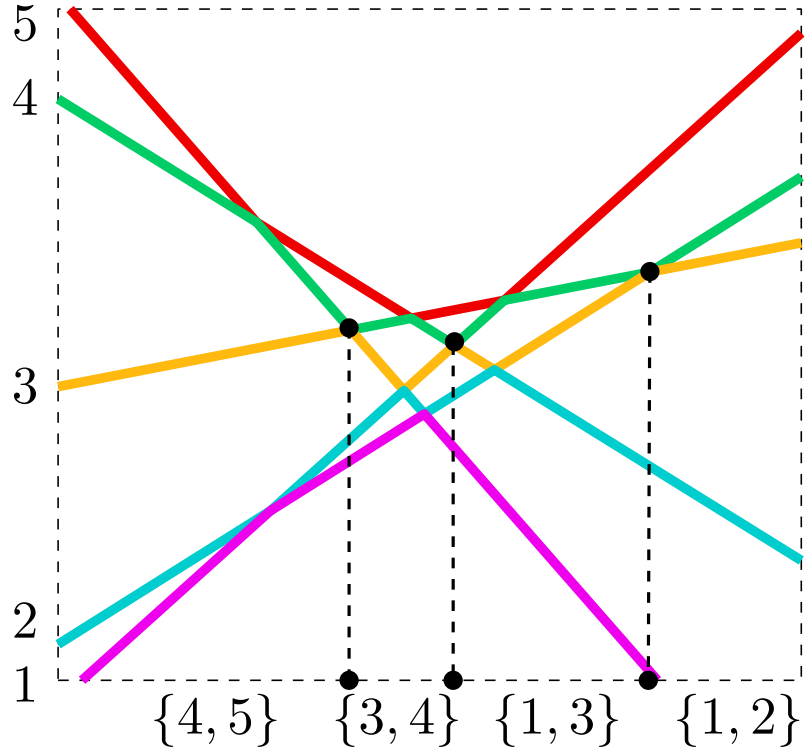


ARRANGEMENTS AND VORONOI DIAGRAMS

All previous definitions and properties extend to arrangements of hyperplanes in $\mathbb{R}^d$ (simplicity, combinatorics, levels,...), but the complexity of the arrangement depends on the dimension.

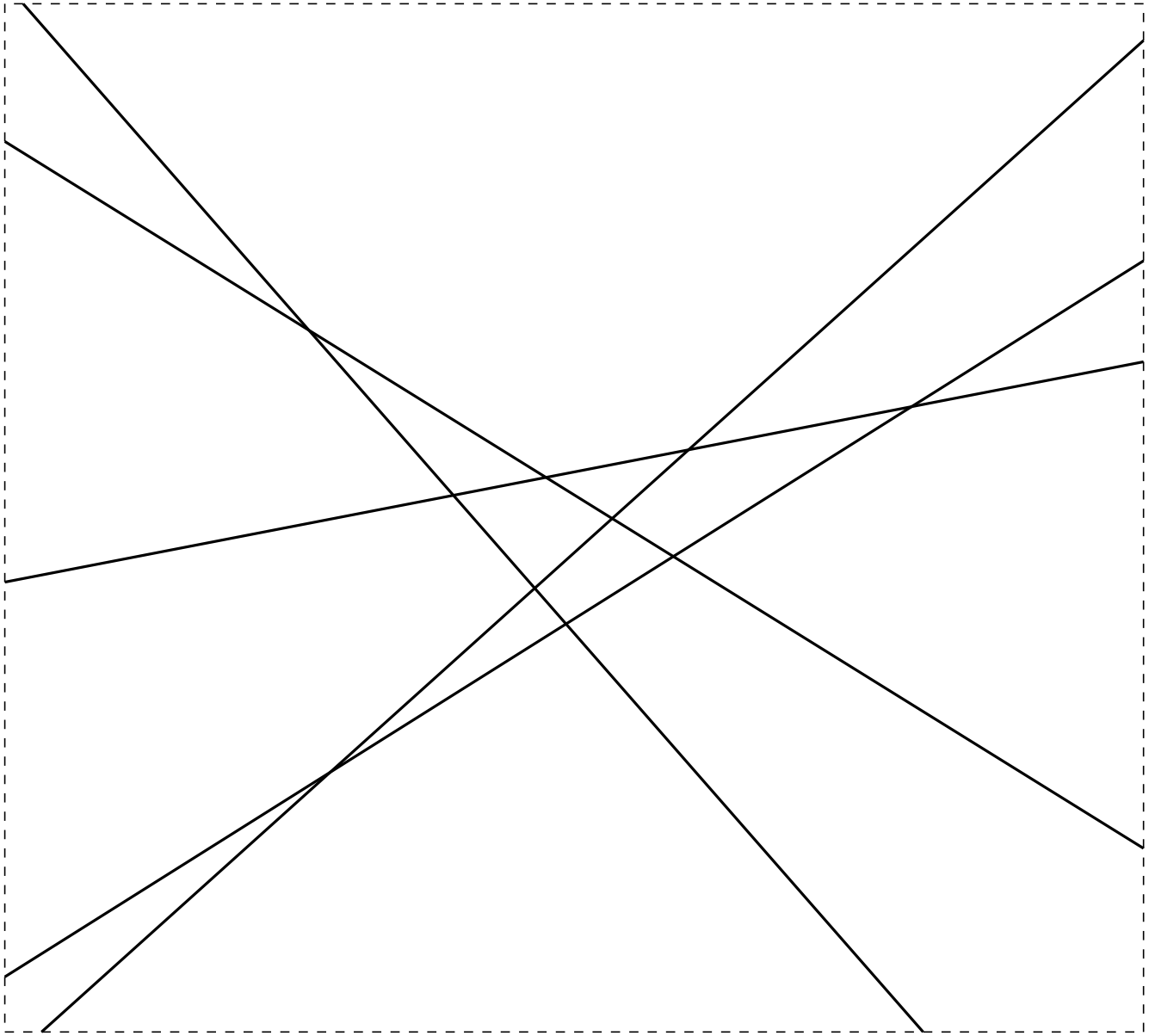
Proposition: Let $P = \{p_1, \ldots, p_n\}$ be a finite set of points $p_i = (a_i, b_i, 0)$. Let $H = \{h_1, \ldots, h_n\}$ be the set of planes which are tangent to the paraboloid $z = x^2 + y^2$ at the points $p_i^* = (a_i, b_i, a_i^2 + b_i^2)$. The 1-skeleton of the kth-order Voronoi diagram of P is the orthogonal projection onto the plane $z = 0$ of the intersection of the levels k and $k+1$ of $\mathcal{A}(H)$.





STORAGE

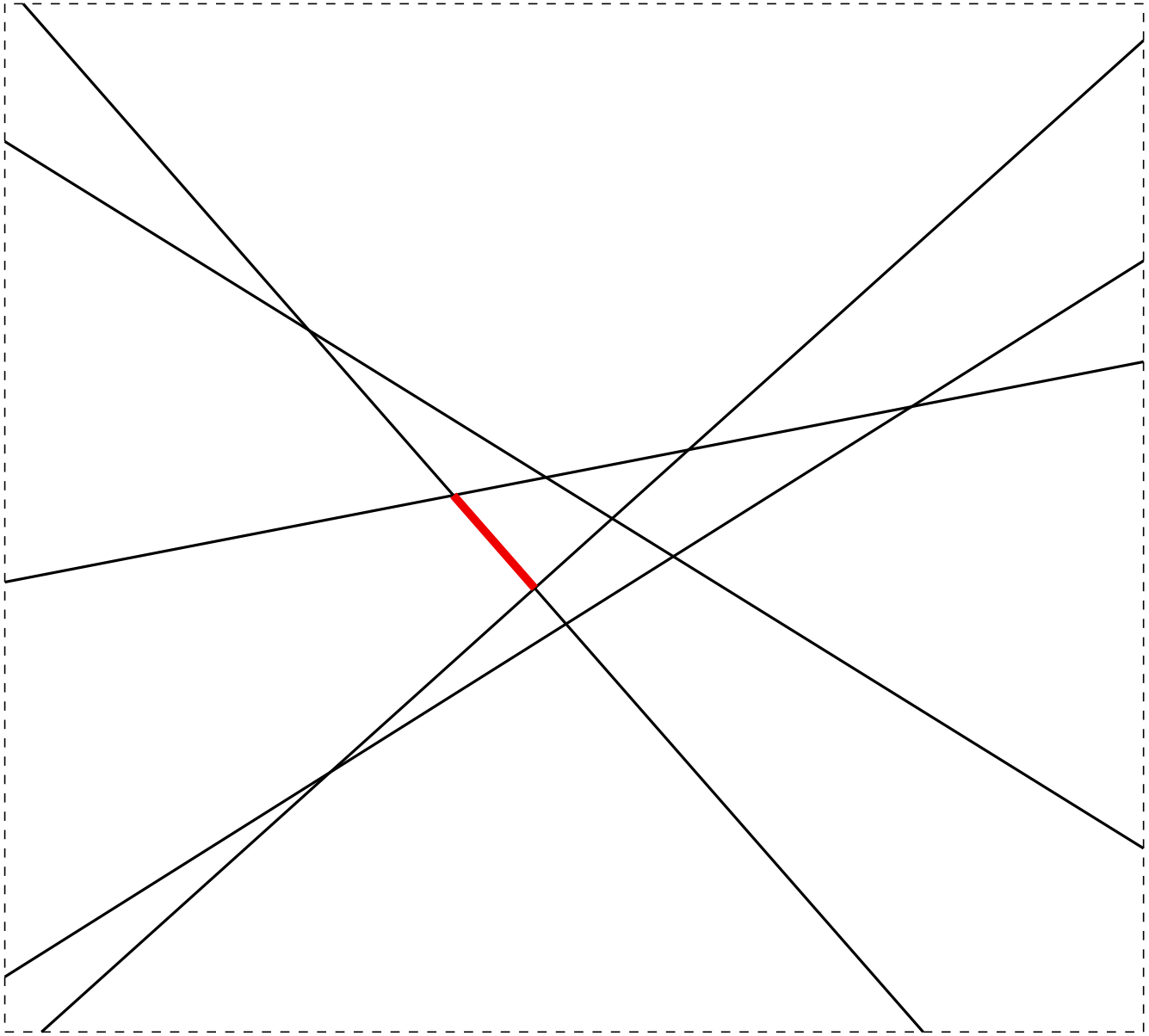
As an arragements of lines is a decomposition of the plane, the structure usually used to store it is a DCEL.



STORAGE

As an arragements of lines is a decomposition of the plane, the structure usually used to store it is a DCEL.

e

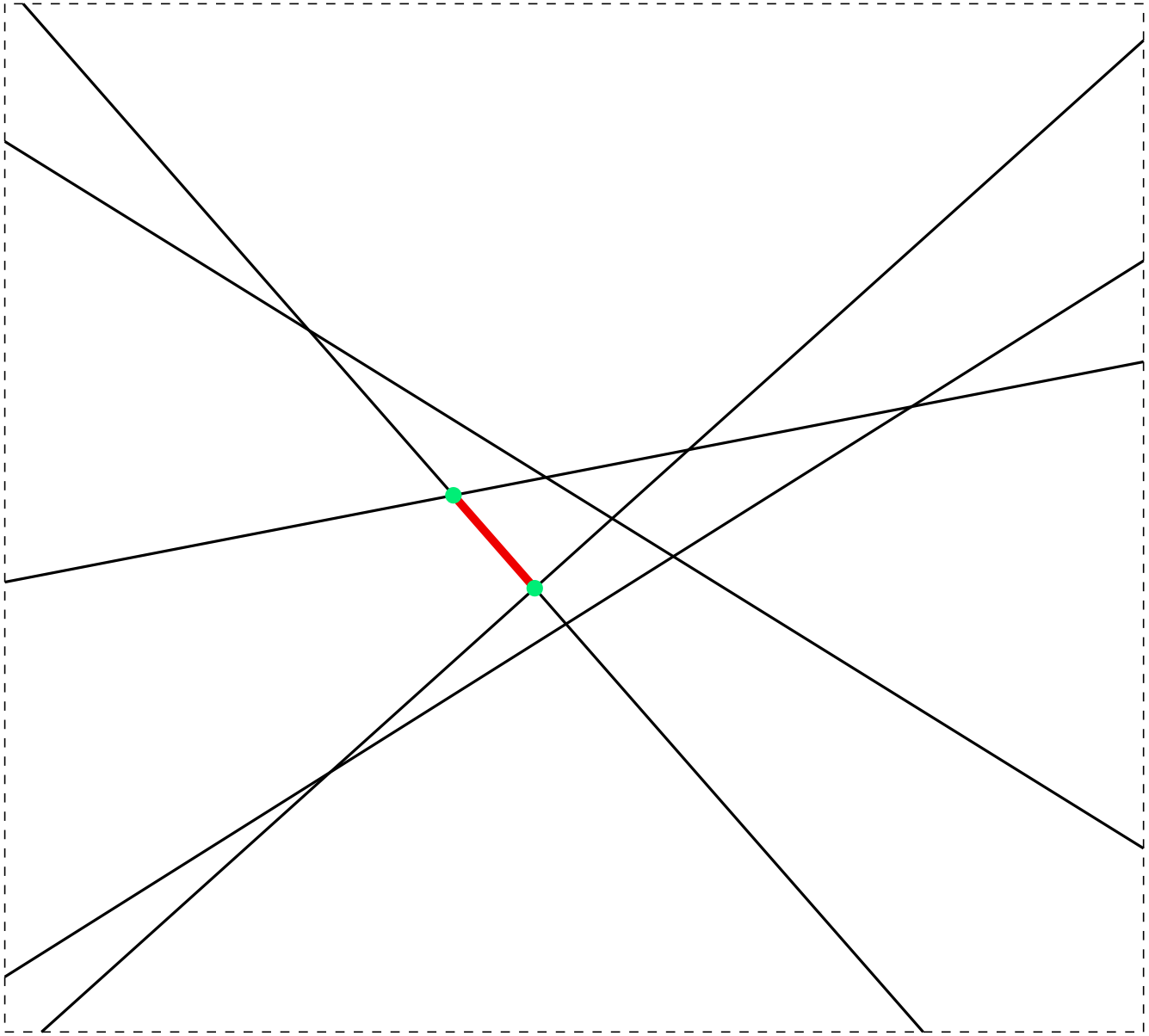


ARRANGEMENTS

STORAGE

As an arragements of lines is a decomposition of the plane, the structure usually used to store it is a DCEL.

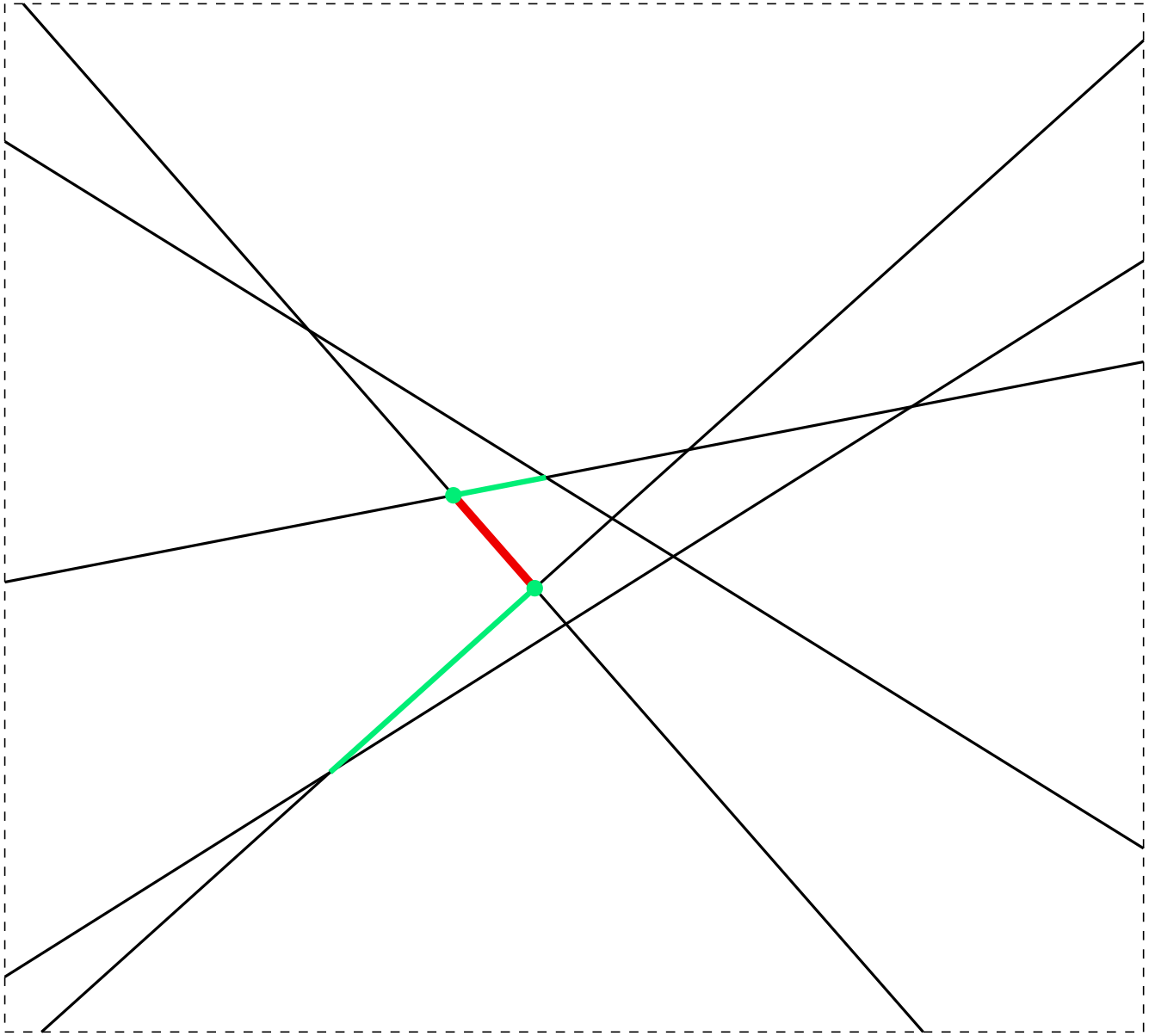
$e \rightarrow v_B \, v_E$



STORAGE

As an arragements of lines is a decomposition of the plane, the structure usually used to store it is a DCEL.

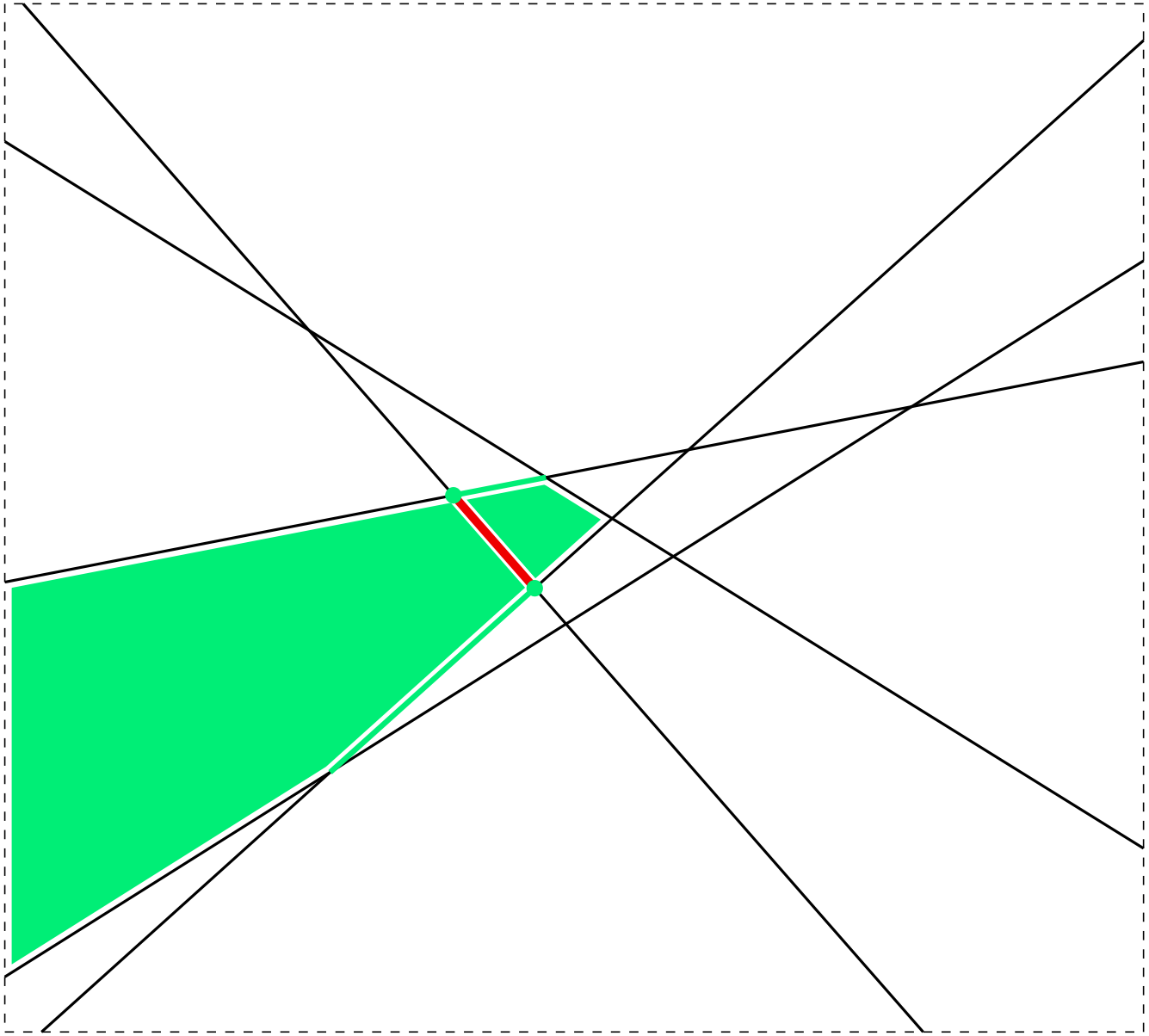
$e \rightarrow v_B\, v_E\ \ e_P\ e_N$



STORAGE

As an arragements of lines is a decomposition of the plane, the structure usually used to store it is a DCEL.

$e \rightarrow v_B \; v_E \;\; e_P \; e_N \;\; f_L \;\; f_R$



STORAGE

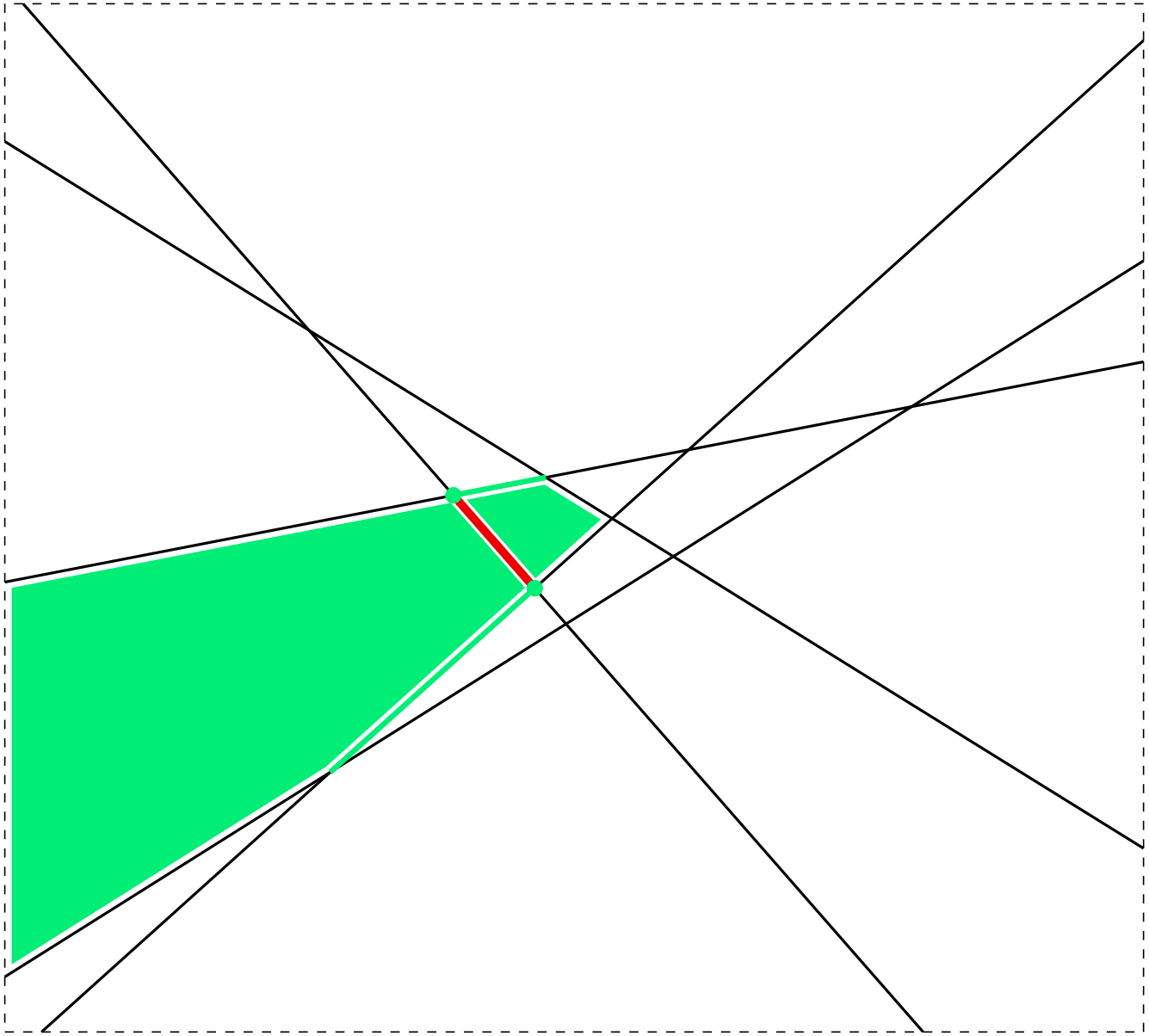
As an arragements of lines is a decomposition of the plane, the structure usually used to store it is a DCEL.

$e \rightarrow v_B\, v_E\ \ e_P\ e_N\ \ f_L\ \ f_R$

$v_i \rightarrow x_i, y_i, e$

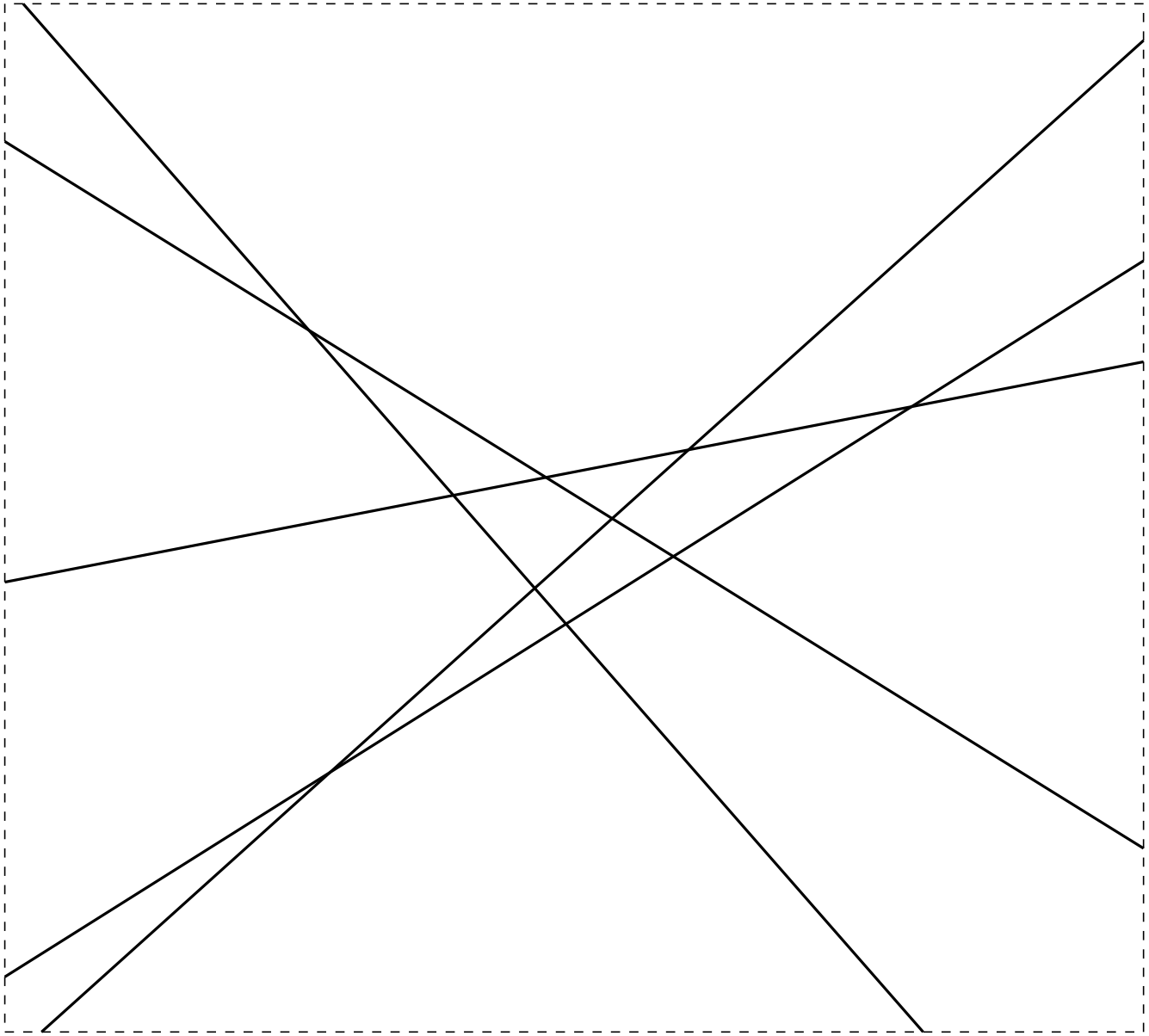
$f_j \rightarrow e$

(Recall that halflines require a specific treatment)



ARRANGEMENTS

COMPUTATION



ARRANGEMENTS

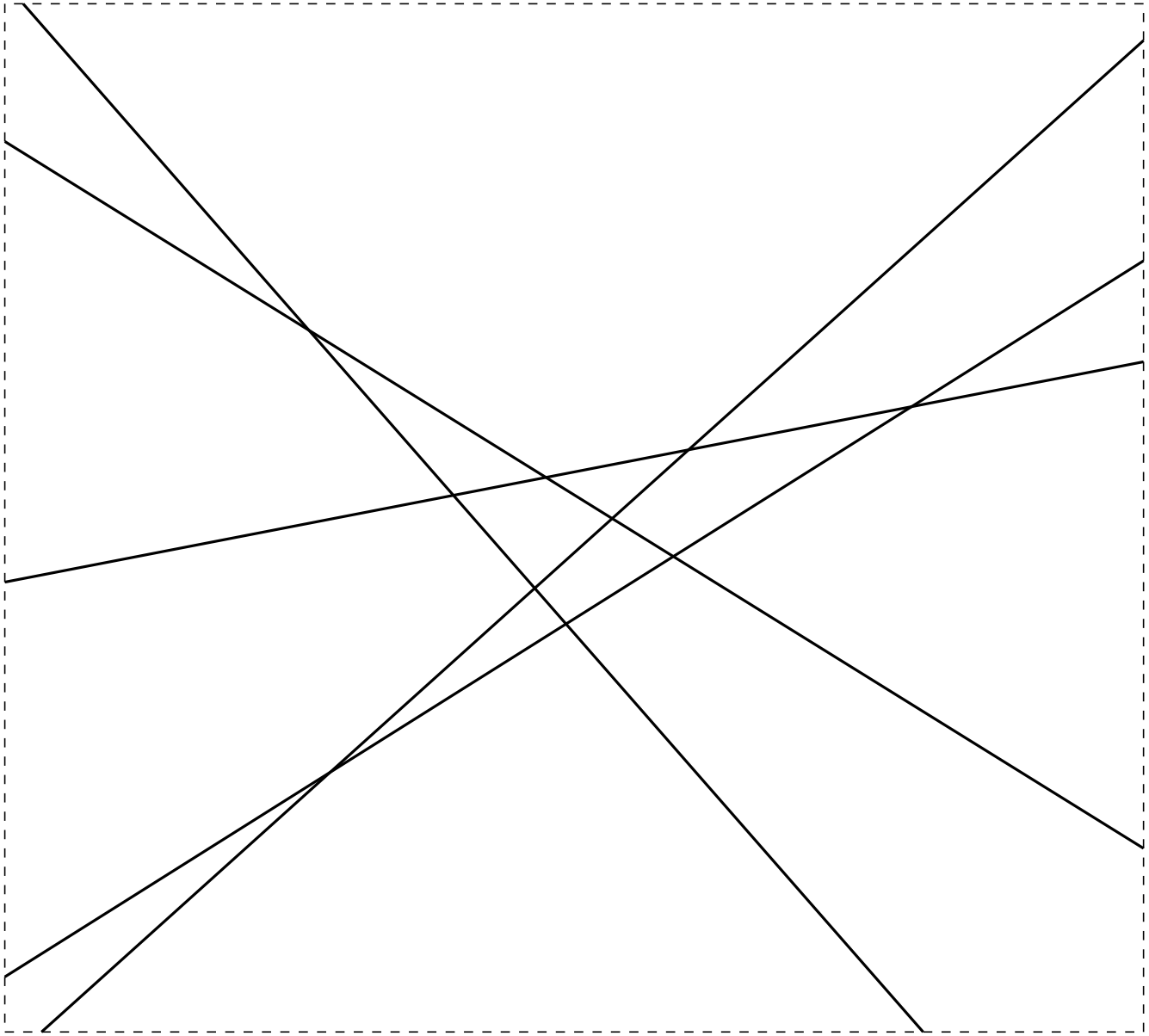
COMPUTATION

Input

A set of n lines $L = \{l_1, \ldots, l_n\}$

Output

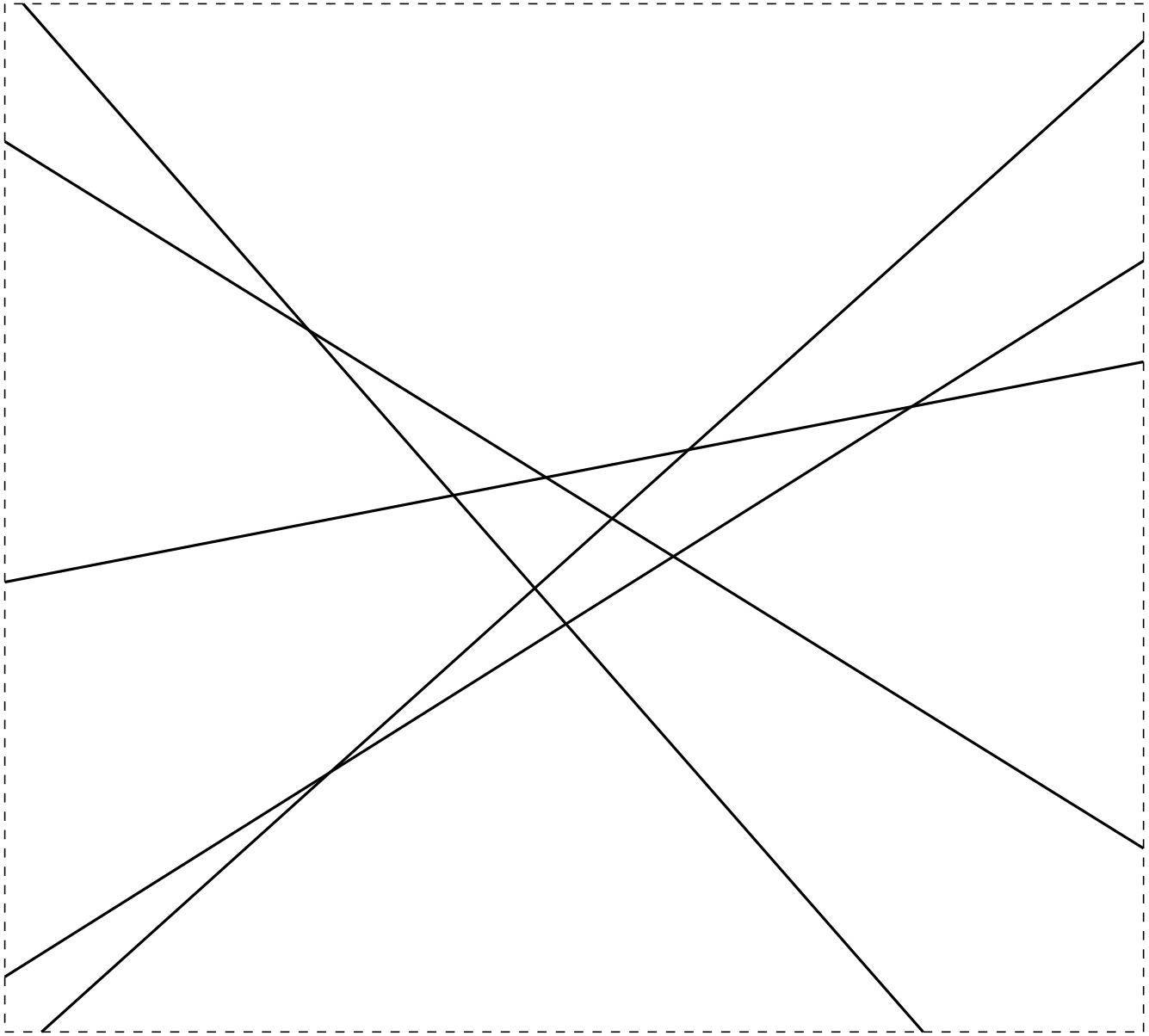
The arrangement $\mathcal{A}(L)$ i.e.,
the DCEL of the arrangement.



ARRANGEMENTS

COMPUTATION

Sweep line algorithm

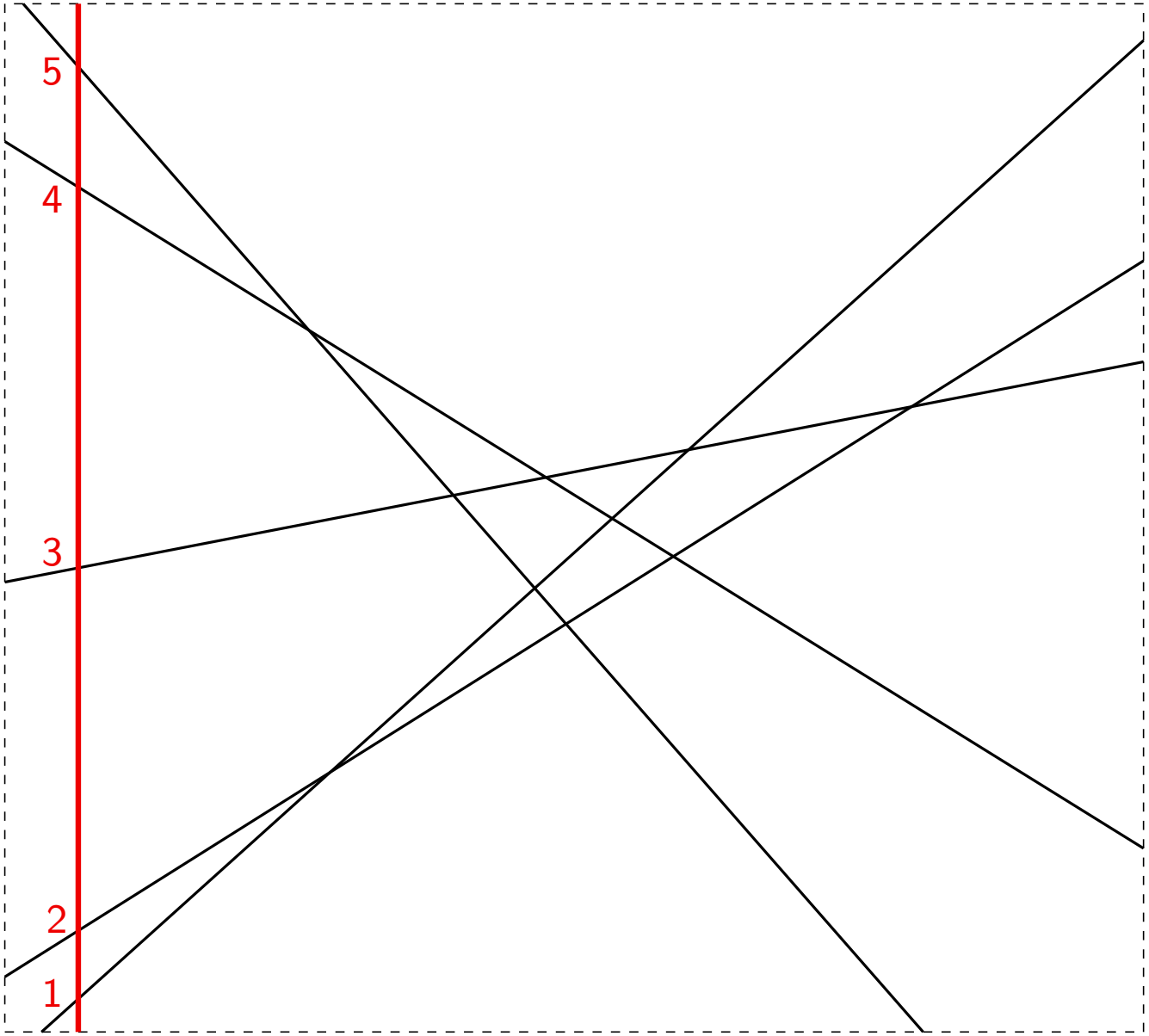


COMPUTATION

Sweep line algorithm

Sweep line

Sorted list of the stabbed lines, in vertical order.



COMPUTATION

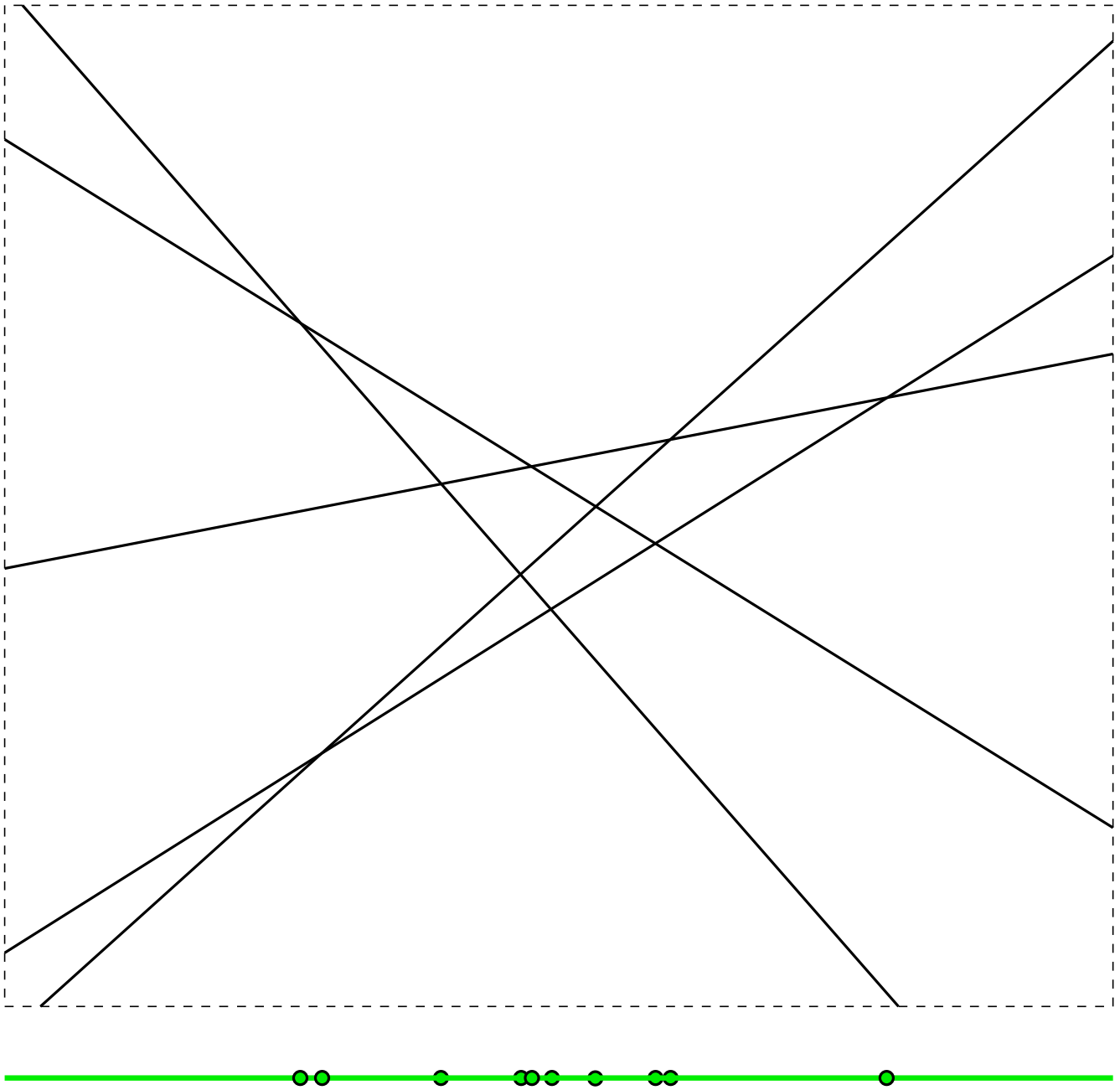
Sweep line algorithm

Sweep line

Sorted list of the stabbed lines, in vertical order.

Events queue

Sorted list of the intersection points, in horizontal order. Not known a priori, they are found on the fly.

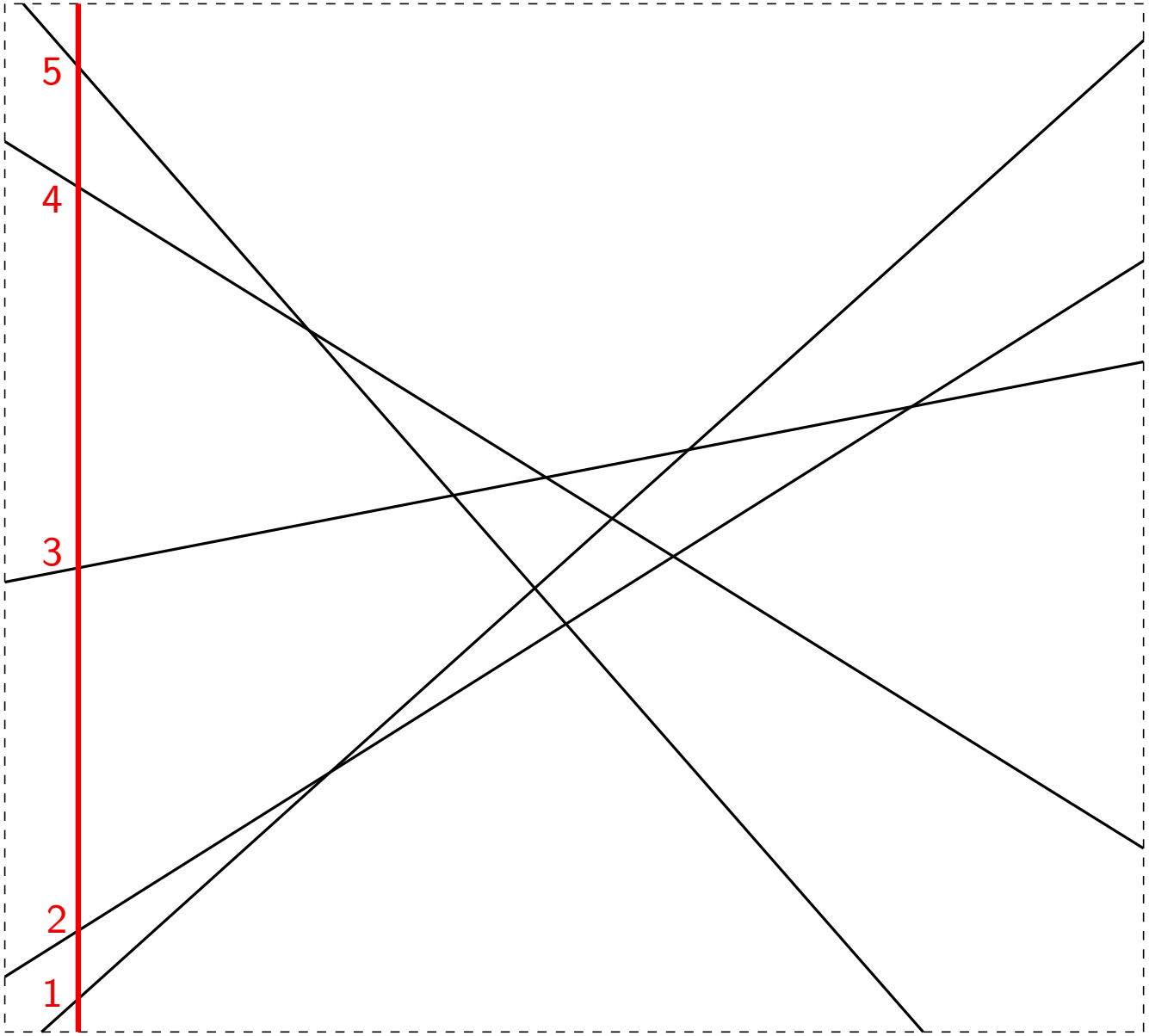


COMPUTATION

Sweep line algorithm

Initialization:

- Sort the lines by slope and store them in the sweep line.

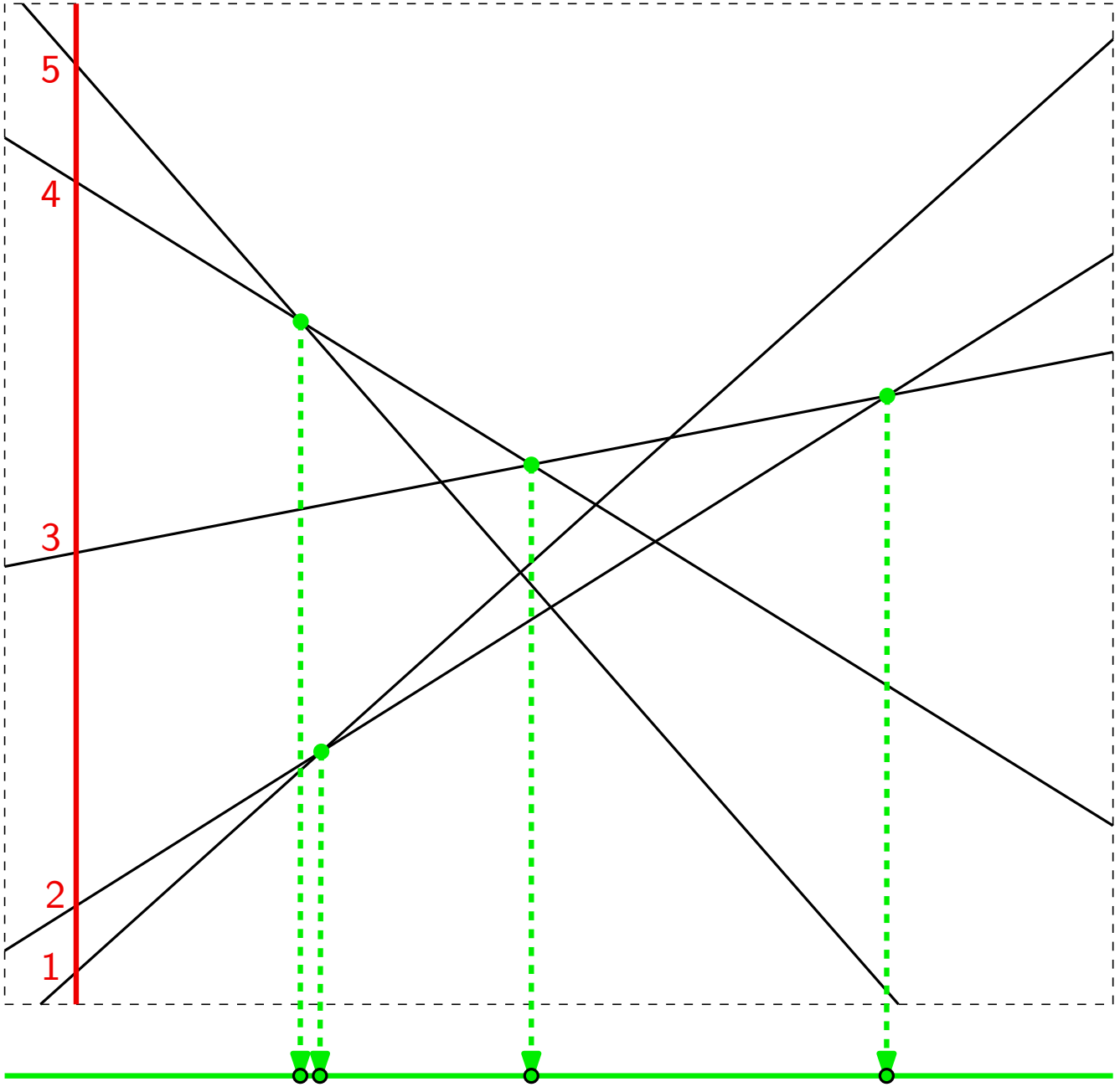


COMPUTATION

Sweep line algorithm

Initialization:

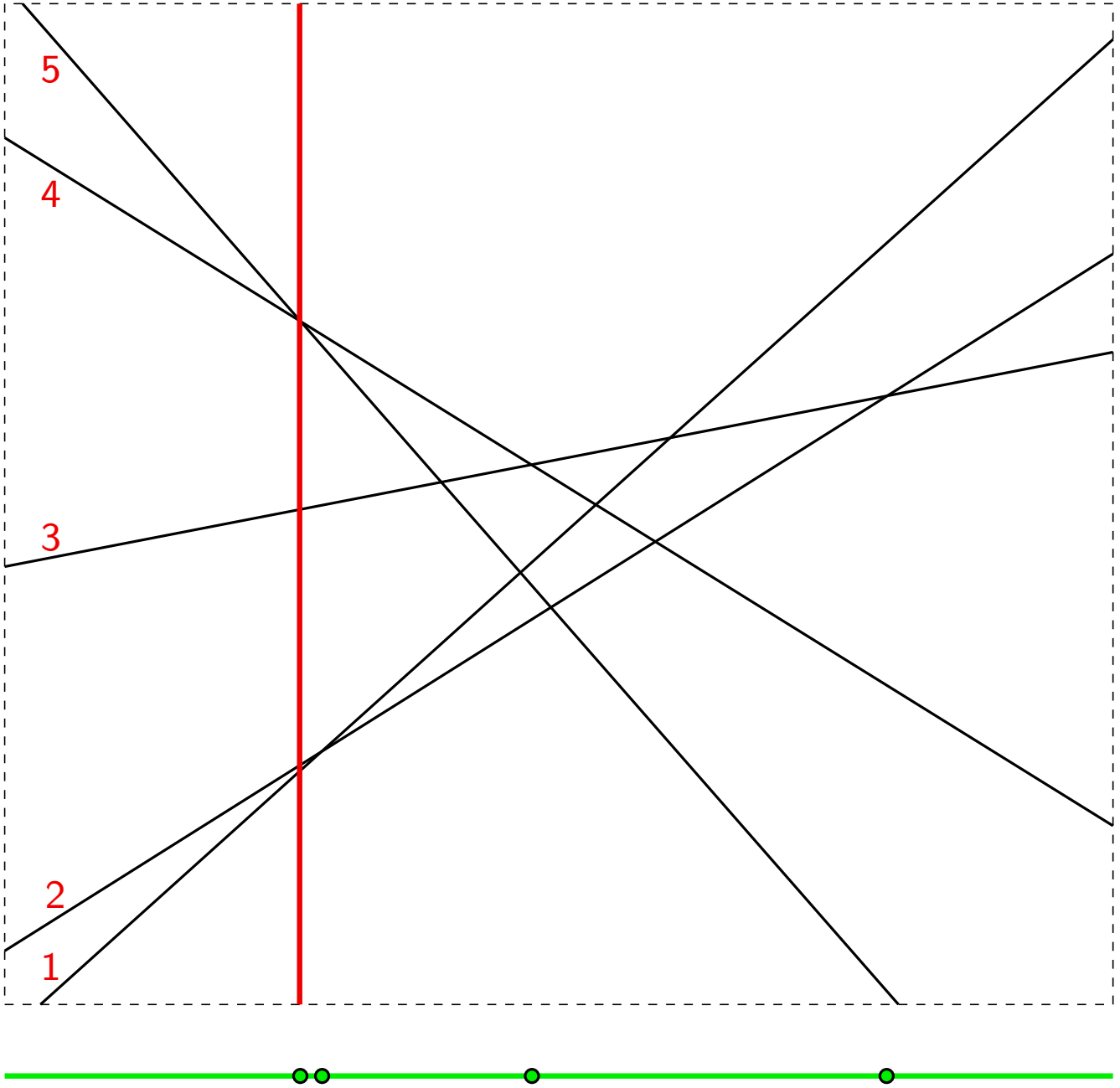
- Sort the lines by slope and store them in the sweep line.
- Detect the intersection point of each pair of consecutive lines, sort them by abscissa and store them in the events queue.



COMPUTATION

Sweep line algorithm

Advance: at each event,

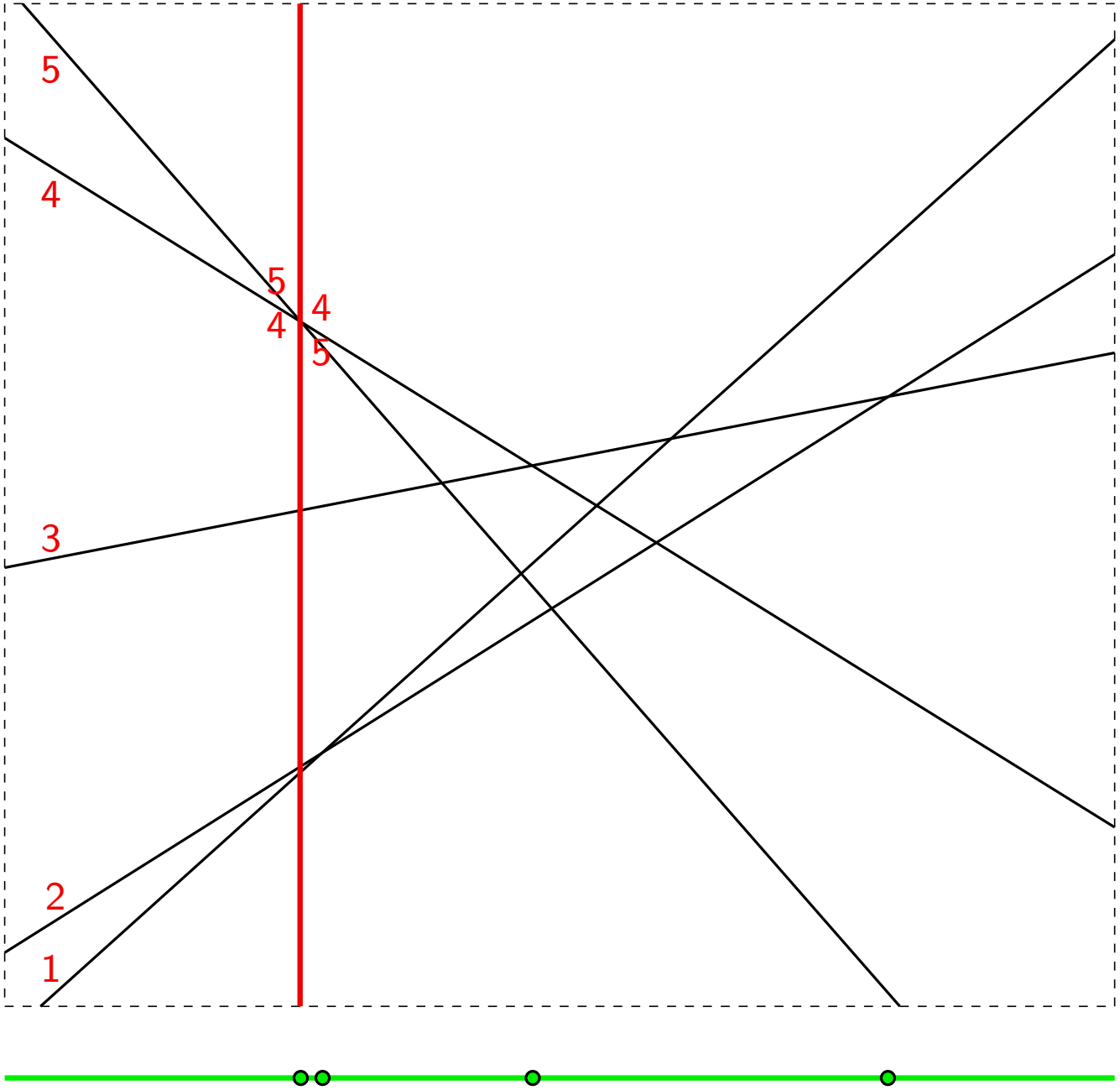


COMPUTATION

Sweep line algorithm

Advance: at each event,

1. Swap the order of the two intersecting lines in the sweep line.

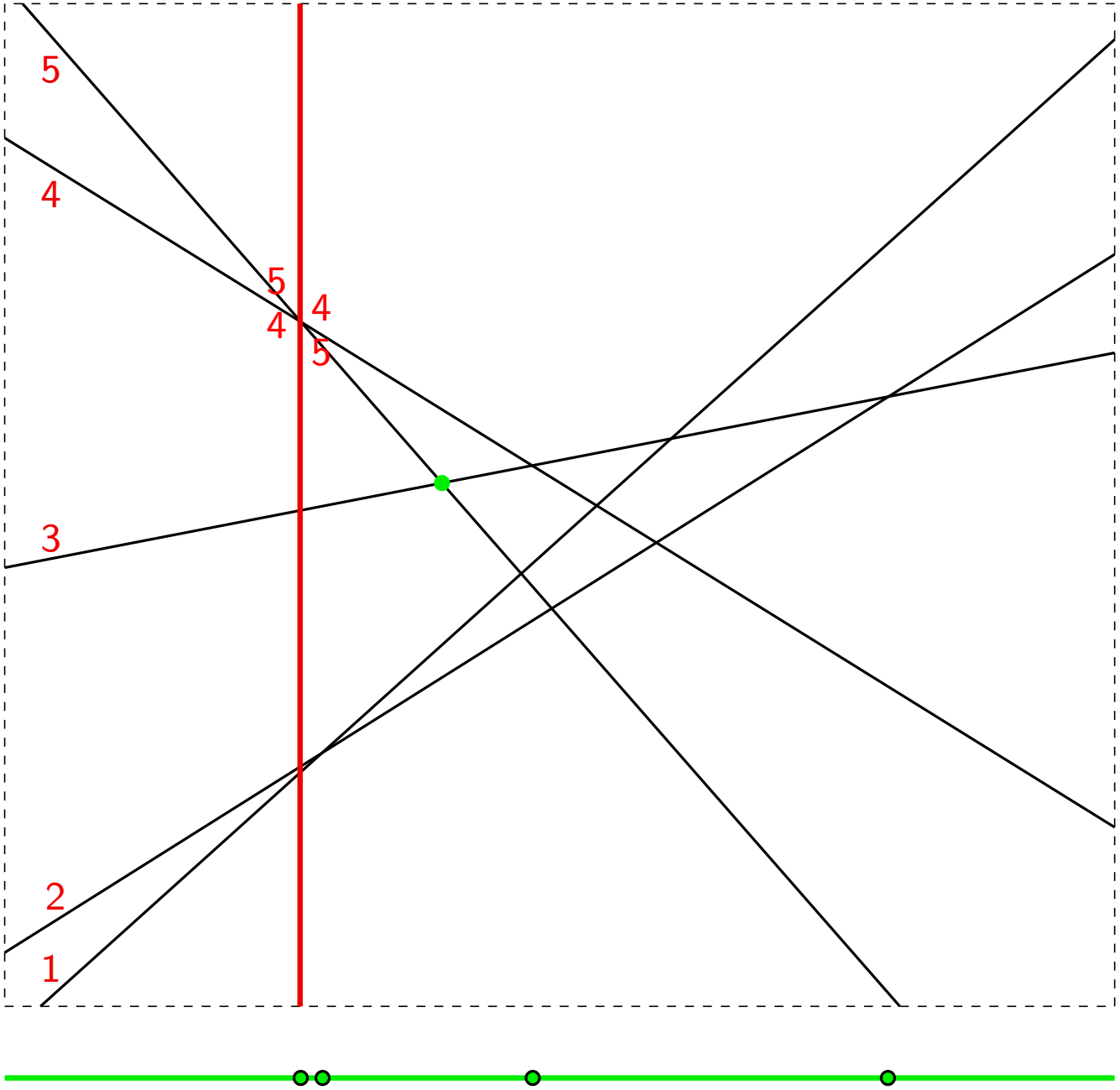


COMPUTATION

Sweep line algorithm

Advance: at each event,

1. Swap the order of the two intersecting lines in the sweep line.
2. Compute the intersection points with the two new neighboring lines.

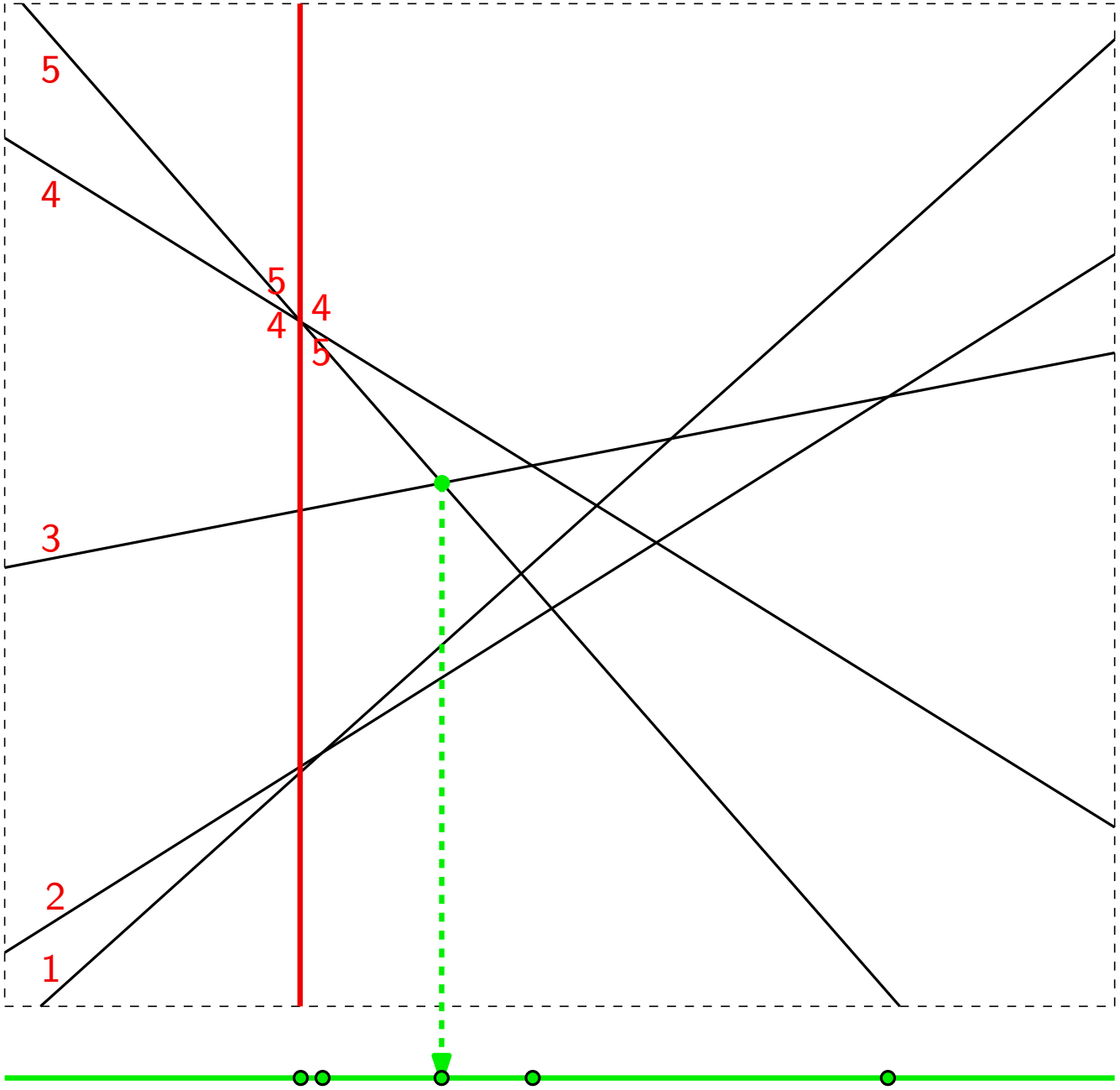


COMPUTATION

Sweep line algorithm

Advance: at each event,

1. Swap the order of the two intersecting lines in the sweep line.
2. Compute the intersection points with the two new neighboring lines.
3. If the intersection points are located forward in the sweeping order, add them to the events queue, if needed.

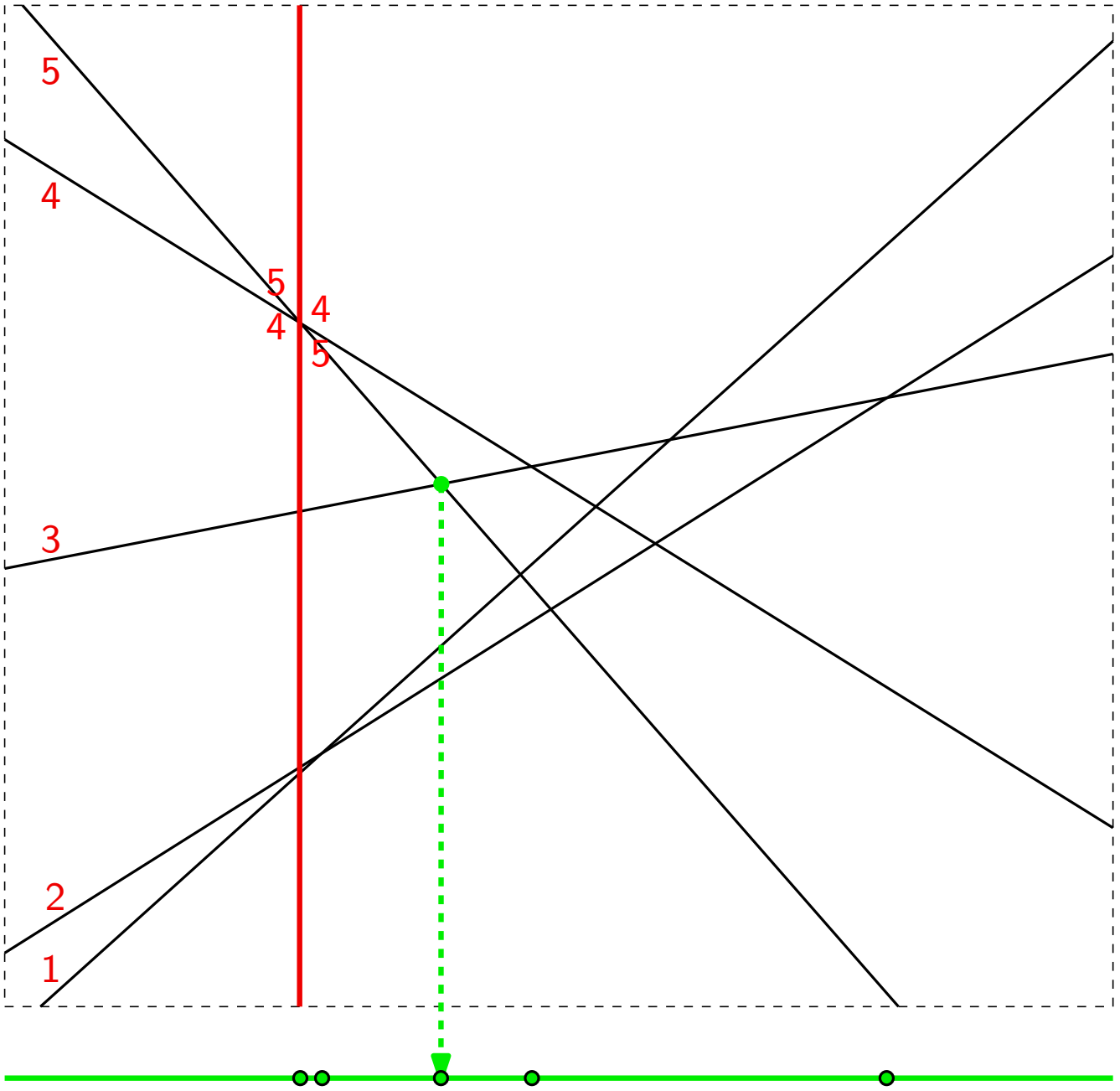


COMPUTATION

Sweep line algorithm

Advance: at each event,

1. Swap the order of the two intersecting lines in the sweep line.
2. Compute the intersection points with the two new neighboring lines.
3. If the intersection points are located forward in the sweeping order, add them to the events queue, if needed.
4. Update the DCEL.

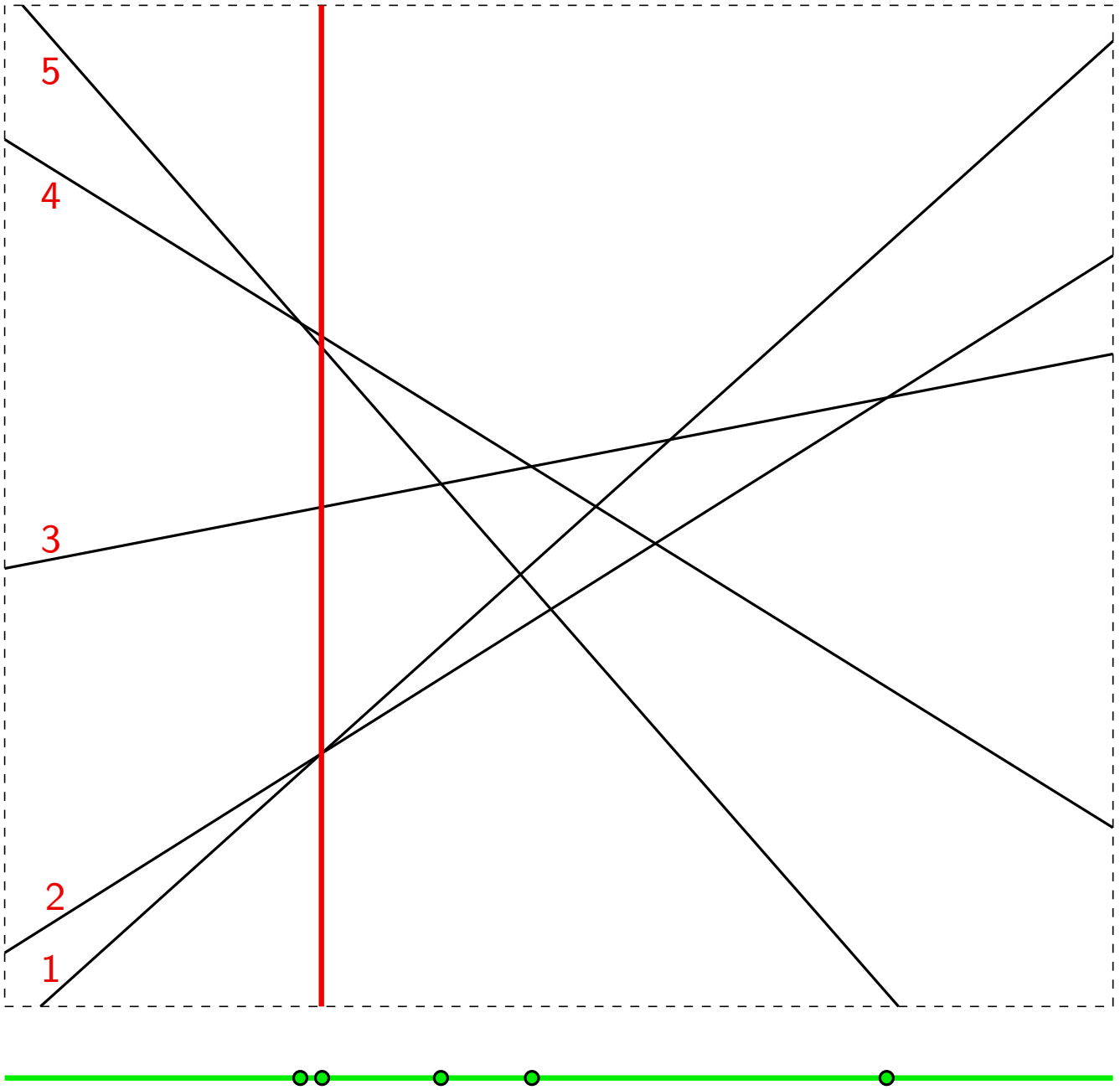


COMPUTATION

Sweep line algorithm

Advance: at each event,

1. Swap the order of the two intersecting lines in the sweep line.
2. Compute the intersection points with the two new neighboring lines.
3. If the intersection points are located forward in the sweeping order, add them to the events queue, if needed.
4. Update the DCEL.

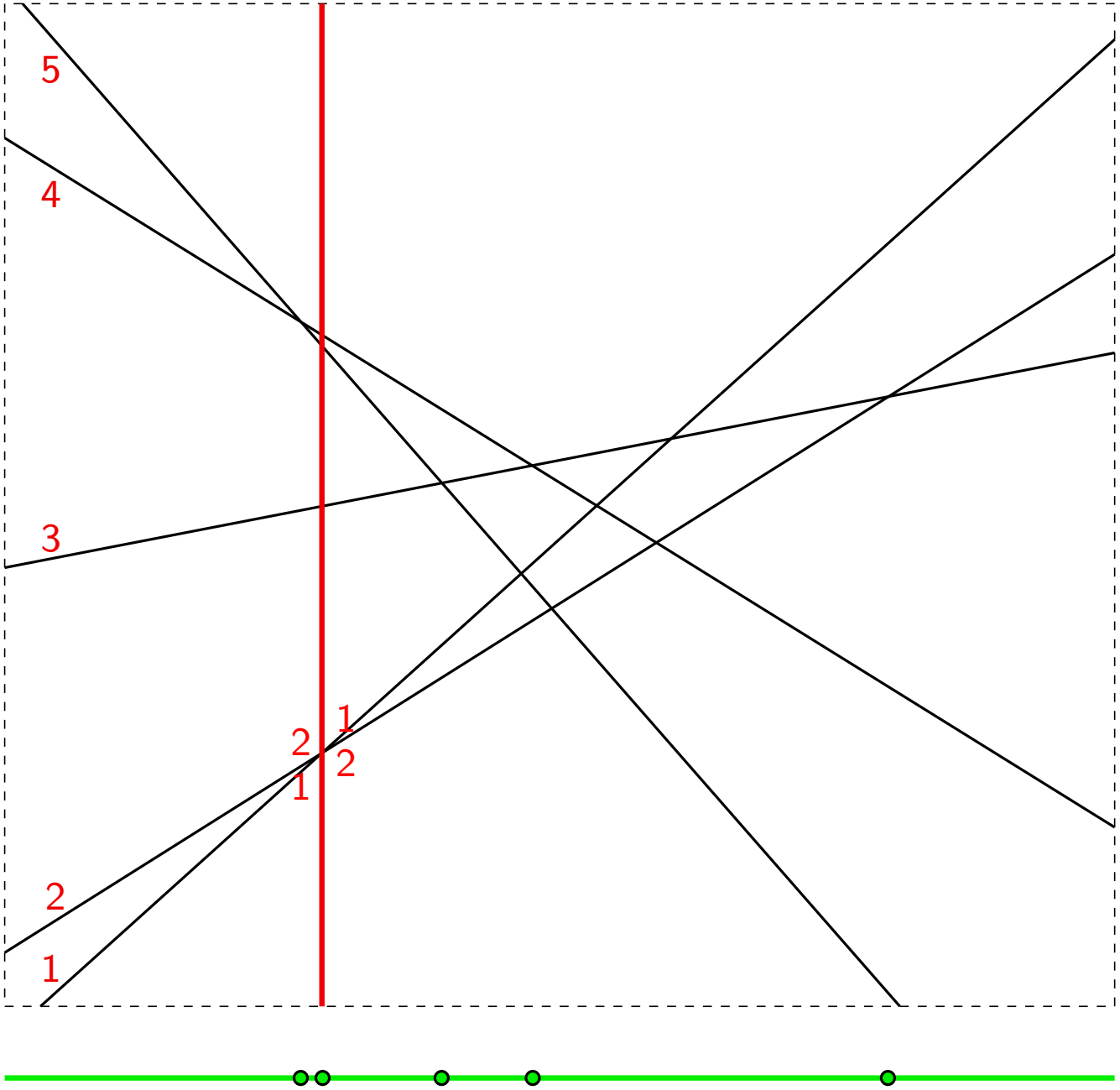


COMPUTATION

Sweep line algorithm

Advance: at each event,

1. Swap the order of the two intersecting lines in the sweep line.
2. Compute the intersection points with the two new neighboring lines.
3. If the intersection points are located forward in the sweeping order, add them to the events queue, if needed.
4. Update the DCEL.

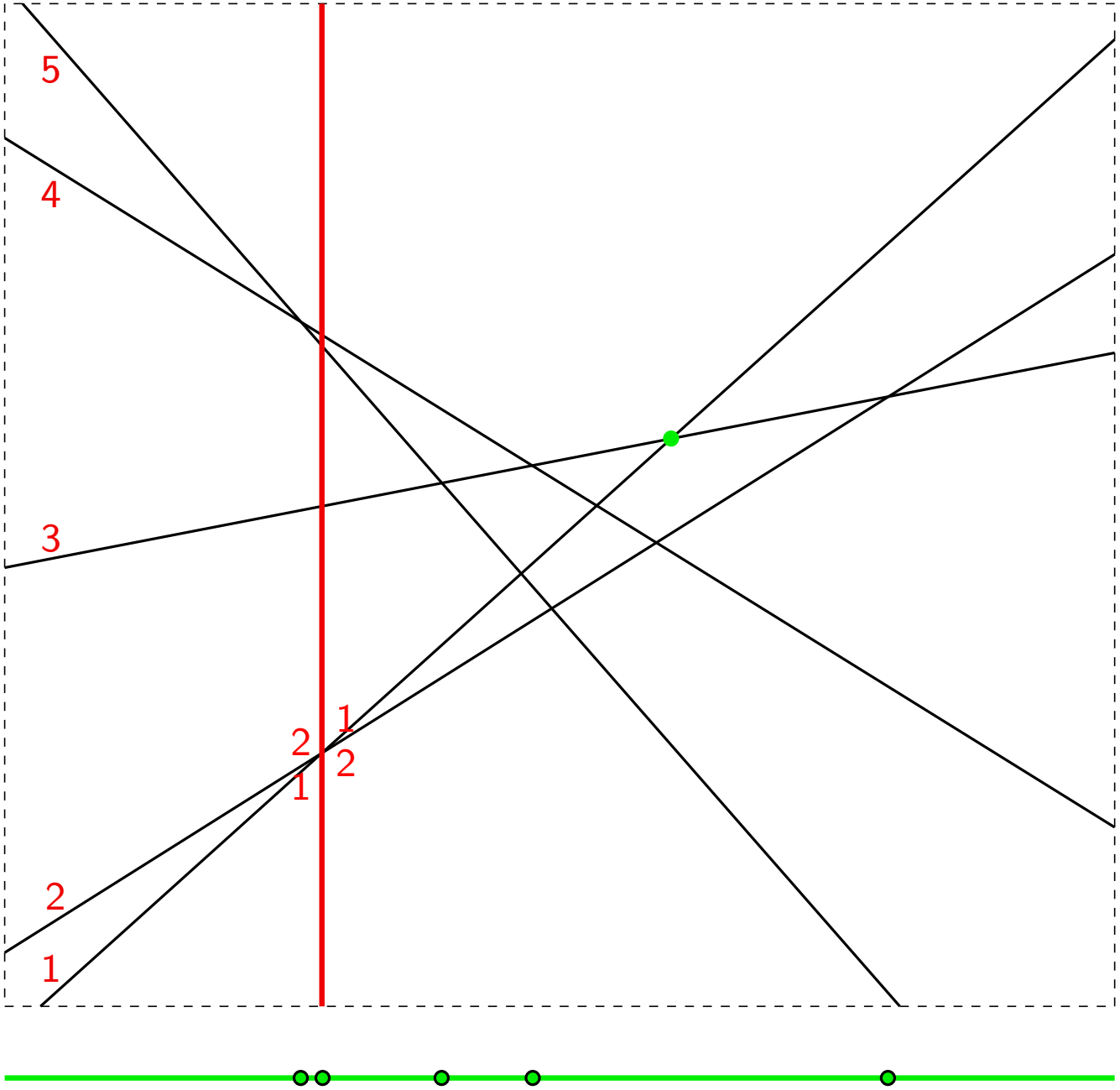


COMPUTATION

Sweep line algorithm

Advance: at each event,

1. Swap the order of the two intersecting lines in the sweep line.
2. Compute the intersection points with the two new neighboring lines.
3. If the intersection points are located forward in the sweeping order, add them to the events queue, if needed.
4. Update the DCEL.

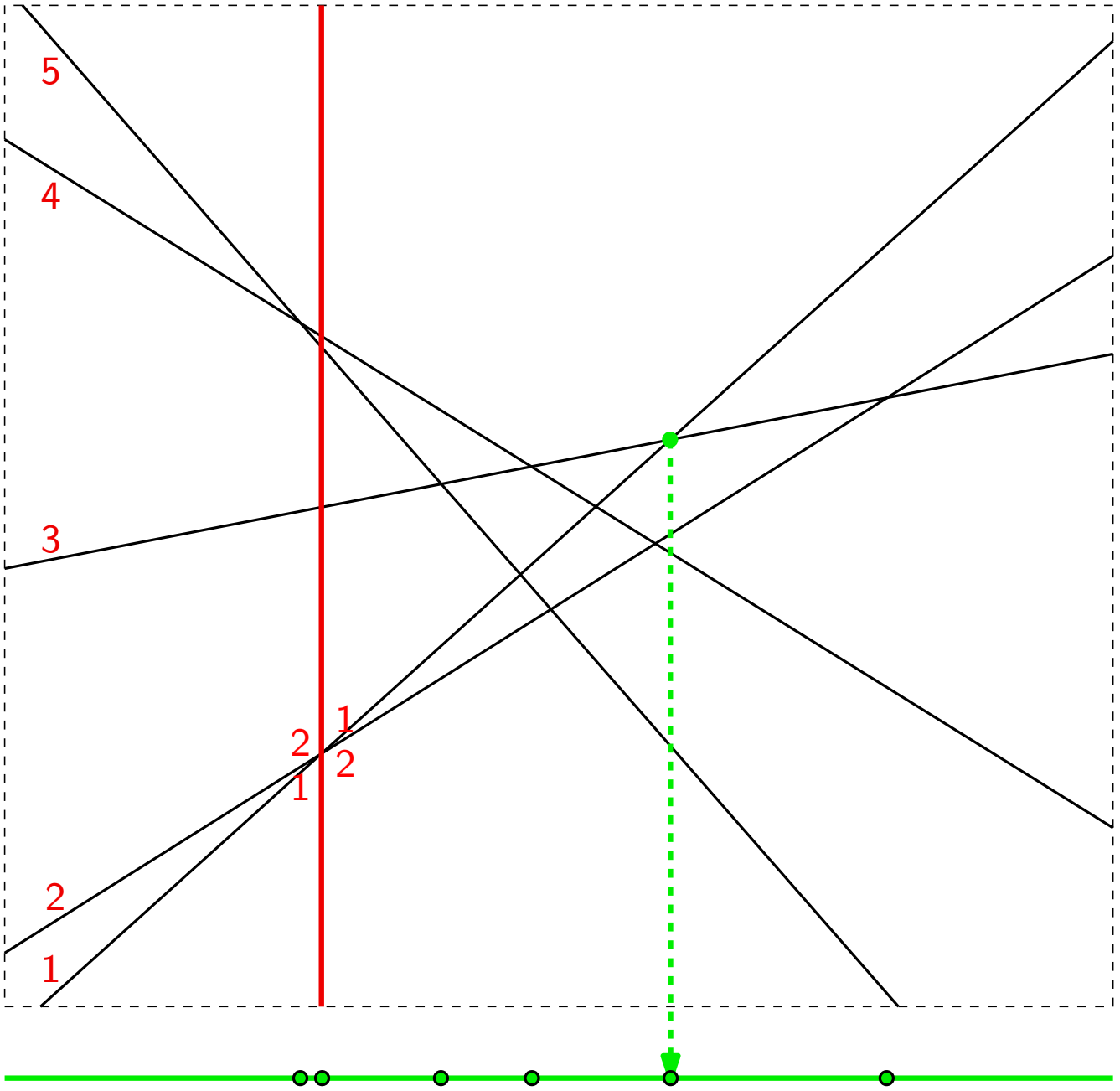


COMPUTATION

Sweep line algorithm

Advance: at each event,

1. Swap the order of the two intersecting lines in the sweep line.
2. Compute the intersection points with the two new neighboring lines.
3. If the intersection points are located forward in the sweeping order, add them to the events queue, if needed.
4. Update the DCEL.

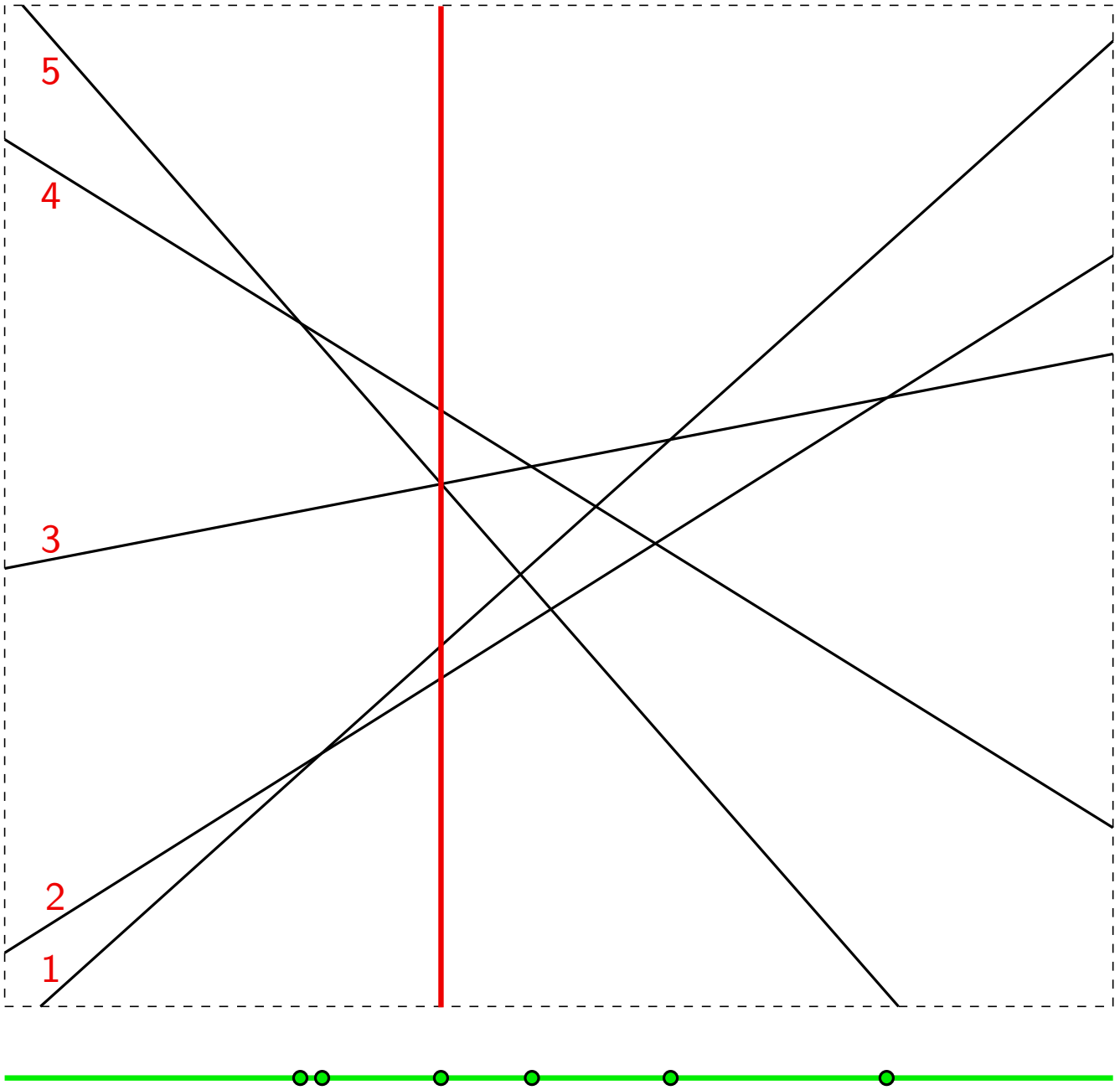


COMPUTATION

Sweep line algorithm

Advance: at each event,

1. Swap the order of the two intersecting lines in the sweep line.
2. Compute the intersection points with the two new neighboring lines.
3. If the intersection points are located forward in the sweeping order, add them to the events queue, if needed.
4. Update the DCEL.

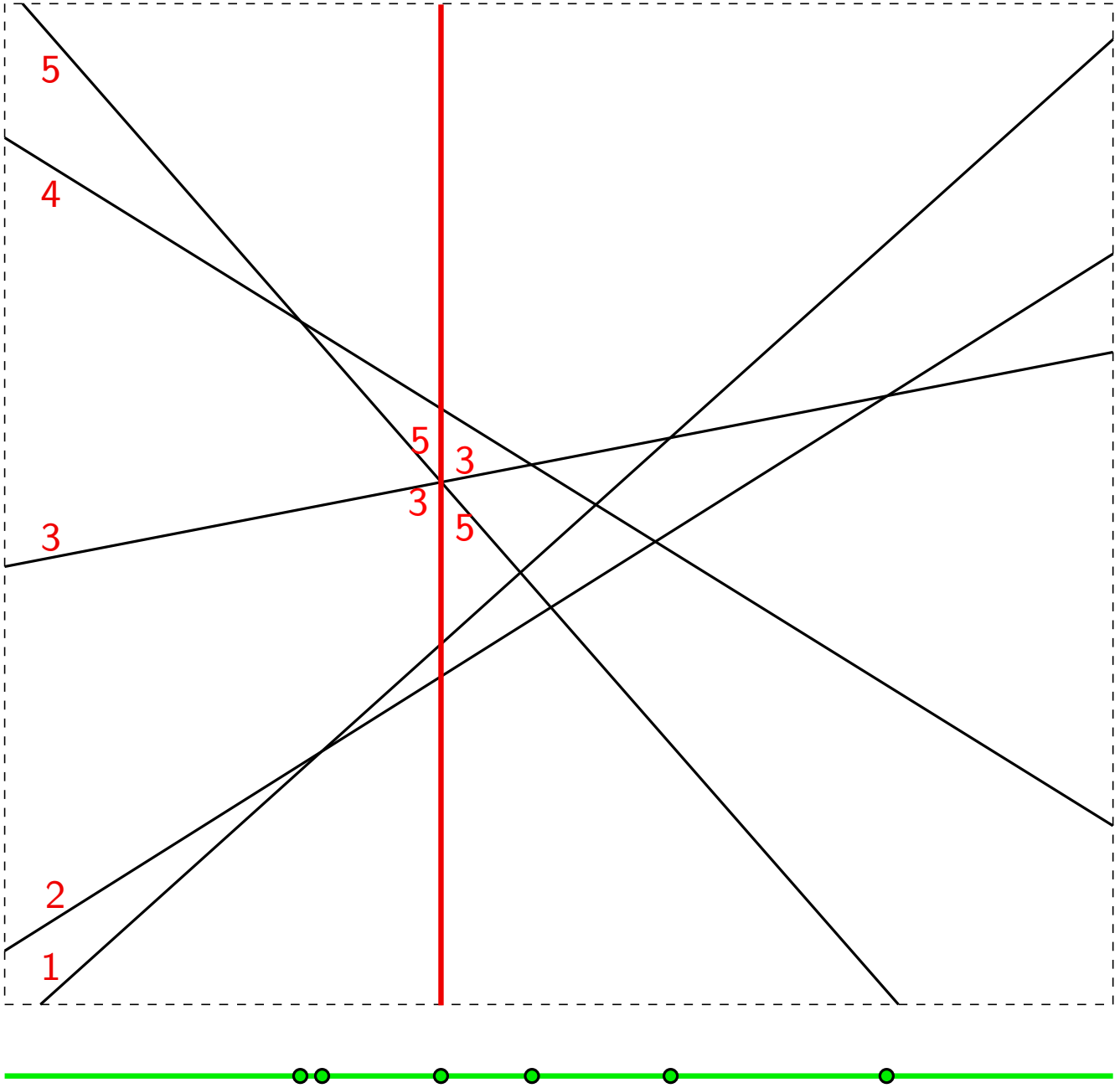


COMPUTATION

Sweep line algorithm

Advance: at each event,

1. Swap the order of the two intersecting lines in the sweep line.
2. Compute the intersection points with the two new neighboring lines.
3. If the intersection points are located forward in the sweeping order, add them to the events queue, if needed.
4. Update the DCEL.

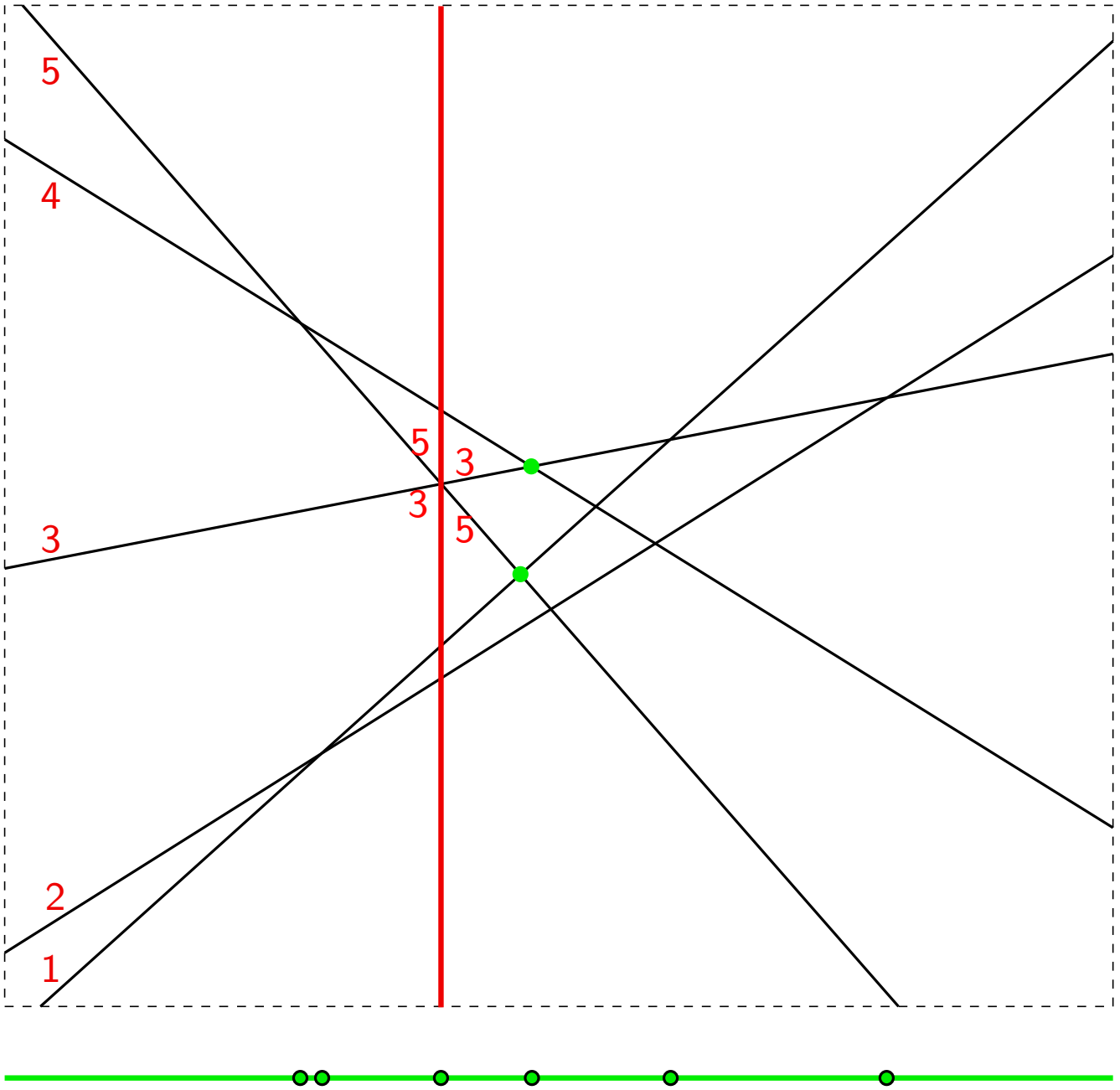


COMPUTATION

Sweep line algorithm

Advance: at each event,

1. Swap the order of the two intersecting lines in the sweep line.
2. Compute the intersection points with the two new neighboring lines.
3. If the intersection points are located forward in the sweeping order, add them to the events queue, if needed.
4. Update the DCEL.

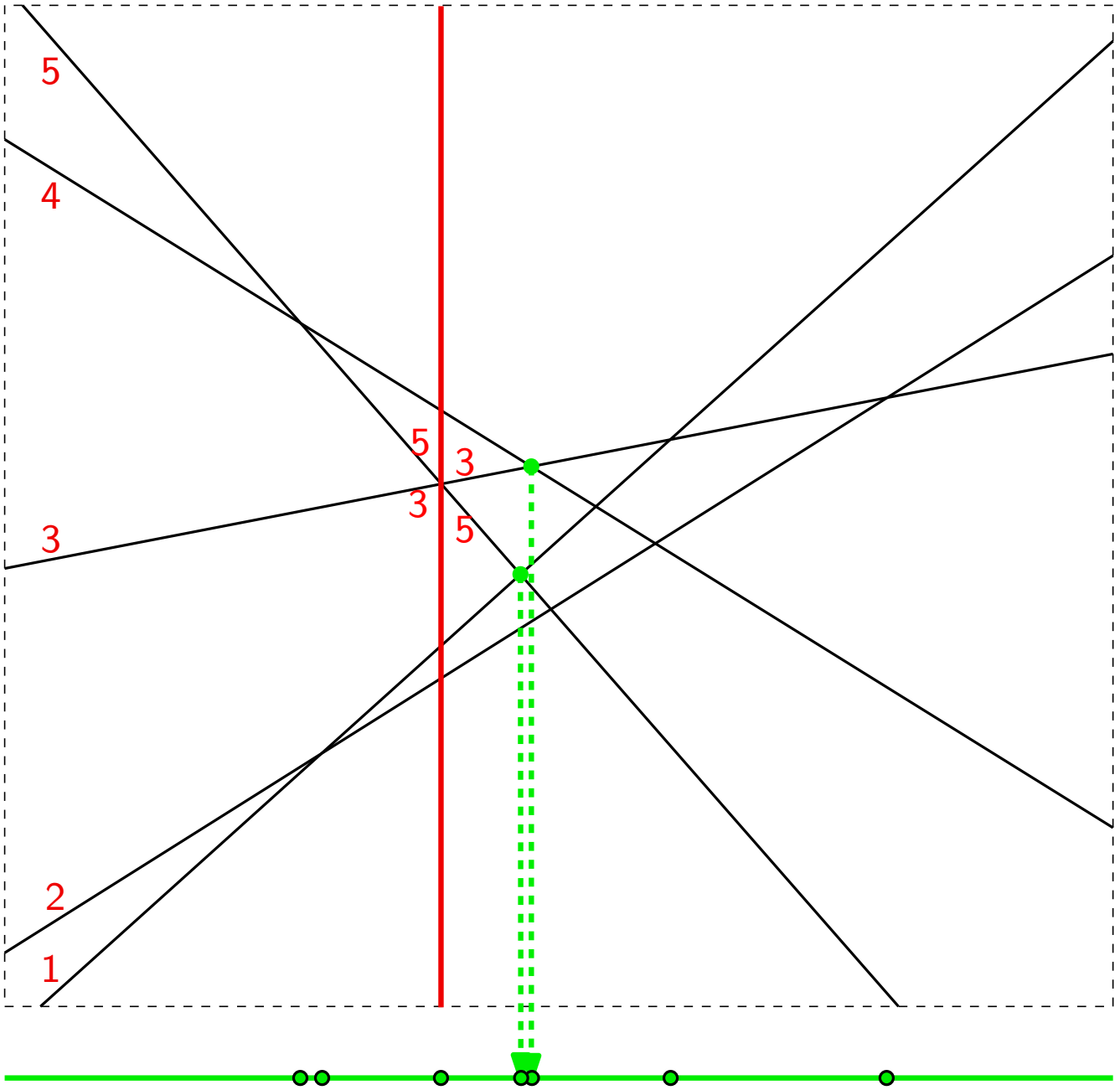


COMPUTATION

Sweep line algorithm

Advance: at each event,

1. Swap the order of the two intersecting lines in the sweep line.
2. Compute the intersection points with the two new neighboring lines.
3. If the intersection points are located forward in the sweeping order, add them to the events queue, if needed.
4. Update the DCEL.

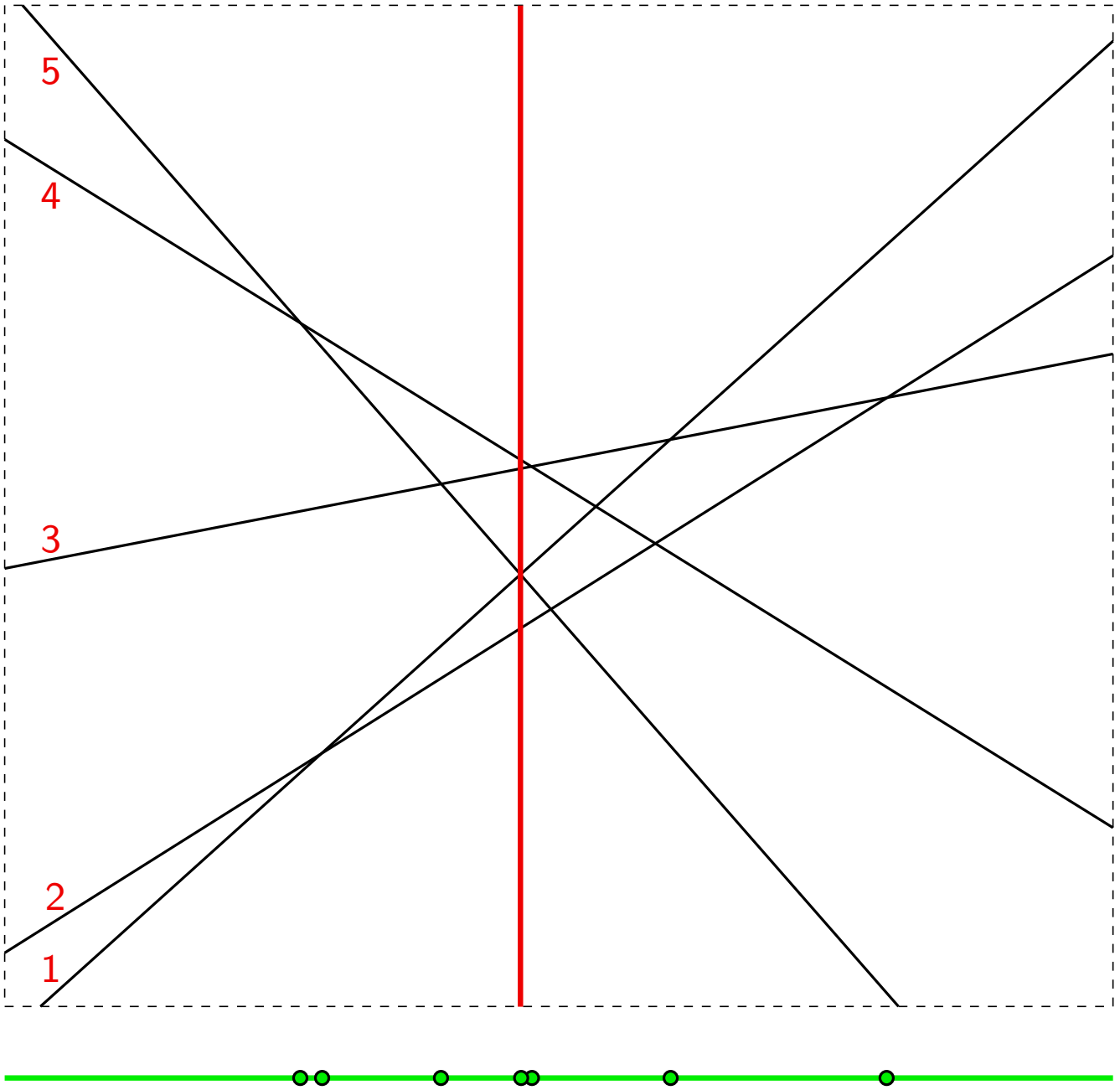


COMPUTATION

Sweep line algorithm

Advance: at each event,

1. Swap the order of the two intersecting lines in the sweep line.
2. Compute the intersection points with the two new neighboring lines.
3. If the intersection points are located forward in the sweeping order, add them to the events queue, if needed.
4. Update the DCEL.

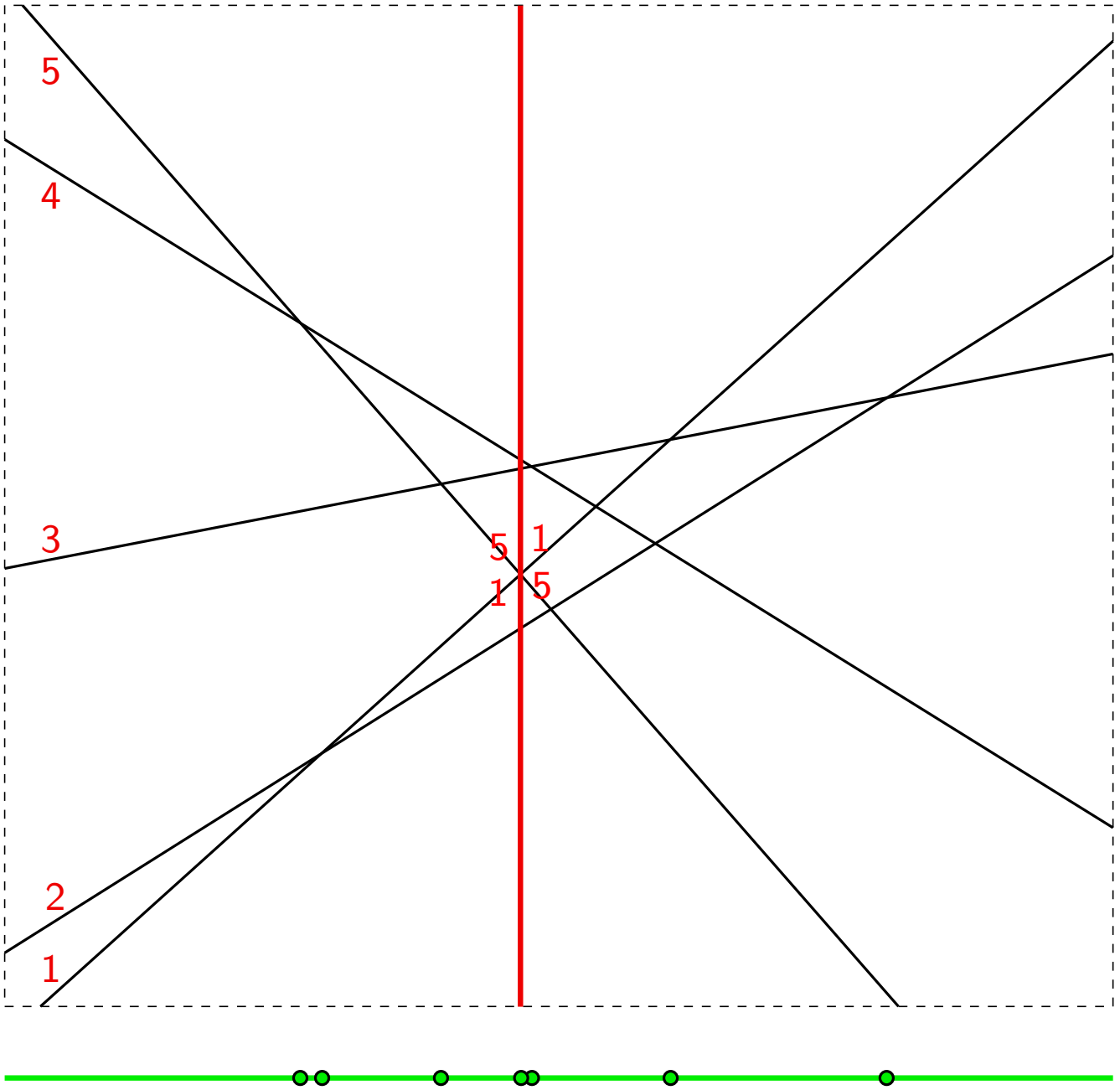


COMPUTATION

Sweep line algorithm

Advance: at each event,

1. Swap the order of the two intersecting lines in the sweep line.
2. Compute the intersection points with the two new neighboring lines.
3. If the intersection points are located forward in the sweeping order, add them to the events queue, if needed.
4. Update the DCEL.

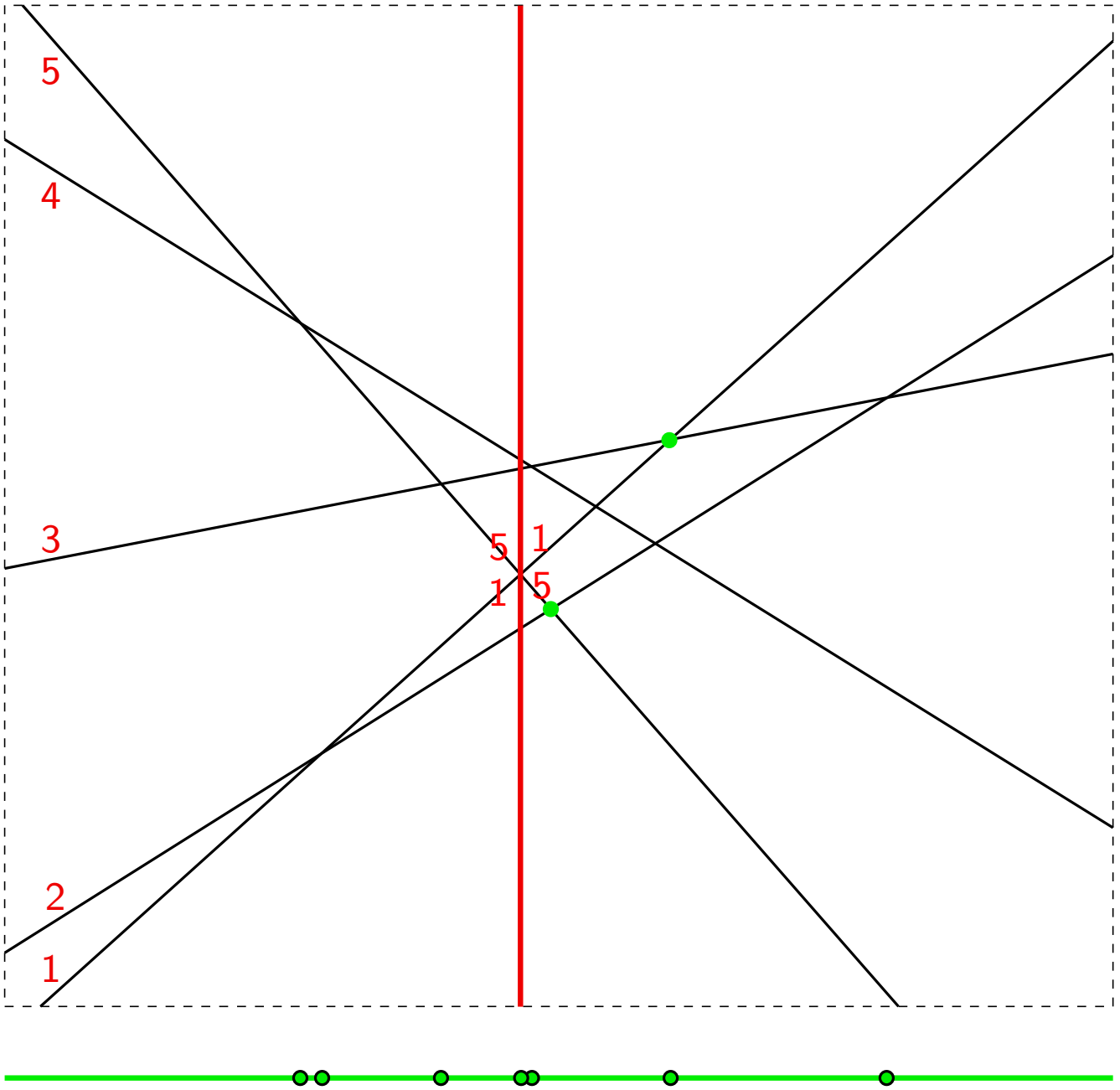


COMPUTATION

Sweep line algorithm

Advance: at each event,

1. Swap the order of the two intersecting lines in the sweep line.
2. Compute the intersection points with the two new neighboring lines.
3. If the intersection points are located forward in the sweeping order, add them to the events queue, if needed.
4. Update the DCEL.

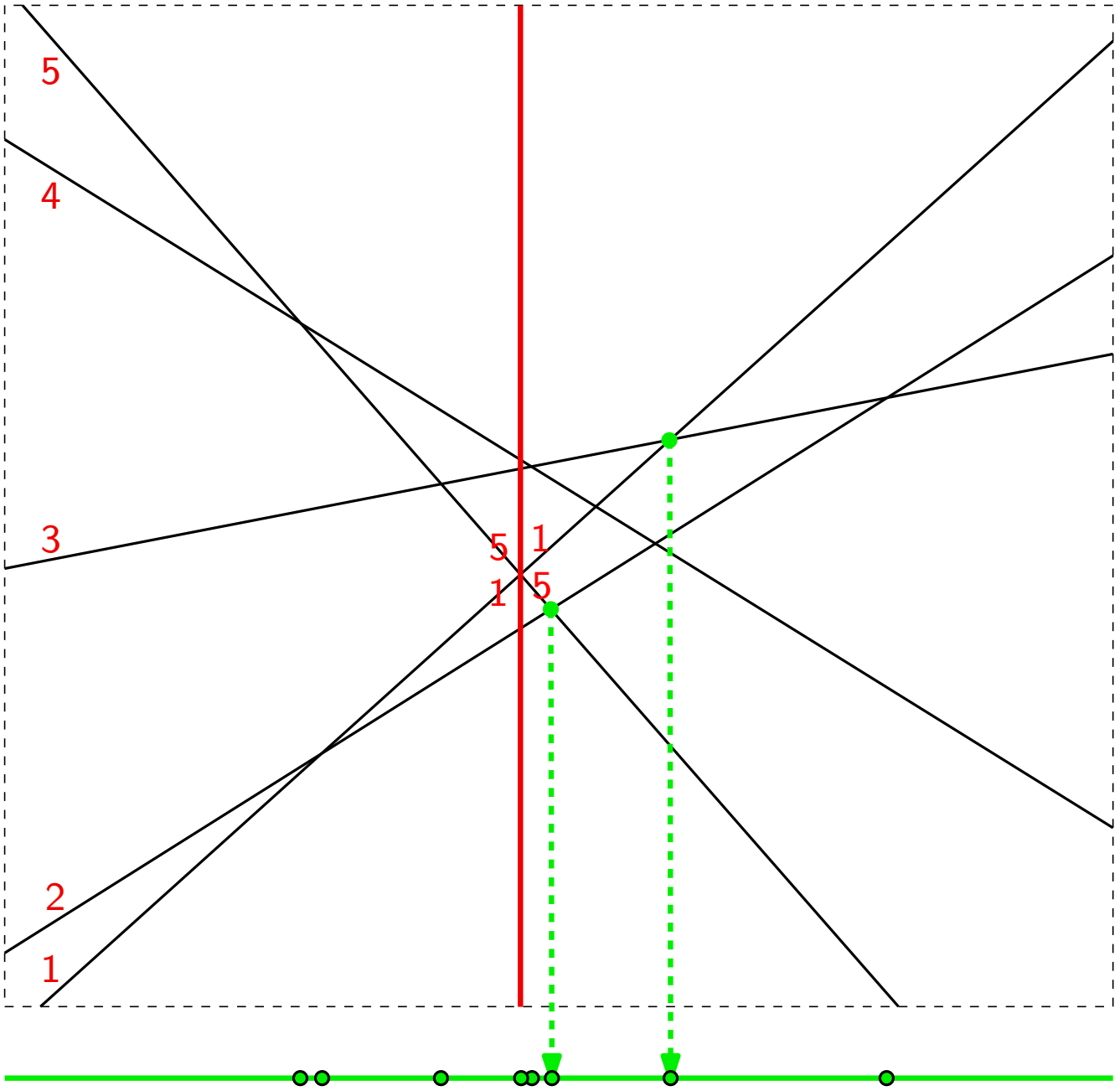


COMPUTATION

Sweep line algorithm

Advance: at each event,

1. Swap the order of the two intersecting lines in the sweep line.
2. Compute the intersection points with the two new neighboring lines.
3. If the intersection points are located forward in the sweeping order, add them to the events queue, if needed.
4. Update the DCEL.

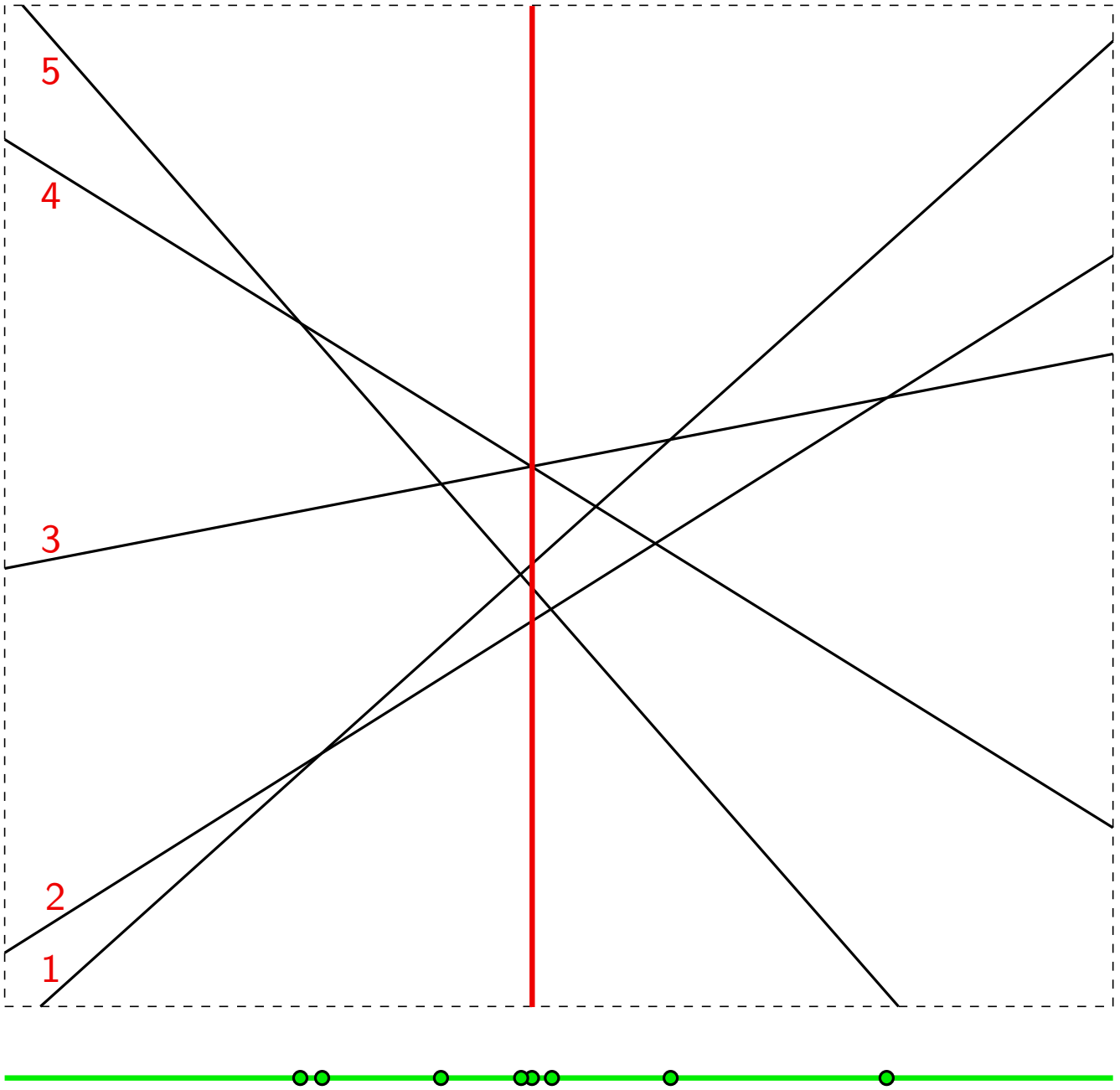


COMPUTATION

Sweep line algorithm

Advance: at each event,

1. Swap the order of the two intersecting lines in the sweep line.
2. Compute the intersection points with the two new neighboring lines.
3. If the intersection points are located forward in the sweeping order, add them to the events queue, if needed.
4. Update the DCEL.

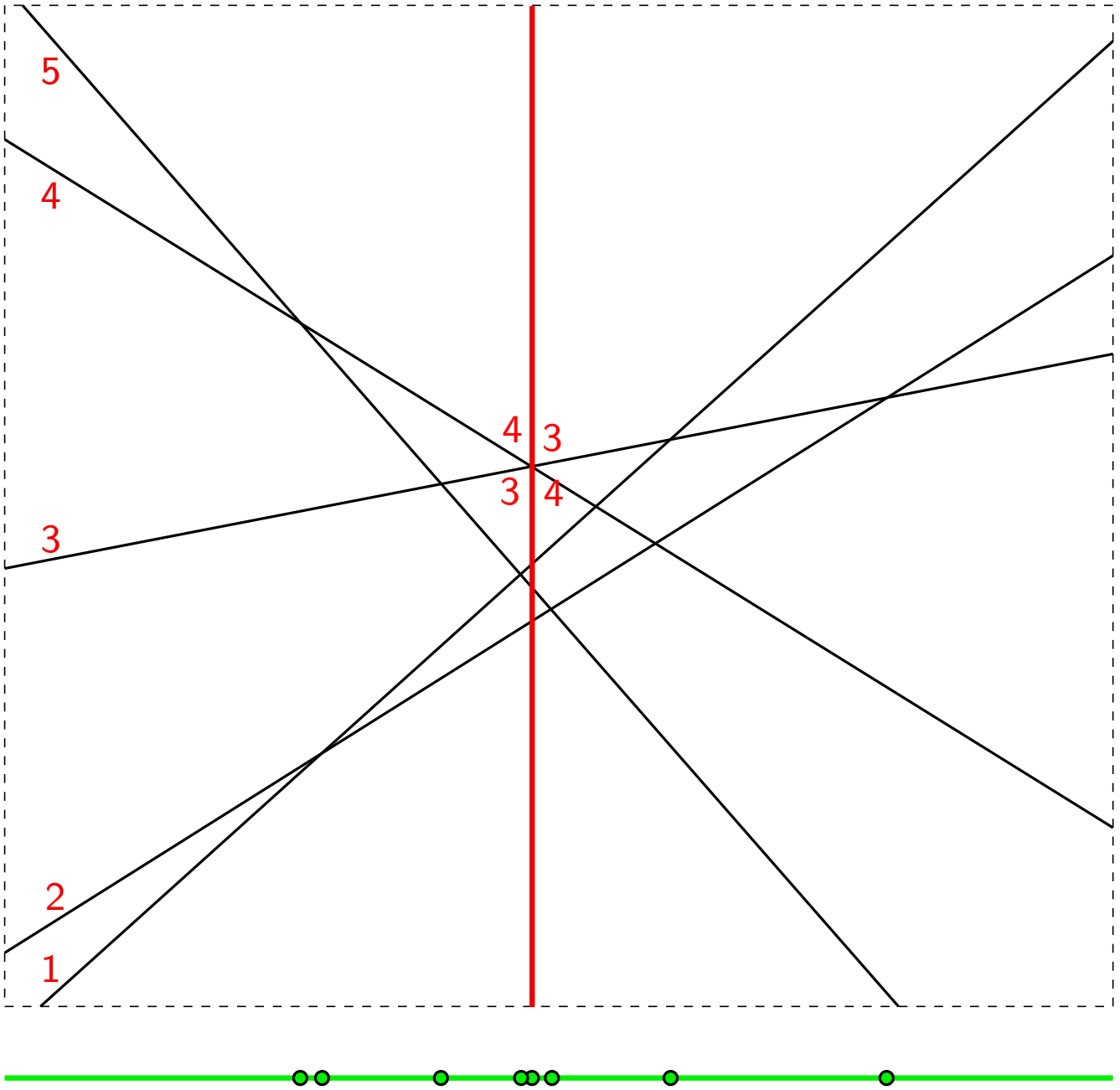


COMPUTATION

Sweep line algorithm

Advance: at each event,

1. Swap the order of the two intersecting lines in the sweep line.
2. Compute the intersection points with the two new neighboring lines.
3. If the intersection points are located forward in the sweeping order, add them to the events queue, if needed.
4. Update the DCEL.

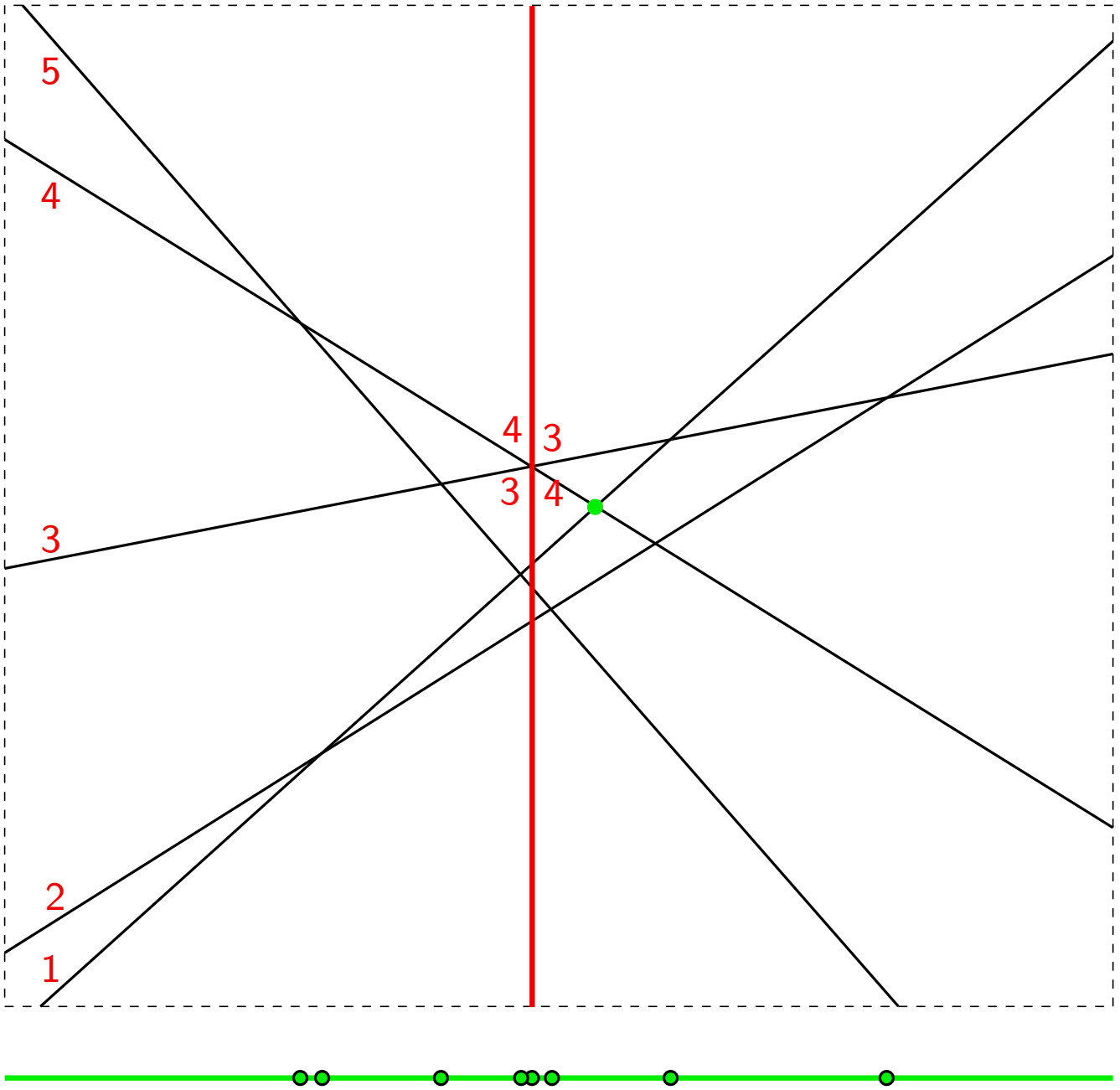


COMPUTATION

Sweep line algorithm

Advance: at each event,

1. Swap the order of the two intersecting lines in the sweep line.
2. Compute the intersection points with the two new neighboring lines.
3. If the intersection points are located forward in the sweeping order, add them to the events queue, if needed.
4. Update the DCEL.

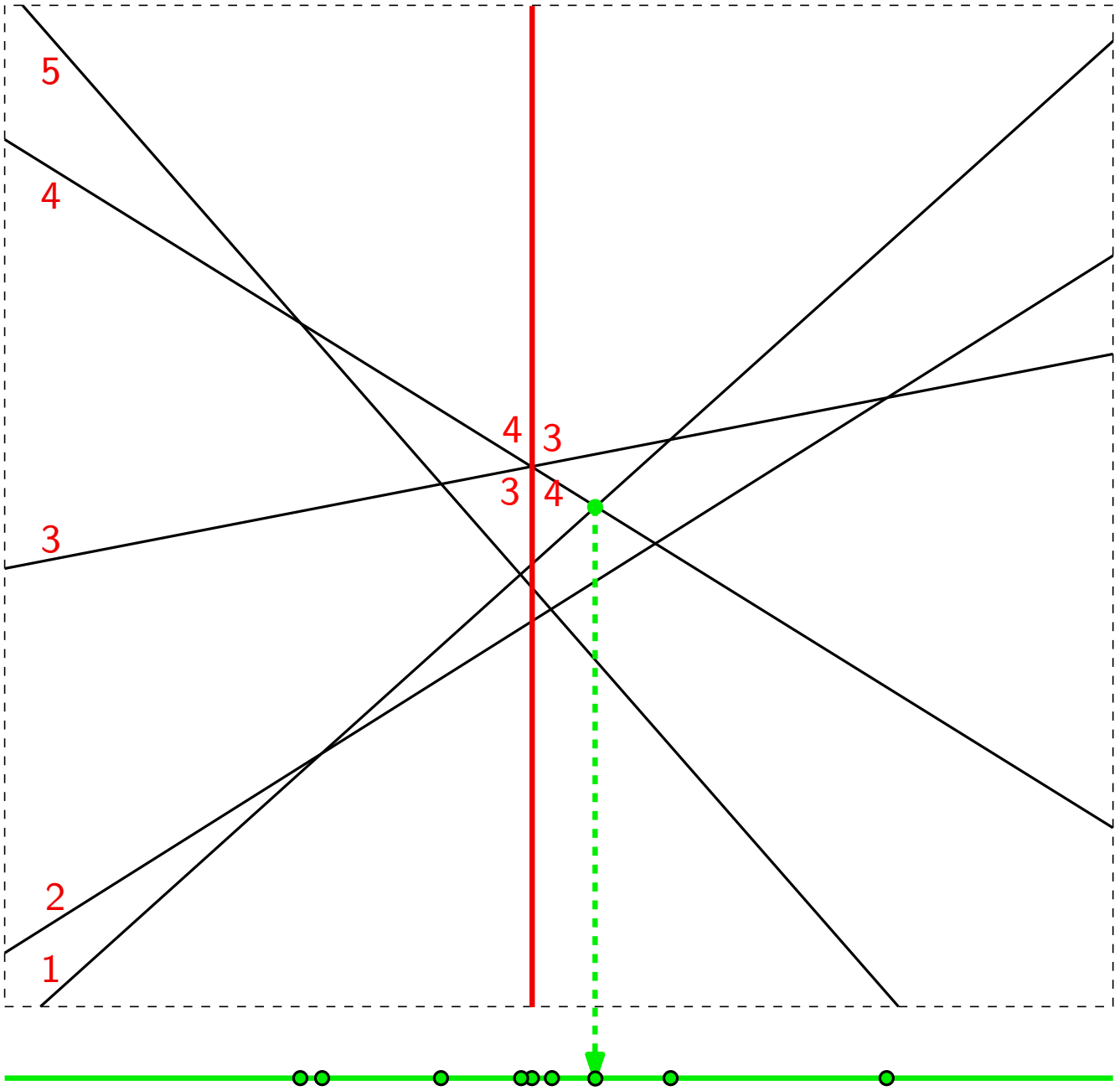


COMPUTATION

Sweep line algorithm

Advance: at each event,

1. Swap the order of the two intersecting lines in the sweep line.
2. Compute the intersection points with the two new neighboring lines.
3. If the intersection points are located forward in the sweeping order, add them to the events queue, if needed.
4. Update the DCEL.

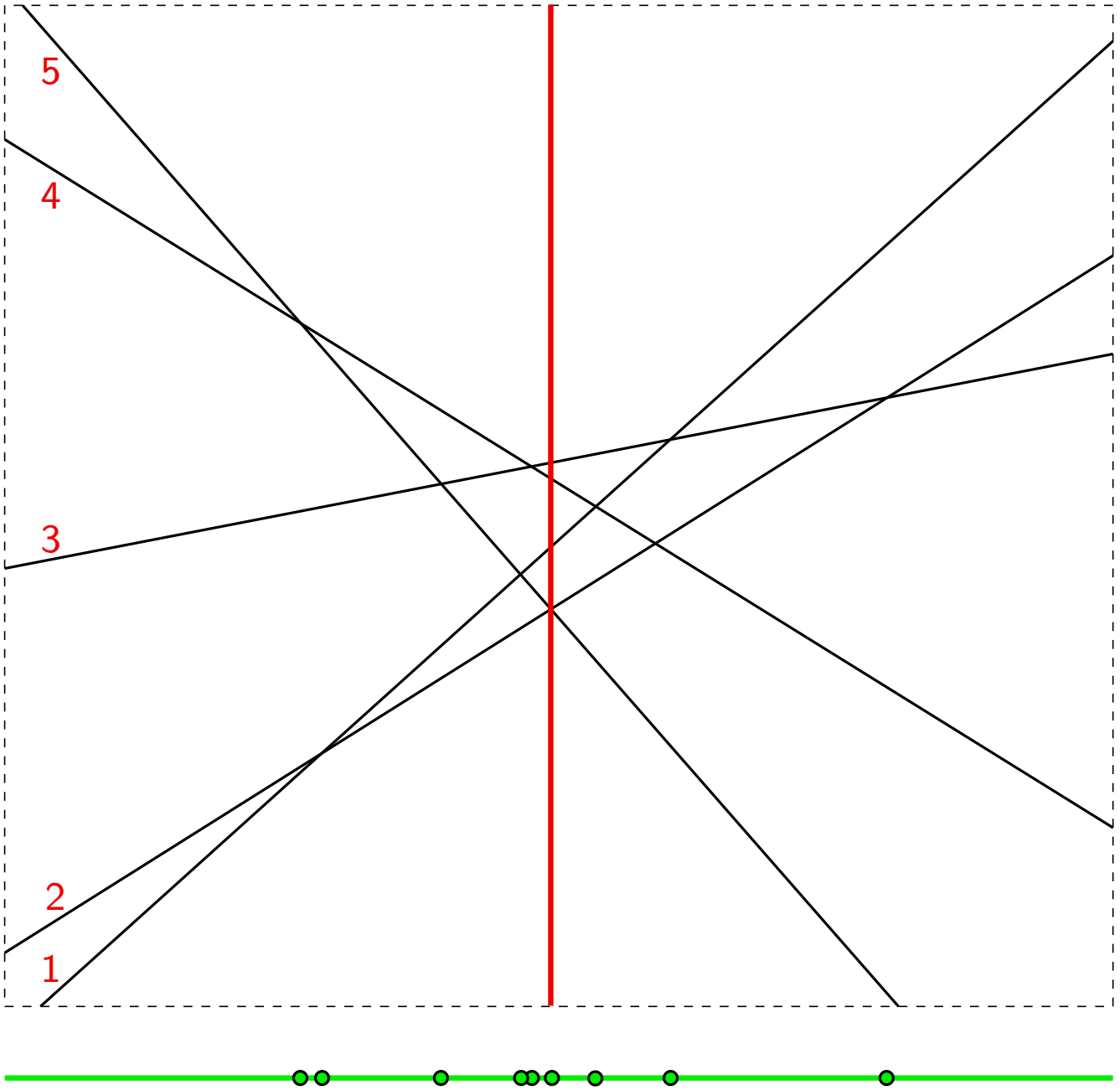


COMPUTATION

Sweep line algorithm

Advance: at each event,

1. Swap the order of the two intersecting lines in the sweep line.
2. Compute the intersection points with the two new neighboring lines.
3. If the intersection points are located forward in the sweeping order, add them to the events queue, if needed.
4. Update the DCEL.

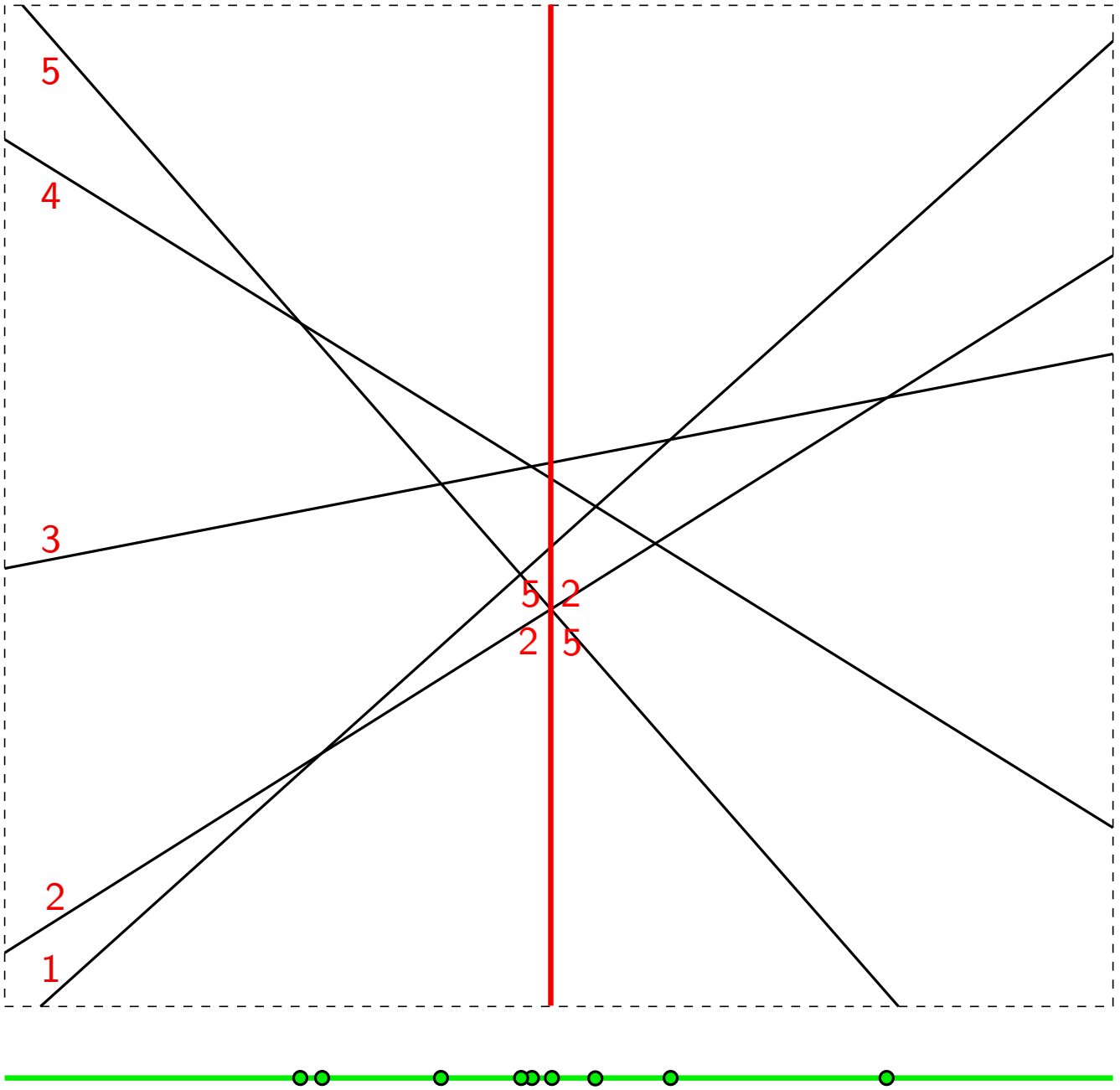


COMPUTATION

Sweep line algorithm

Advance: at each event,

1. Swap the order of the two intersecting lines in the sweep line.
2. Compute the intersection points with the two new neighboring lines.
3. If the intersection points are located forward in the sweeping order, add them to the events queue, if needed.
4. Update the DCEL.

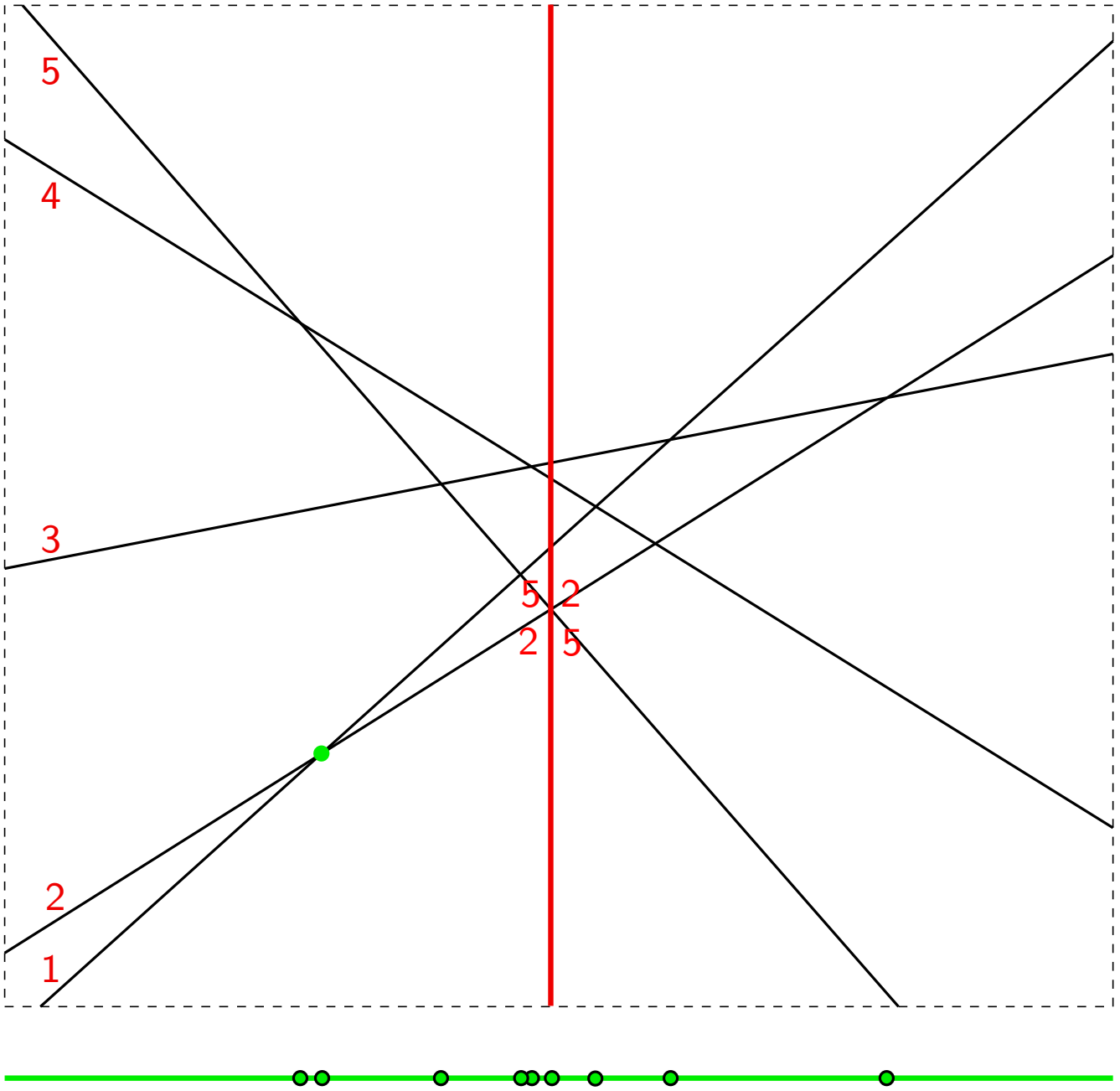


COMPUTATION

Sweep line algorithm

Advance: at each event,

1. Swap the order of the two intersecting lines in the sweep line.
2. Compute the intersection points with the two new neighboring lines.
3. If the intersection points are located forward in the sweeping order, add them to the events queue, if needed.
4. Update the DCEL.

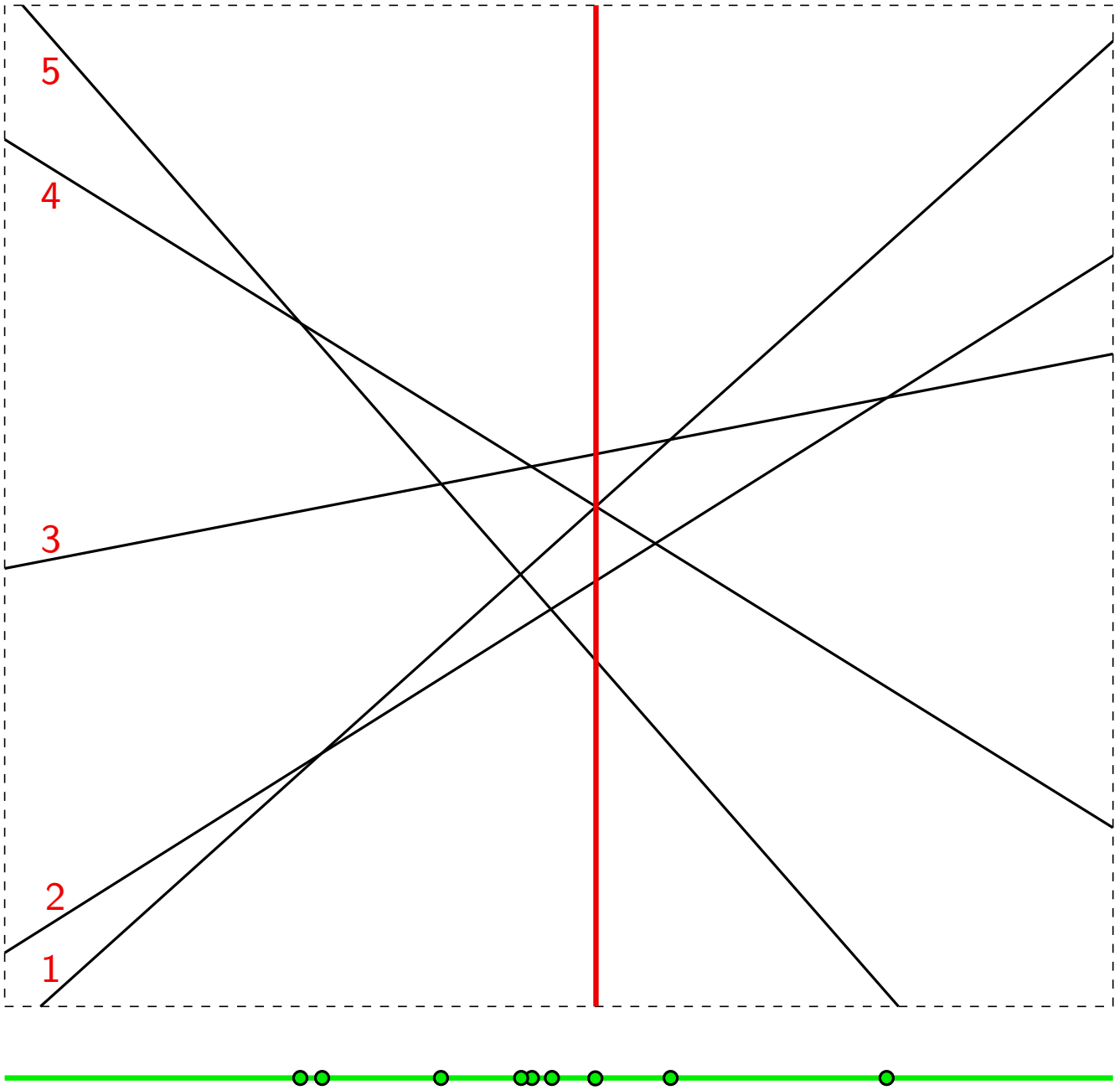


COMPUTATION

Sweep line algorithm

Advance: at each event,

1. Swap the order of the two intersecting lines in the sweep line.
2. Compute the intersection points with the two new neighboring lines.
3. If the intersection points are located forward in the sweeping order, add them to the events queue, if needed.
4. Update the DCEL.

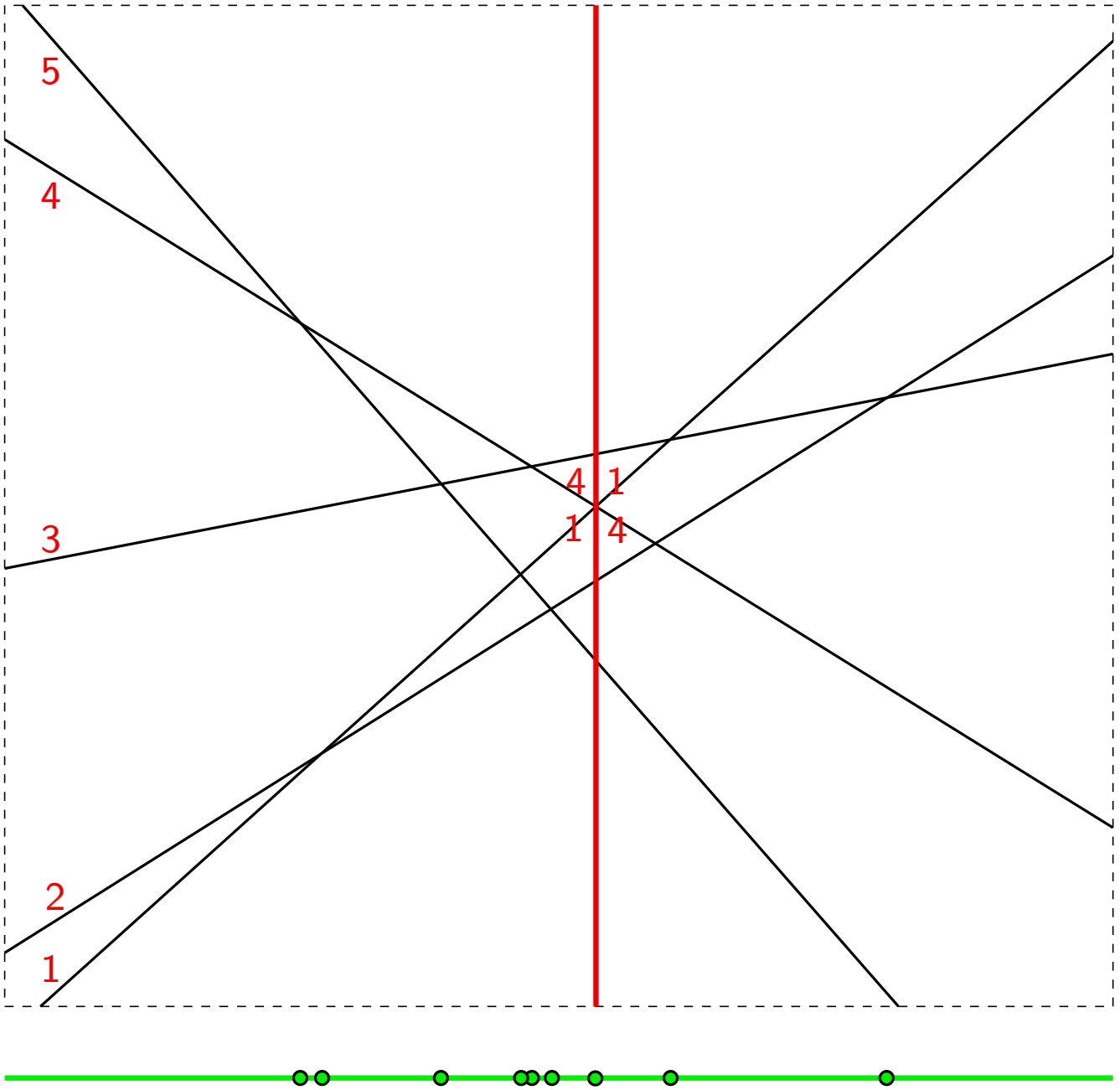


COMPUTATION

Sweep line algorithm

Advance: at each event,

1. Swap the order of the two intersecting lines in the sweep line.
2. Compute the intersection points with the two new neighboring lines.
3. If the intersection points are located forward in the sweeping order, add them to the events queue, if needed.
4. Update the DCEL.

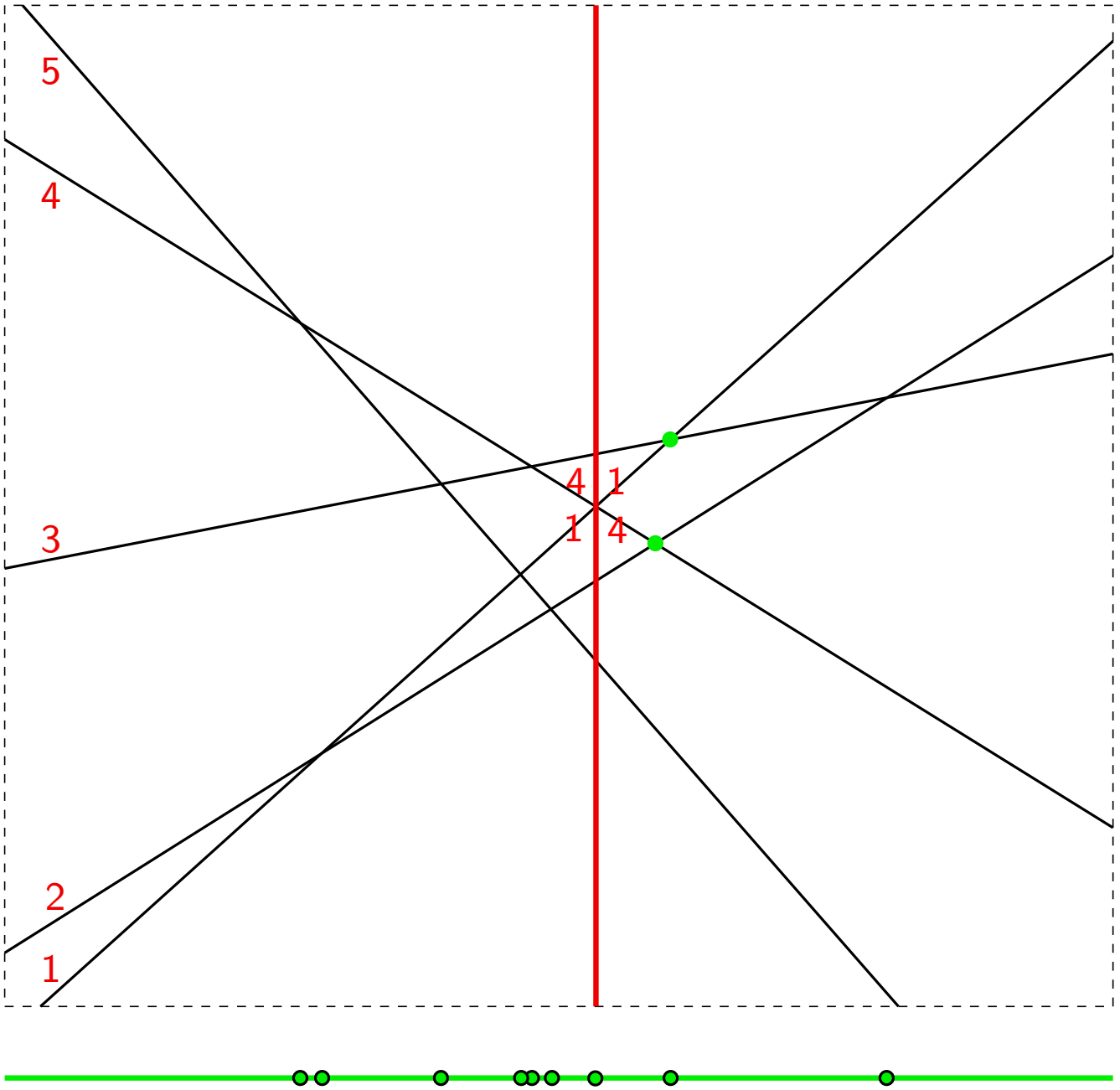


COMPUTATION

Sweep line algorithm

Advance: at each event,

1. Swap the order of the two intersecting lines in the sweep line.
2. Compute the intersection points with the two new neighboring lines.
3. If the intersection points are located forward in the sweeping order, add them to the events queue, if needed.
4. Update the DCEL.

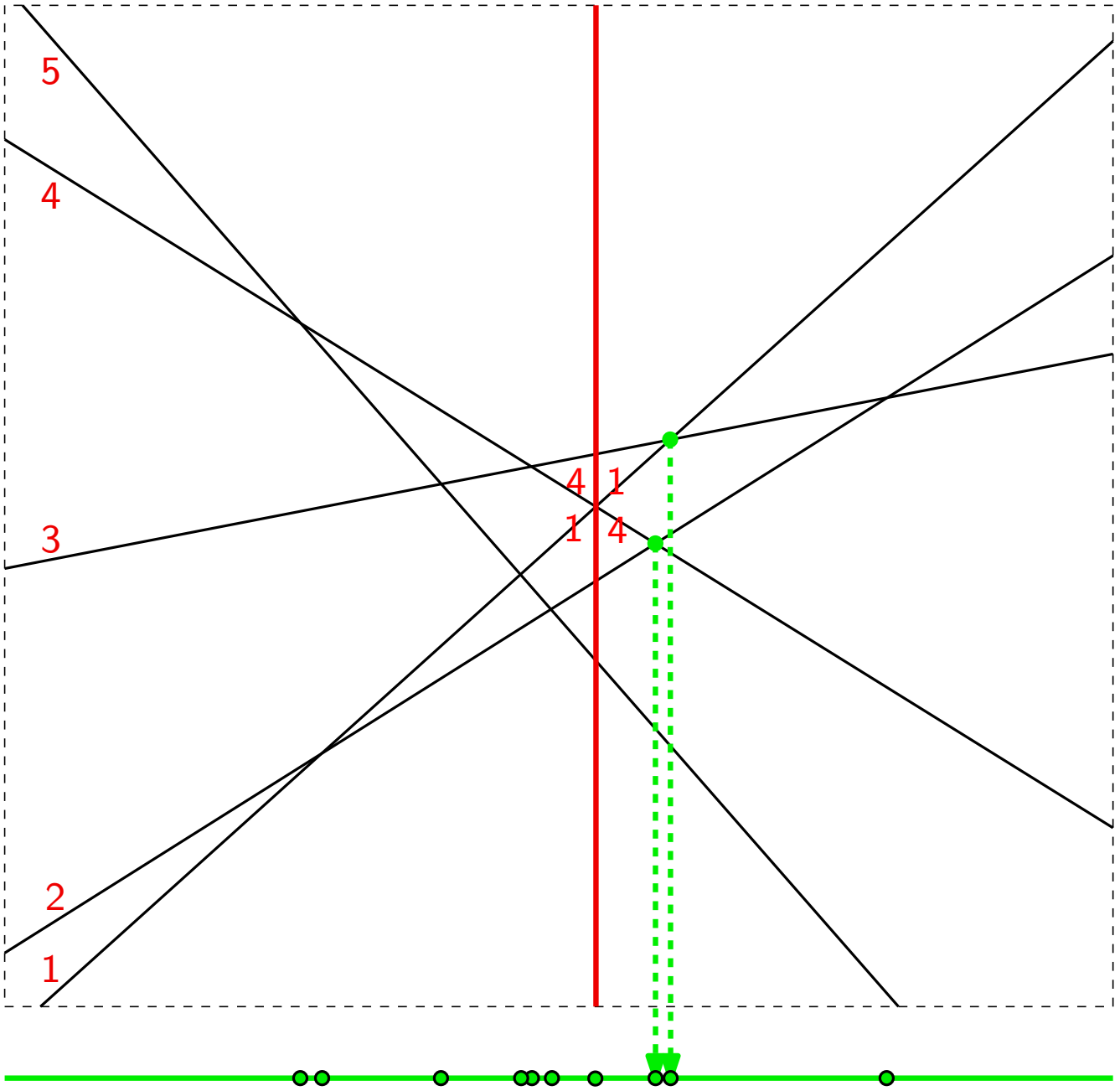


COMPUTATION

Sweep line algorithm

Advance: at each event,

1. Swap the order of the two intersecting lines in the sweep line.
2. Compute the intersection points with the two new neighboring lines.
3. If the intersection points are located forward in the sweeping order, add them to the events queue, if needed.
4. Update the DCEL.

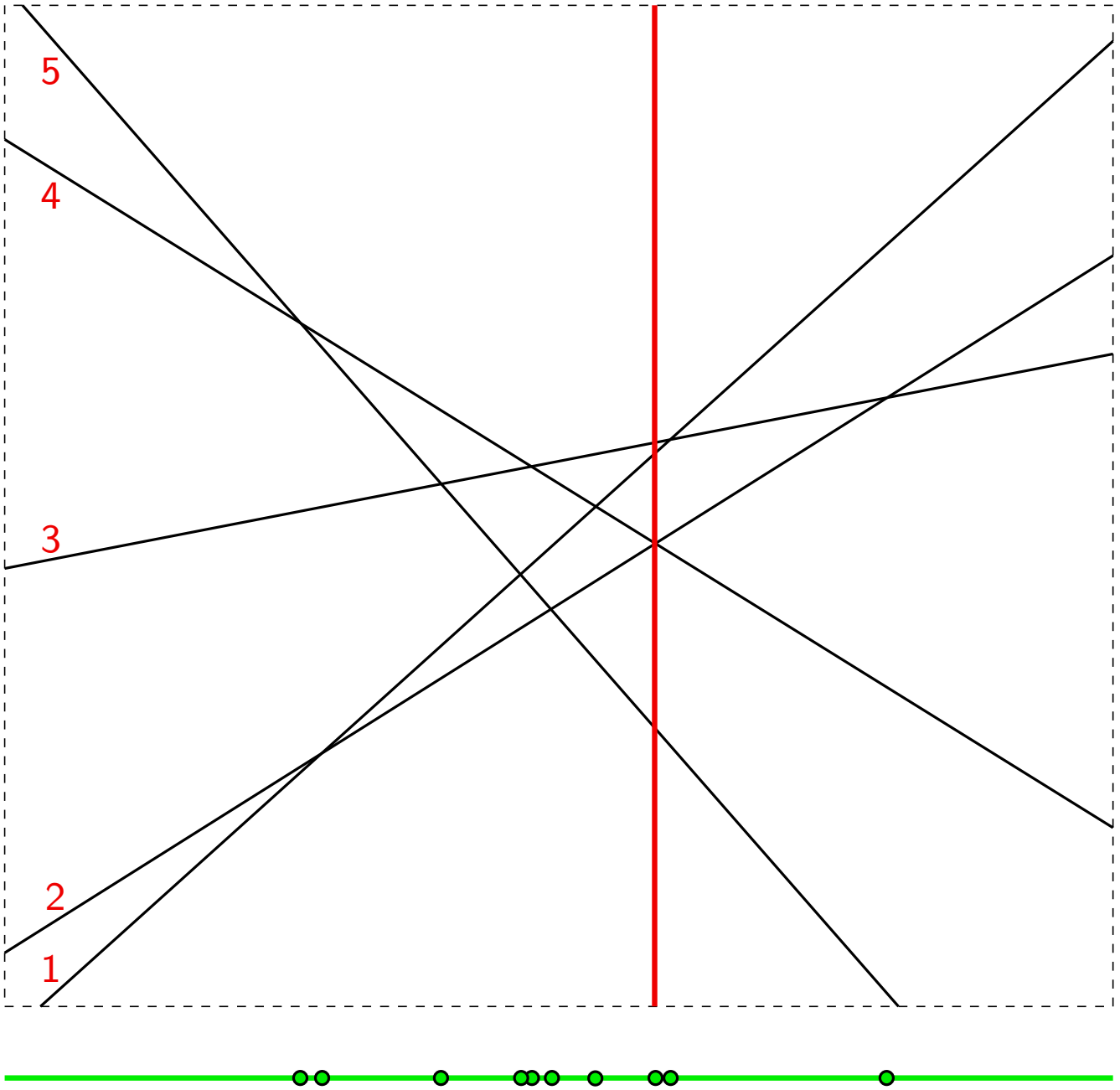


COMPUTATION

Sweep line algorithm

Advance: at each event,

1. Swap the order of the two intersecting lines in the sweep line.
2. Compute the intersection points with the two new neighboring lines.
3. If the intersection points are located forward in the sweeping order, add them to the events queue, if needed.
4. Update the DCEL.

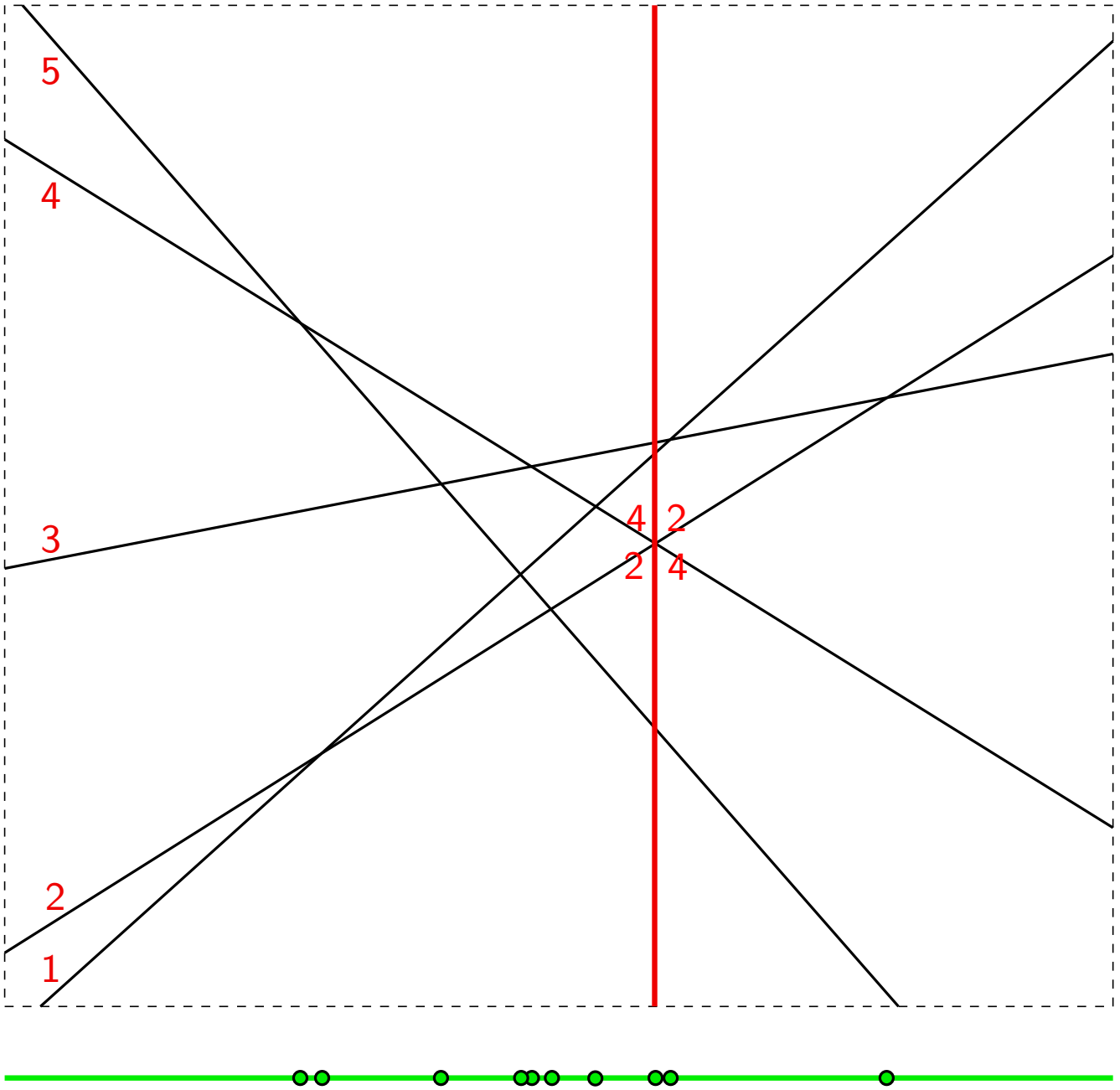


COMPUTATION

Sweep line algorithm

Advance: at each event,

1. Swap the order of the two intersecting lines in the sweep line.
2. Compute the intersection points with the two new neighboring lines.
3. If the intersection points are located forward in the sweeping order, add them to the events queue, if needed.
4. Update the DCEL.

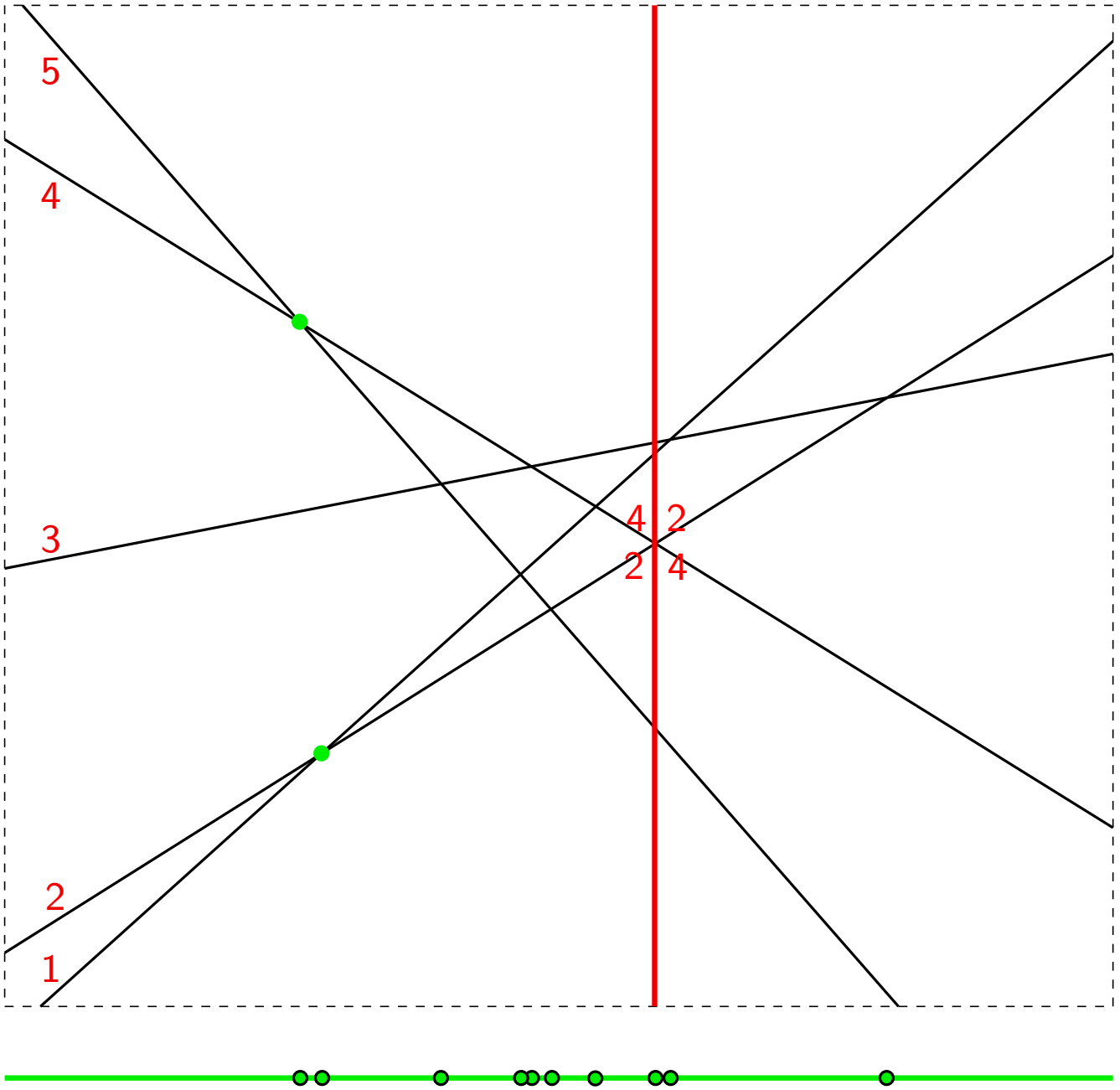


COMPUTATION

Sweep line algorithm

Advance: at each event,

1. Swap the order of the two intersecting lines in the sweep line.
2. Compute the intersection points with the two new neighboring lines.
3. If the intersection points are located forward in the sweeping order, add them to the events queue, if needed.
4. Update the DCEL.

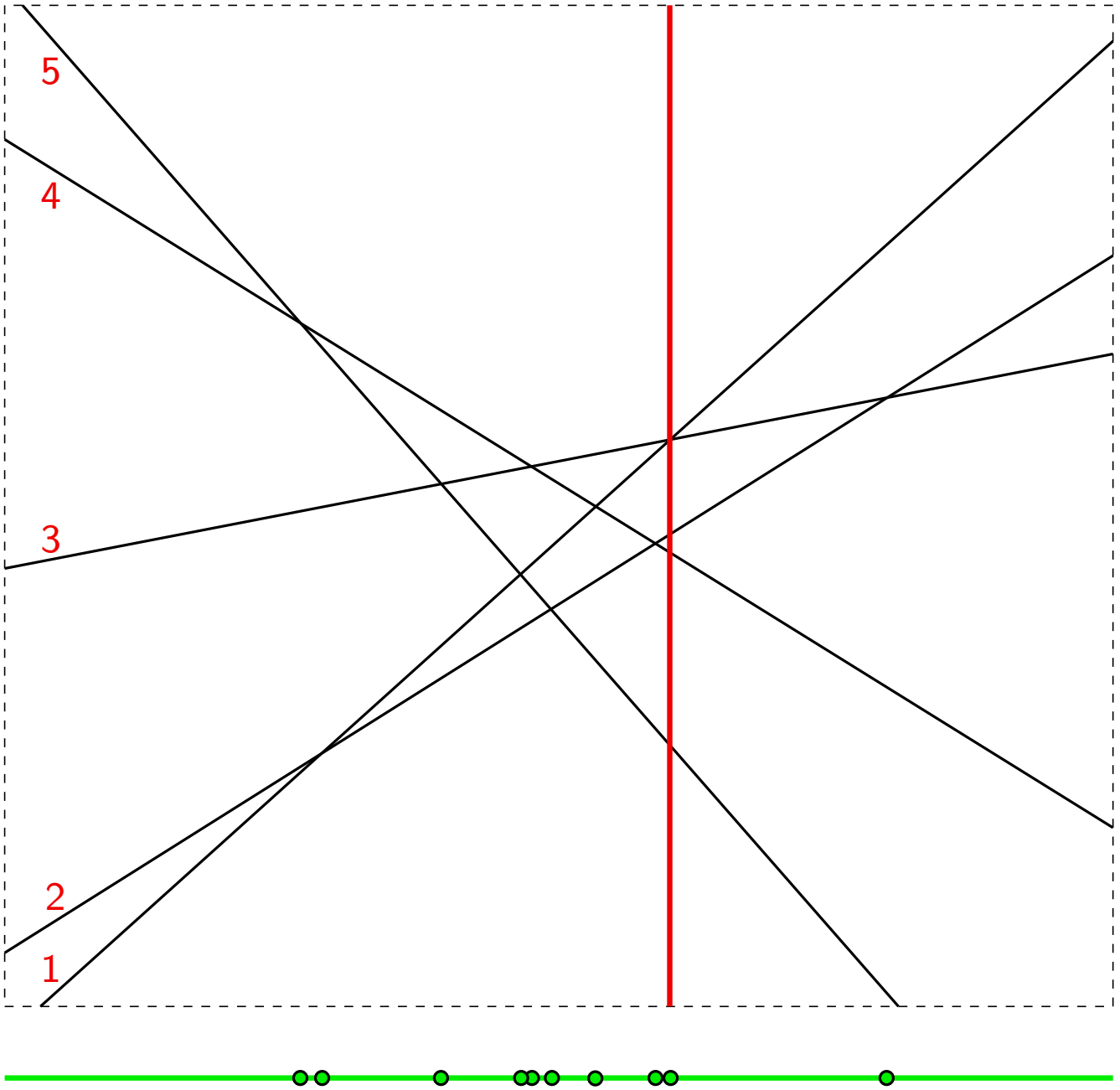


COMPUTATION

Sweep line algorithm

Advance: at each event,

1. Swap the order of the two intersecting lines in the sweep line.
2. Compute the intersection points with the two new neighboring lines.
3. If the intersection points are located forward in the sweeping order, add them to the events queue, if needed.
4. Update the DCEL.



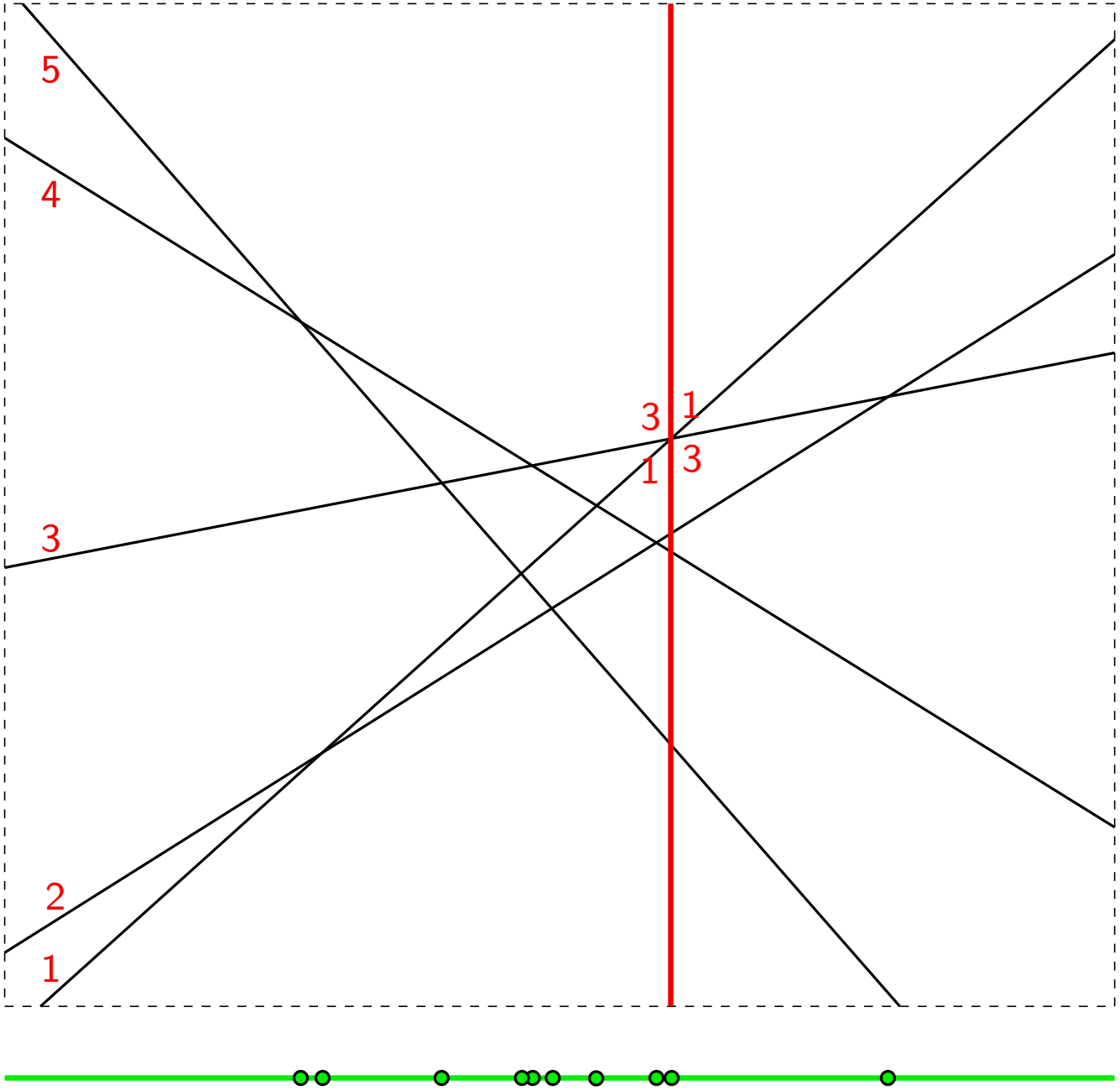
ARRANGEMENTS

COMPUTATION

Sweep line algorithm

Advance: at each event,

1. Swap the order of the two intersecting lines in the sweep line.
2. Compute the intersection points with the two new neighboring lines.
3. If the intersection points are located forward in the sweeping order, add them to the events queue, if needed.
4. Update the DCEL.

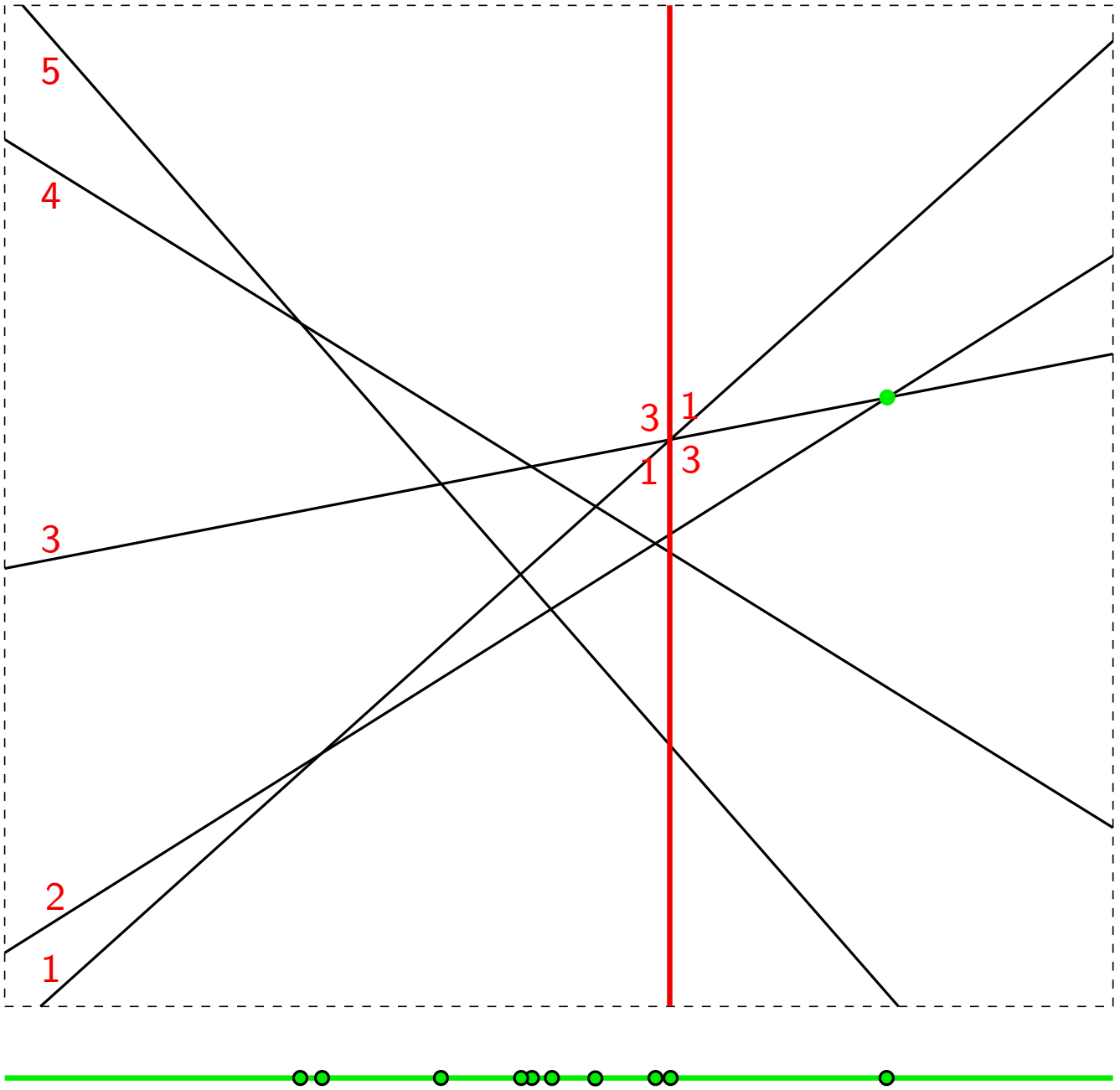


COMPUTATION

Sweep line algorithm

Advance: at each event,

1. Swap the order of the two intersecting lines in the sweep line.
2. Compute the intersection points with the two new neighboring lines.
3. If the intersection points are located forward in the sweeping order, add them to the events queue, if needed.
4. Update the DCEL.

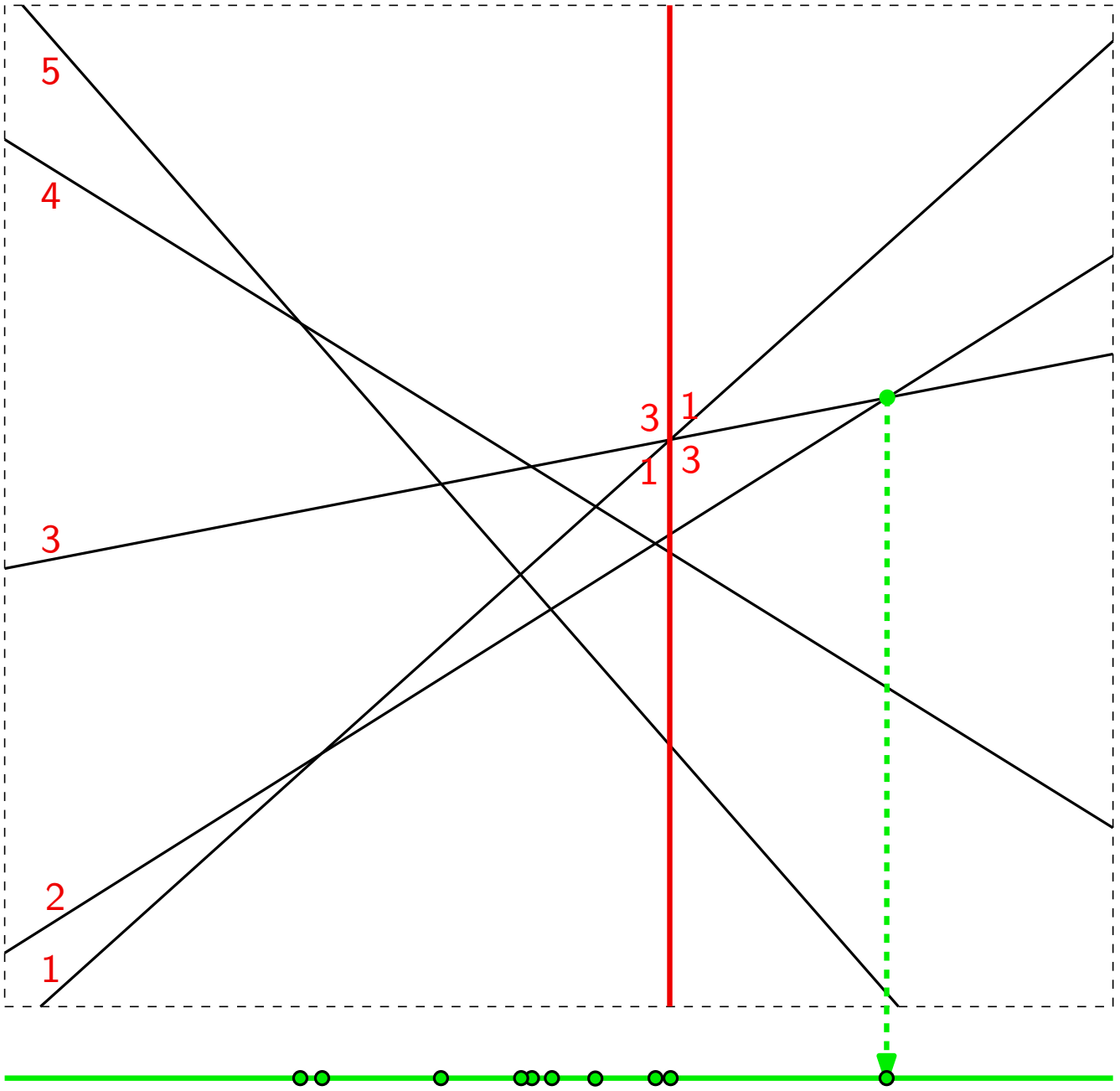


COMPUTATION

Sweep line algorithm

Advance: at each event,

1. Swap the order of the two intersecting lines in the sweep line.
2. Compute the intersection points with the two new neighboring lines.
3. If the intersection points are located forward in the sweeping order, add them to the events queue, if needed.
4. Update the DCEL.

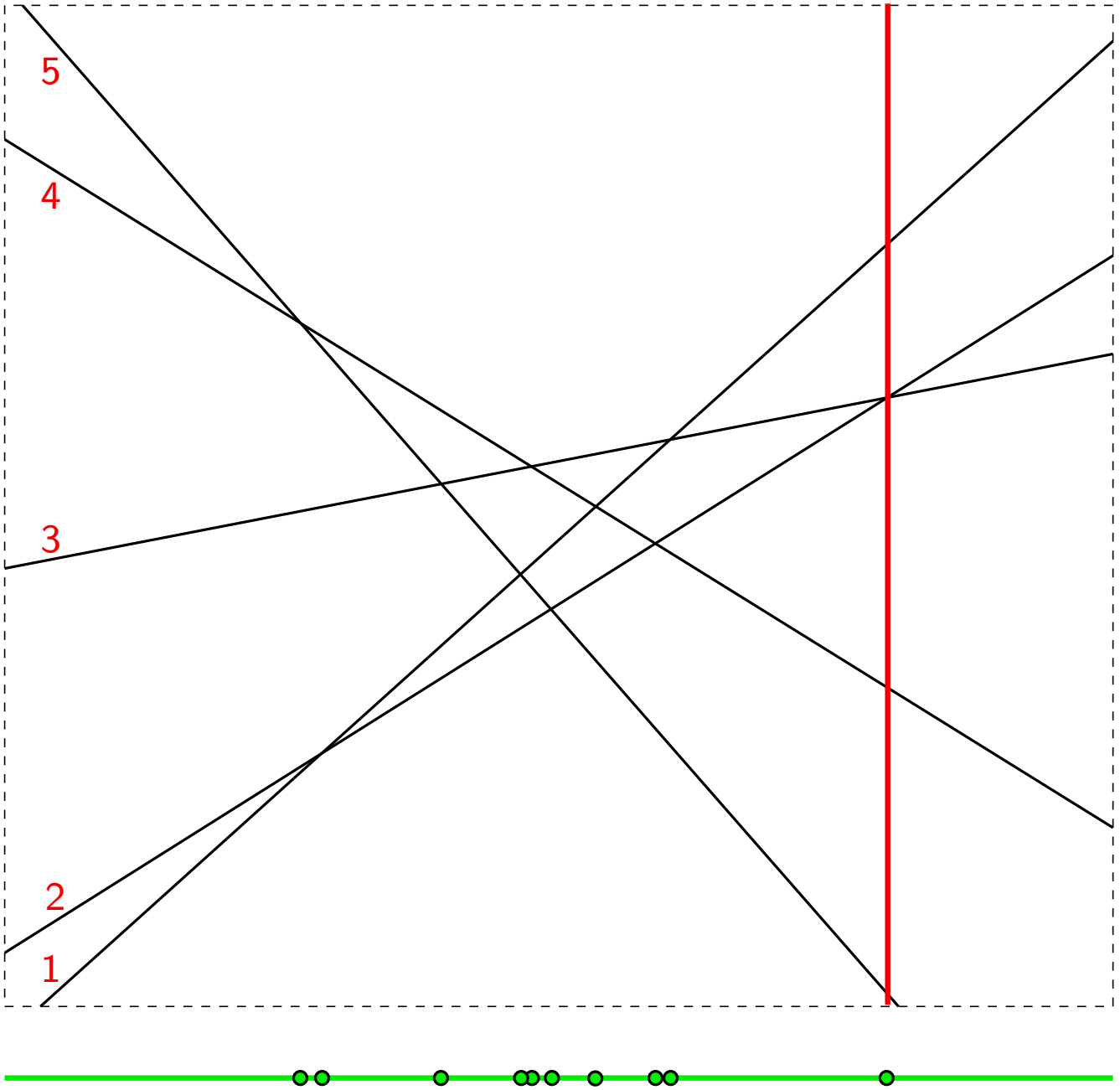


COMPUTATION

Sweep line algorithm

Advance: at each event,

1. Swap the order of the two intersecting lines in the sweep line.
2. Compute the intersection points with the two new neighboring lines.
3. If the intersection points are located forward in the sweeping order, add them to the events queue, if needed.
4. Update the DCEL.

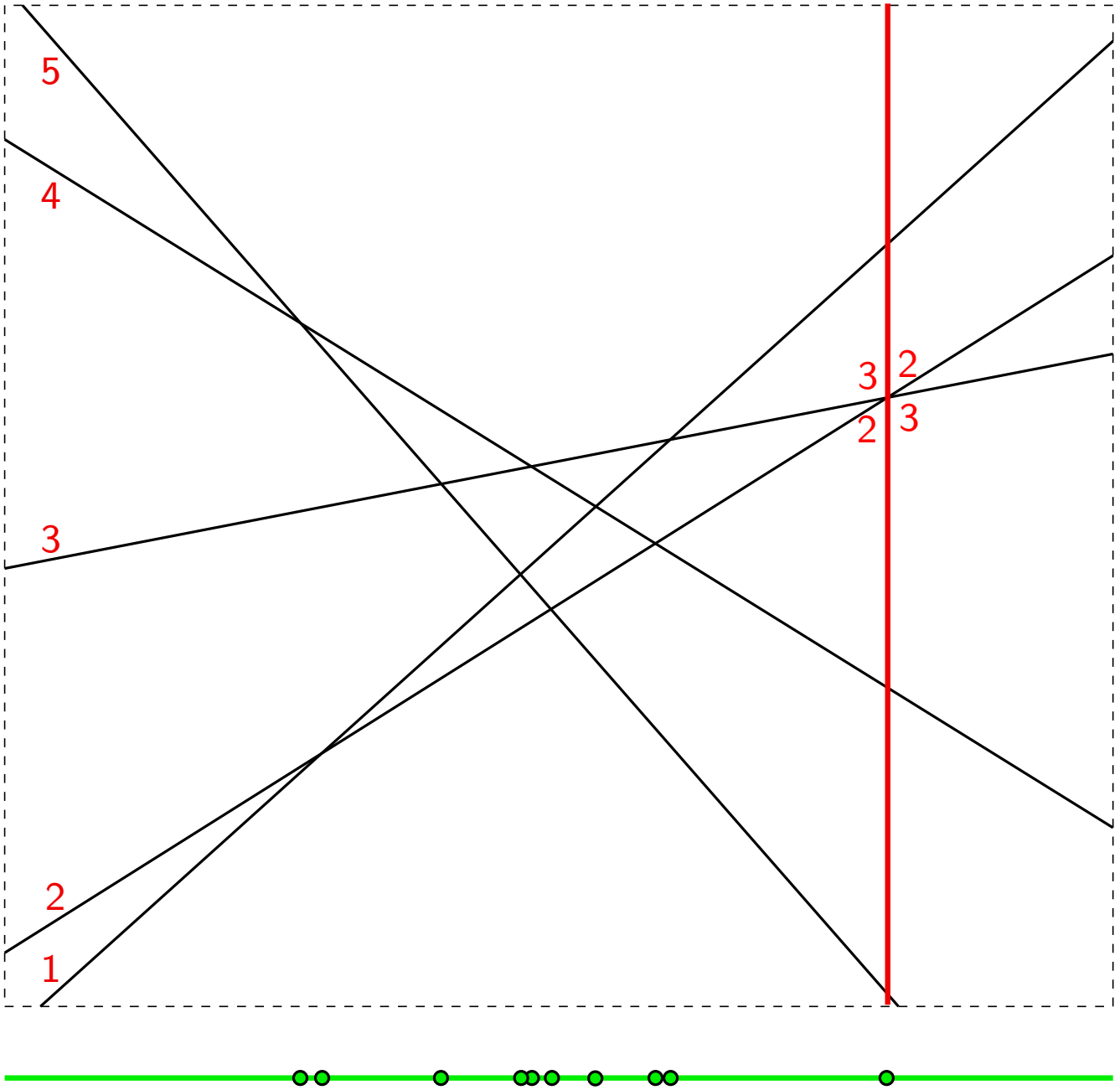


COMPUTATION

Sweep line algorithm

Advance: at each event,

1. Swap the order of the two intersecting lines in the sweep line.
2. Compute the intersection points with the two new neighboring lines.
3. If the intersection points are located forward in the sweeping order, add them to the events queue, if needed.
4. Update the DCEL.

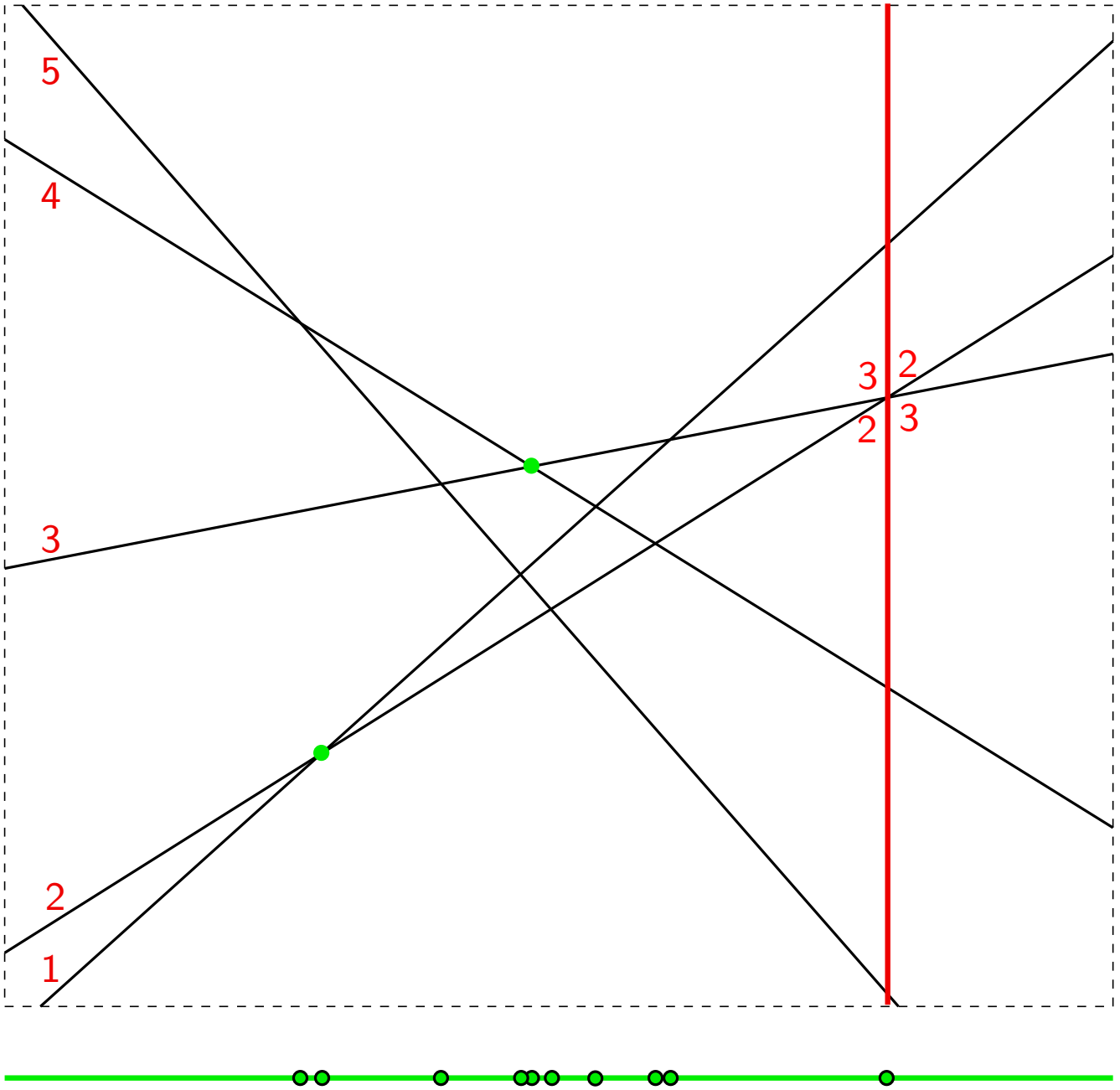


COMPUTATION

Sweep line algorithm

Advance: at each event,

1. Swap the order of the two intersecting lines in the sweep line.
2. Compute the intersection points with the two new neighboring lines.
3. If the intersection points are located forward in the sweeping order, add them to the events queue, if needed.
4. Update the DCEL.

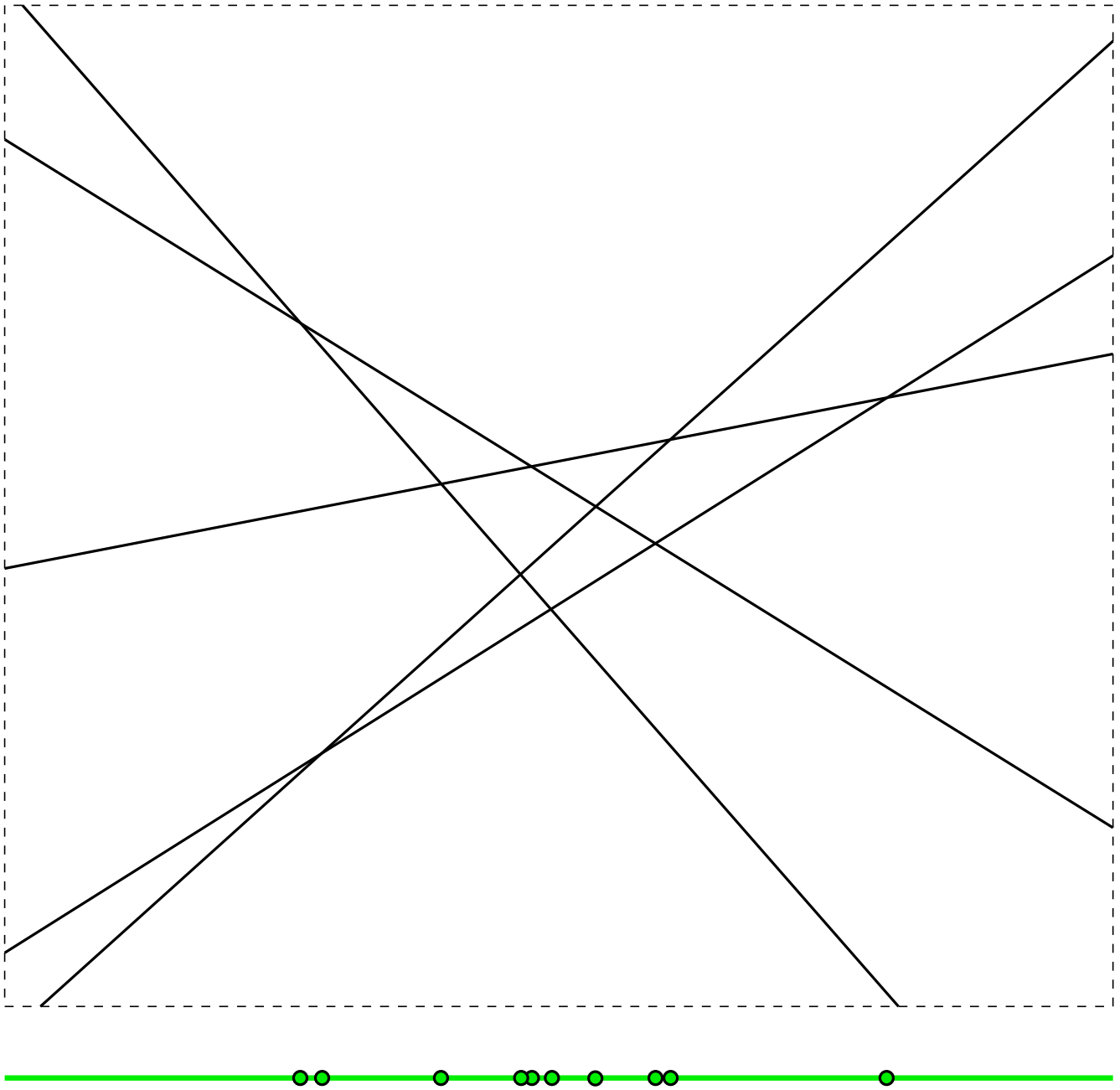


COMPUTATION

Sweep line algorithm

Advance: at each event,

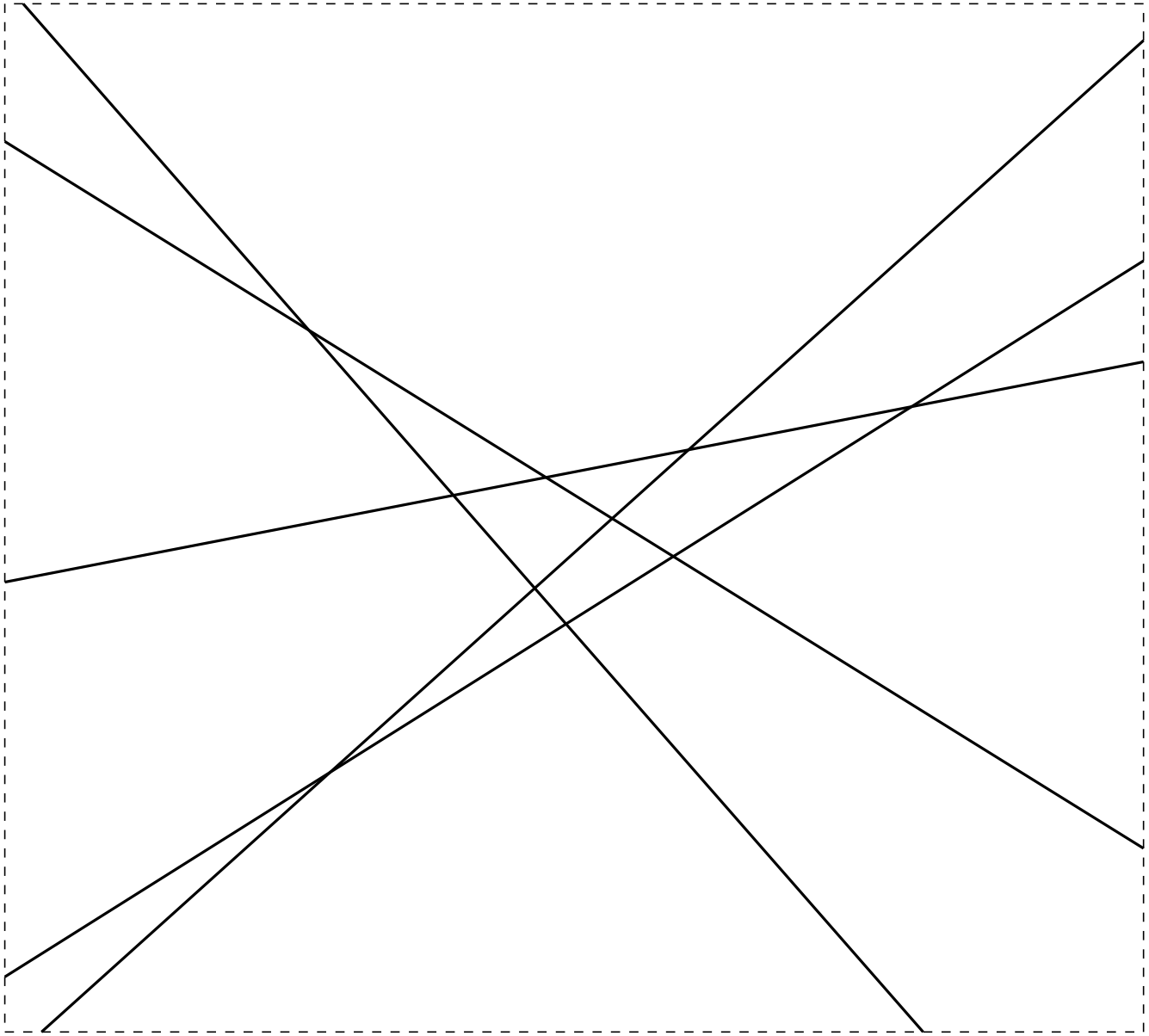
1. Swap the order of the two intersecting lines in the sweep line.
2. Compute the intersection points with the two new neighboring lines.
3. If the intersection points are located forward in the sweeping order, add them to the events queue, if needed.
4. Update the DCEL.



COMPUTATION

Sweep line algorithm

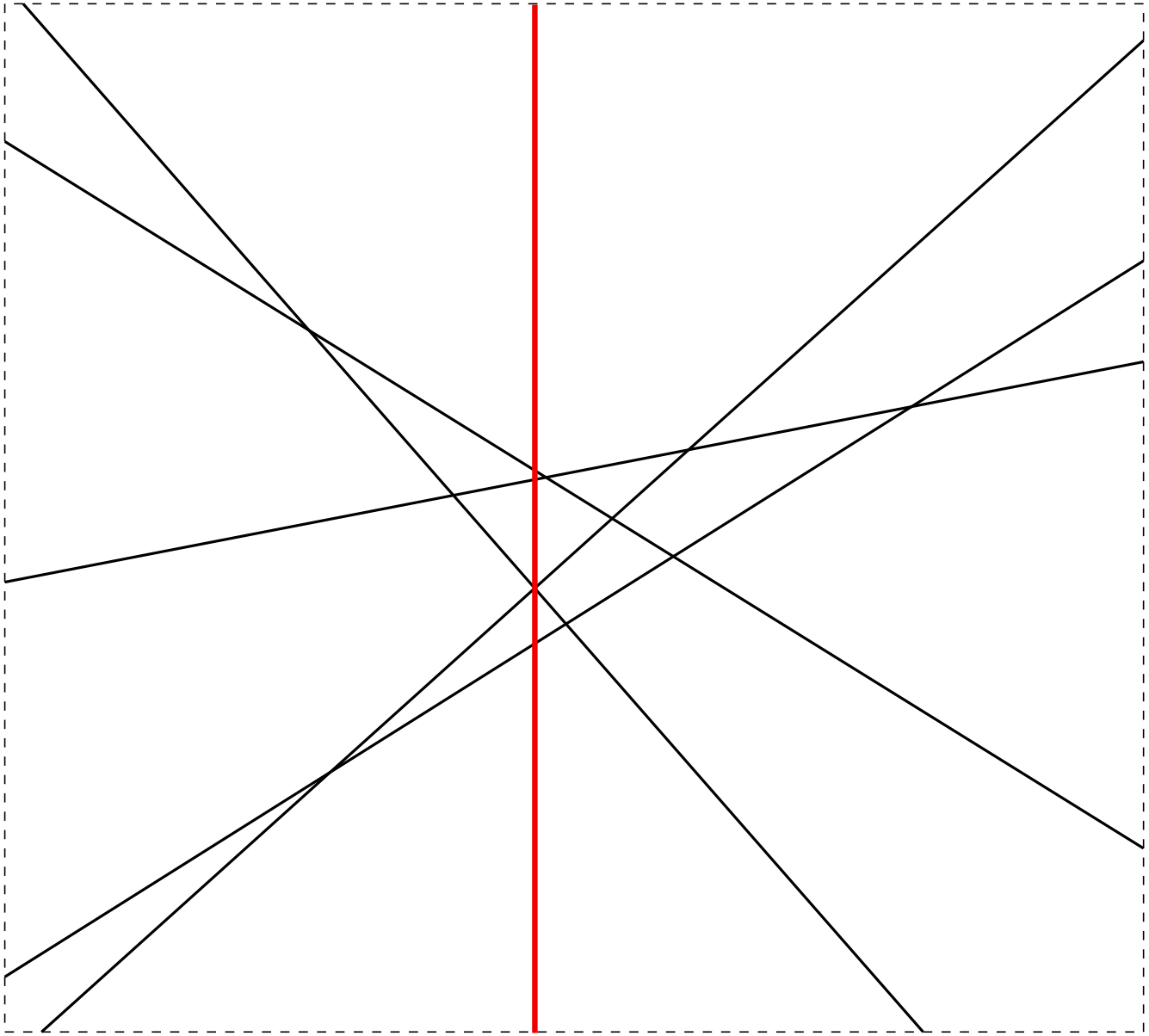
Updating the DCEL:



COMPUTATION

Sweep line algorithm

Updating the DCEL:

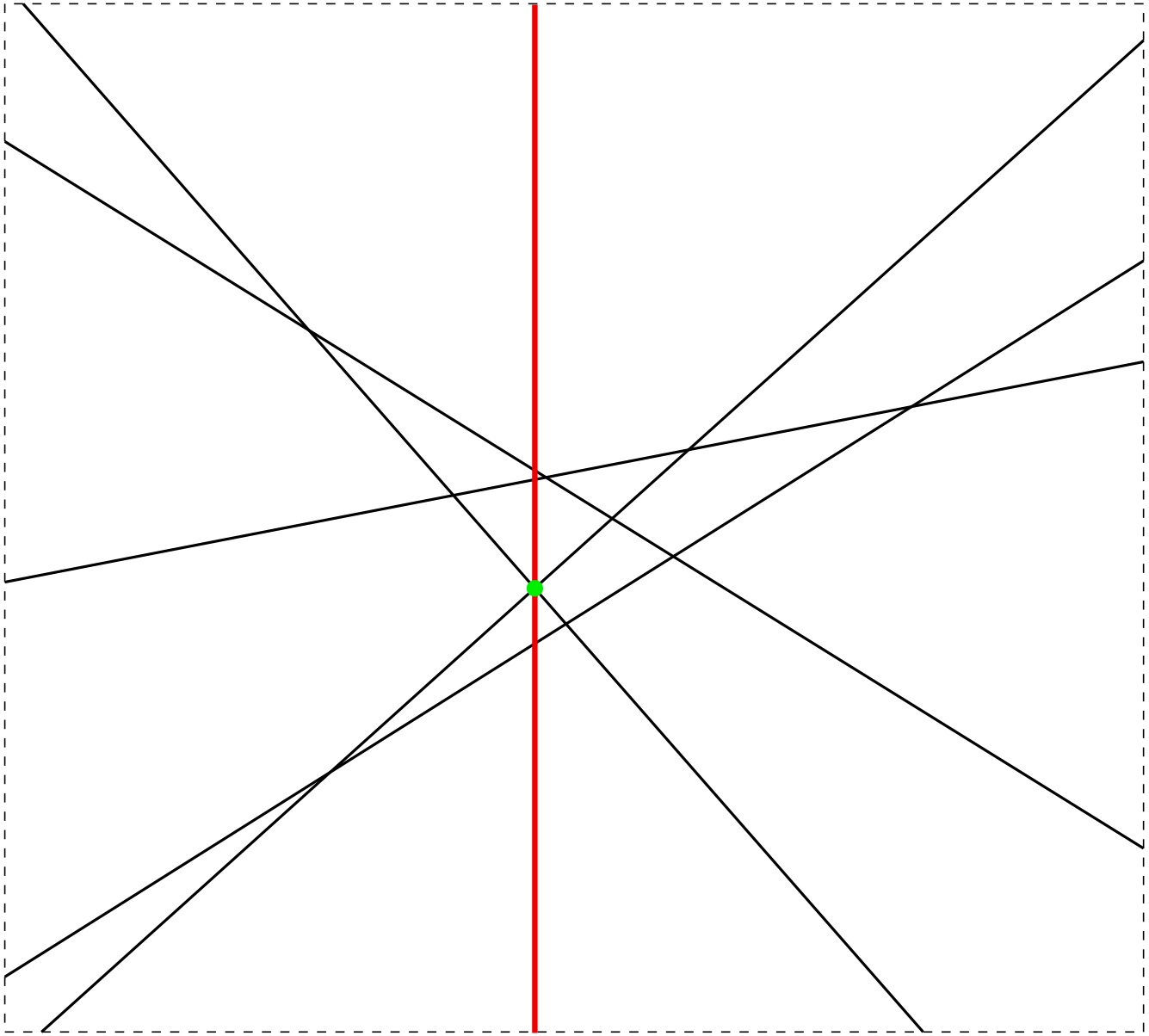


COMPUTATION

Sweep line algorithm

Updating the DCEL:

- Add the vertex to the table of vertices, with its coordinates and a pointer to an edge.

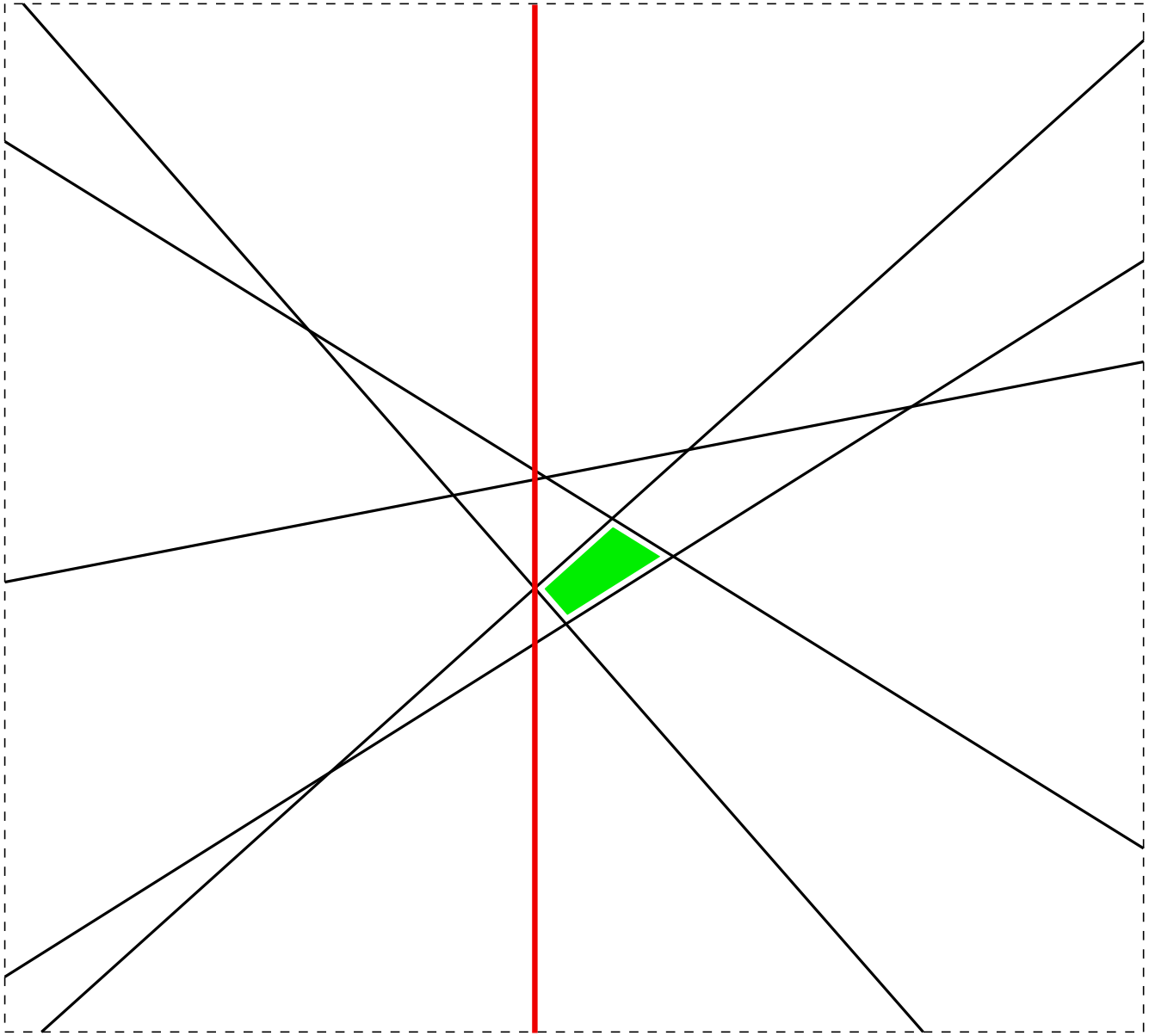


COMPUTATION

Sweep line algorithm

Updating the DCEL:

- Add the vertex to the table of vertices, with its coordinates and a pointer to an edge.
- Add the starting face to the table of faces, with a pointer to an edge.

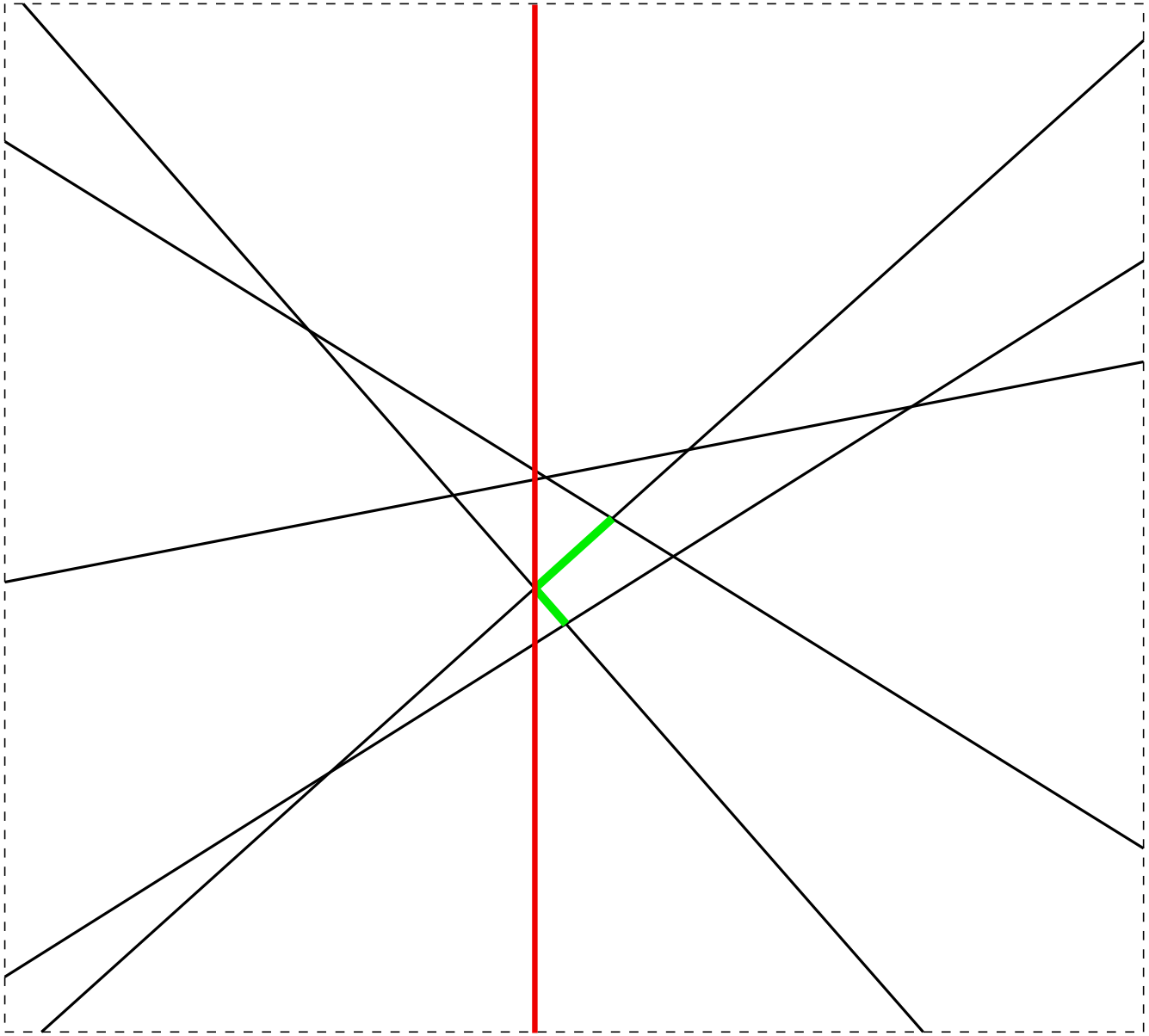


COMPUTATION

Sweep line algorithm

Updating the DCEL:

- Add the vertex to the table of vertices, with its coordinates and a pointer to an edge.
- Add the starting face to the table of faces, with a pointer to an edge.
- Add to the DCEL the two starting edges, with pointers to $v_B = v$, e_P, f_L and f_R.

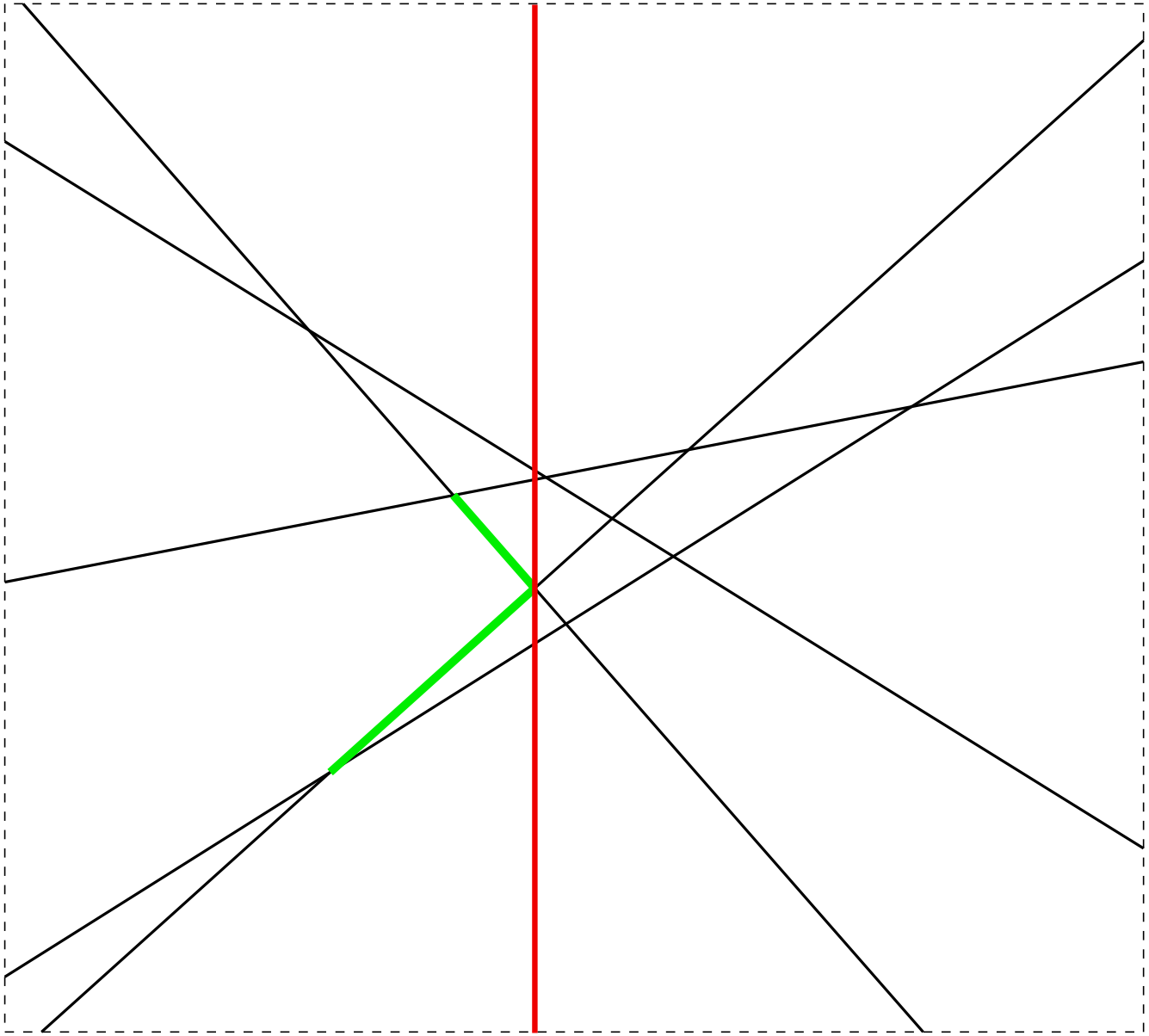


COMPUTATION

Sweep line algorithm

Updating the DCEL:

- Add the vertex to the table of vertices, with its coordinates and a pointer to an edge.
- Add the starting face to the table of faces, with a pointer to an edge.
- Add to the DCEL the two starting edges, with pointers to $v_B = v$, e_P, f_L and f_R.
- Add to the DCEL pointers to $v_E = v$ and e_N, corresponding to the two ending edges.



COMPUTATION

Sweep line algorithm

Correctness:

All intersection points are found because:

- Any two lines must be adjacent in the sweeping line immediatly before intersecting.
- Adjacent lines in the sweep line are always checked for intersection.

COMPUTATION

Sweep line algorithm

Running time:

Initialization: $O(n \log n)$

At each event (there are $O(n^2)$ events):

1. Locate and swap: $O(\log n)$
2. Intersect: $O(1)$
3. Insert: $O(\log n^2) = O(\log n)$
4. Update the DCEL: $O(1)$

Total running time: $O(n^2 \log n)$

COMPUTATION

Sweep line algorithm

Running time:

Initialization: $O(n \log n)$

At each event (there are $O(n^2)$ events):

1. Locate and swap: $O(\log n)$
2. Intersect: $O(1)$
3. Insert: $O(\log n^2) = O(\log n)$
4. Update the DCEL: $O(1)$

Total running time: $O(n^2 \log n)$

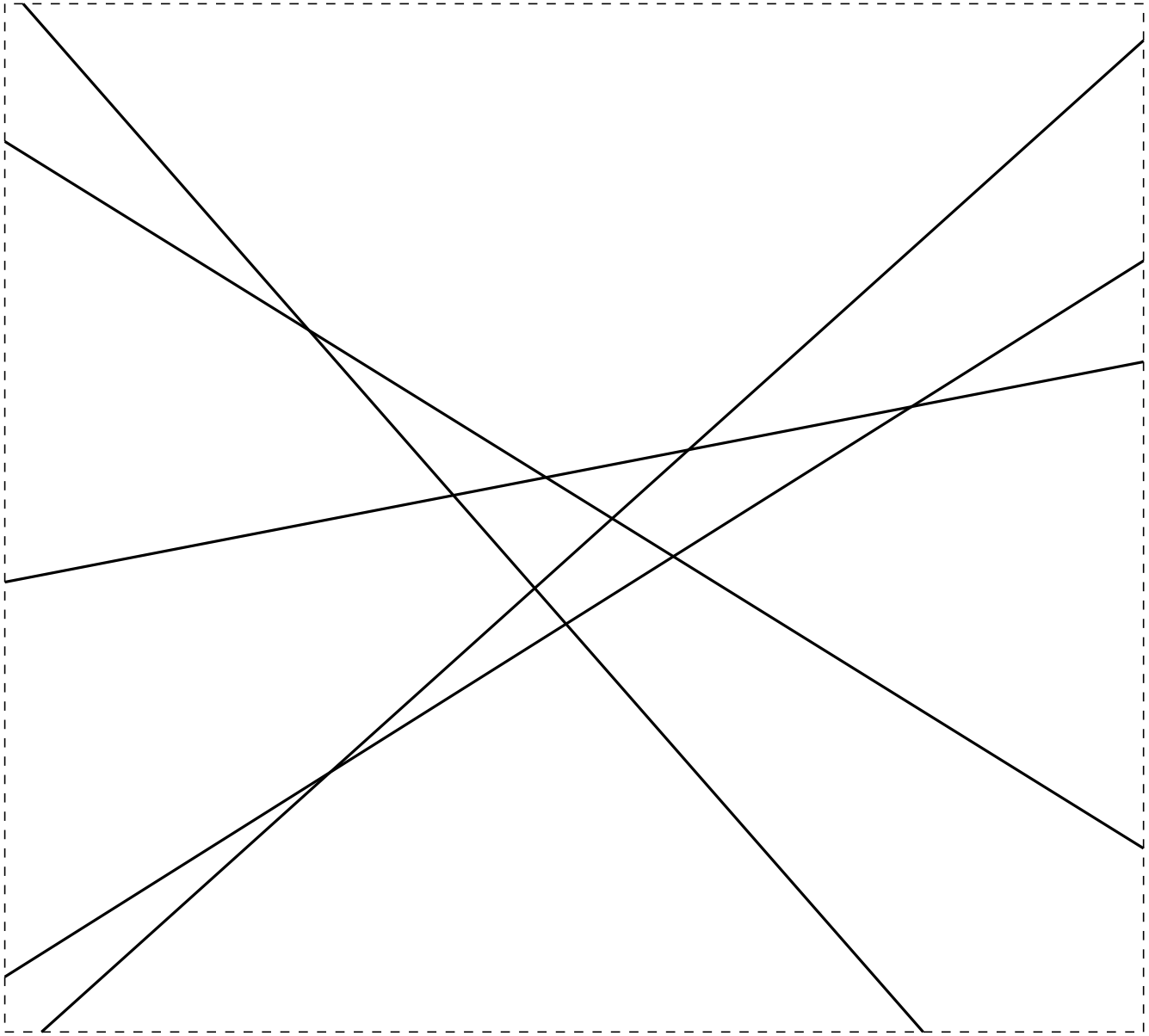
The description of the algorithm we have shown assumes that the arrangement is simple.

An easy modification allows adapting it to the general case.

ARRANGEMENTS

COMPUTATION

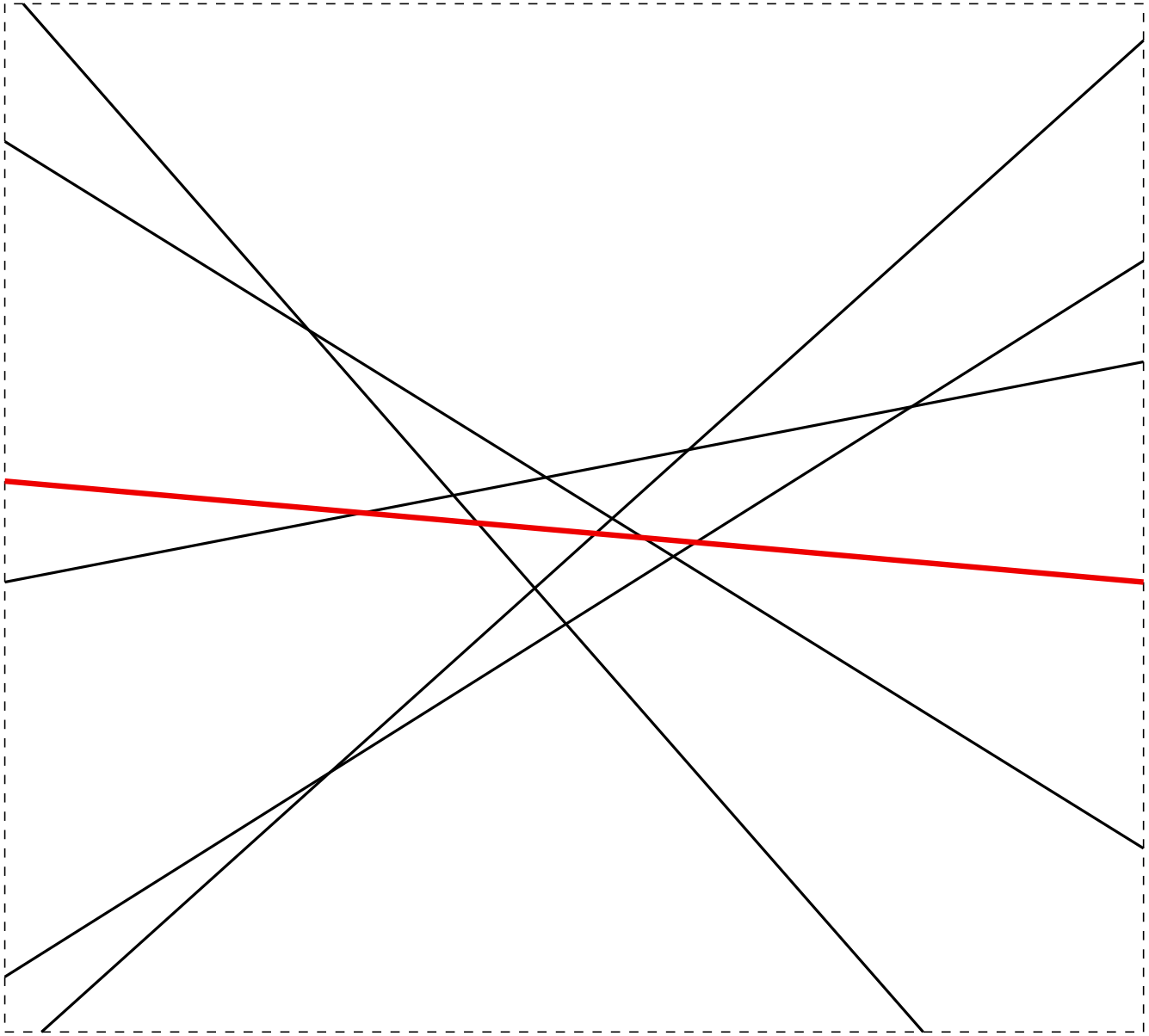
Incremental algorithm



COMPUTATION

Incremental algorithm

For each new line l:

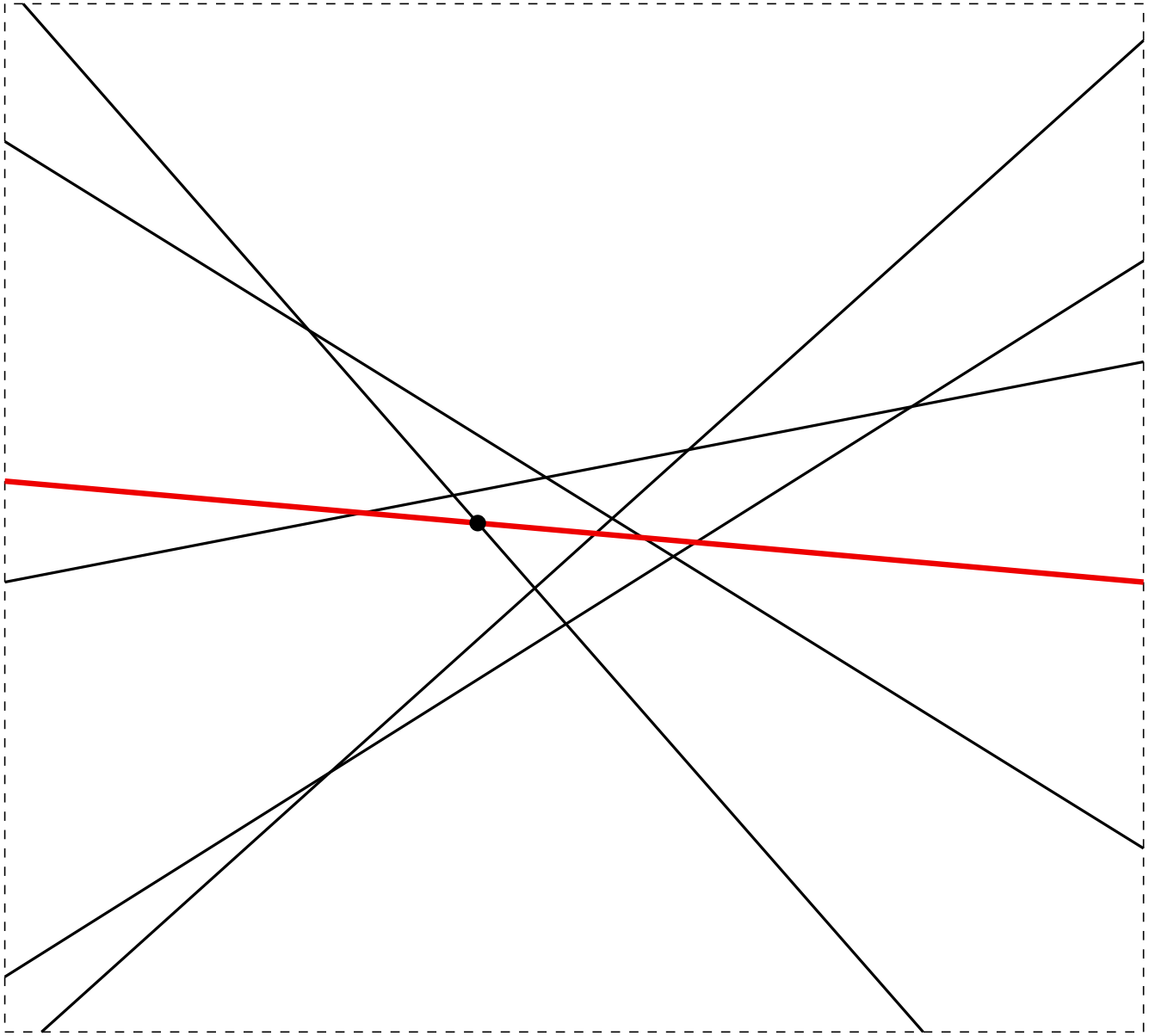


COMPUTATION

Incremental algorithm

For each new line l:

1. Intersect l with any line l_i pre existent in the arrangement.

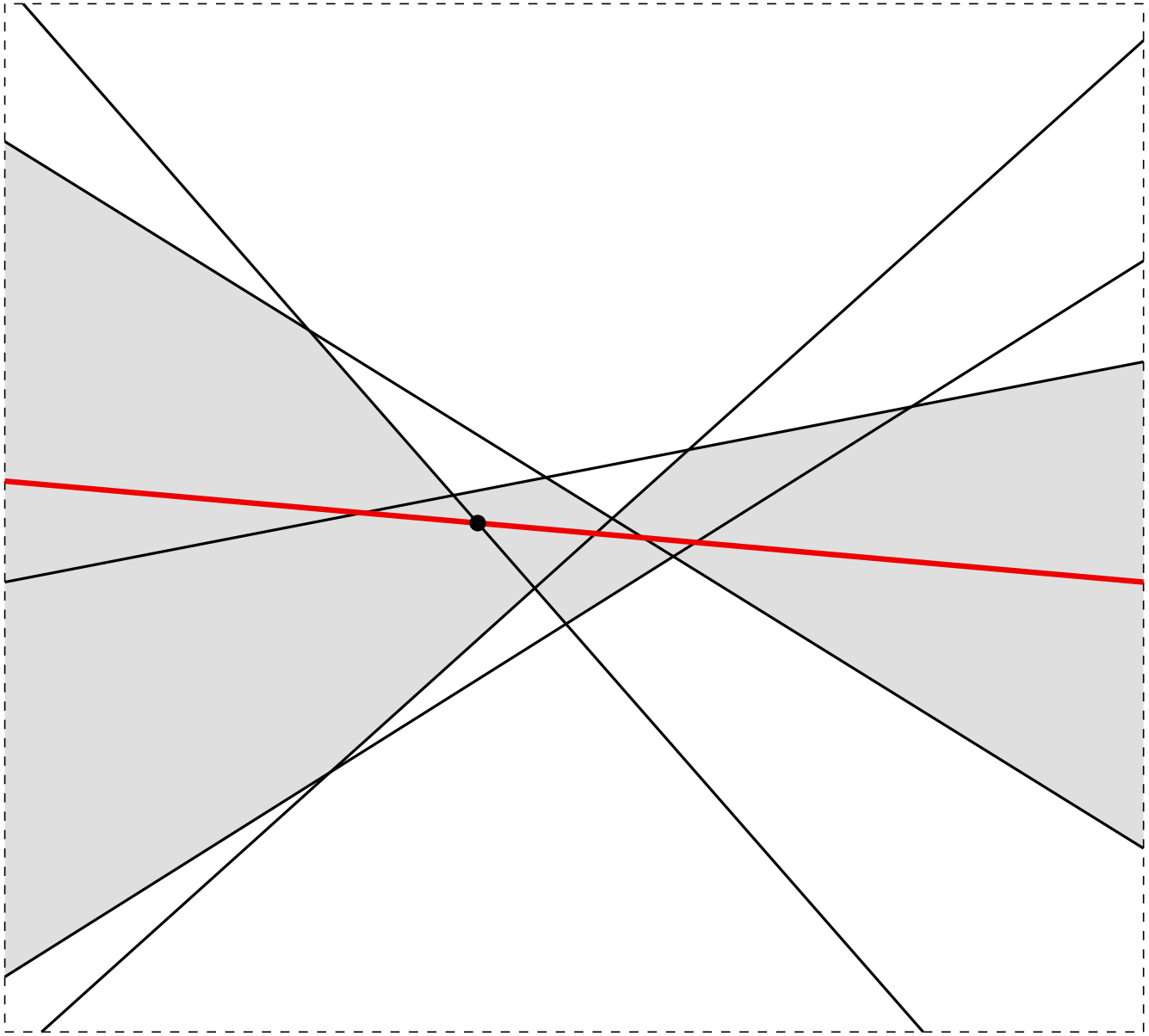


COMPUTATION

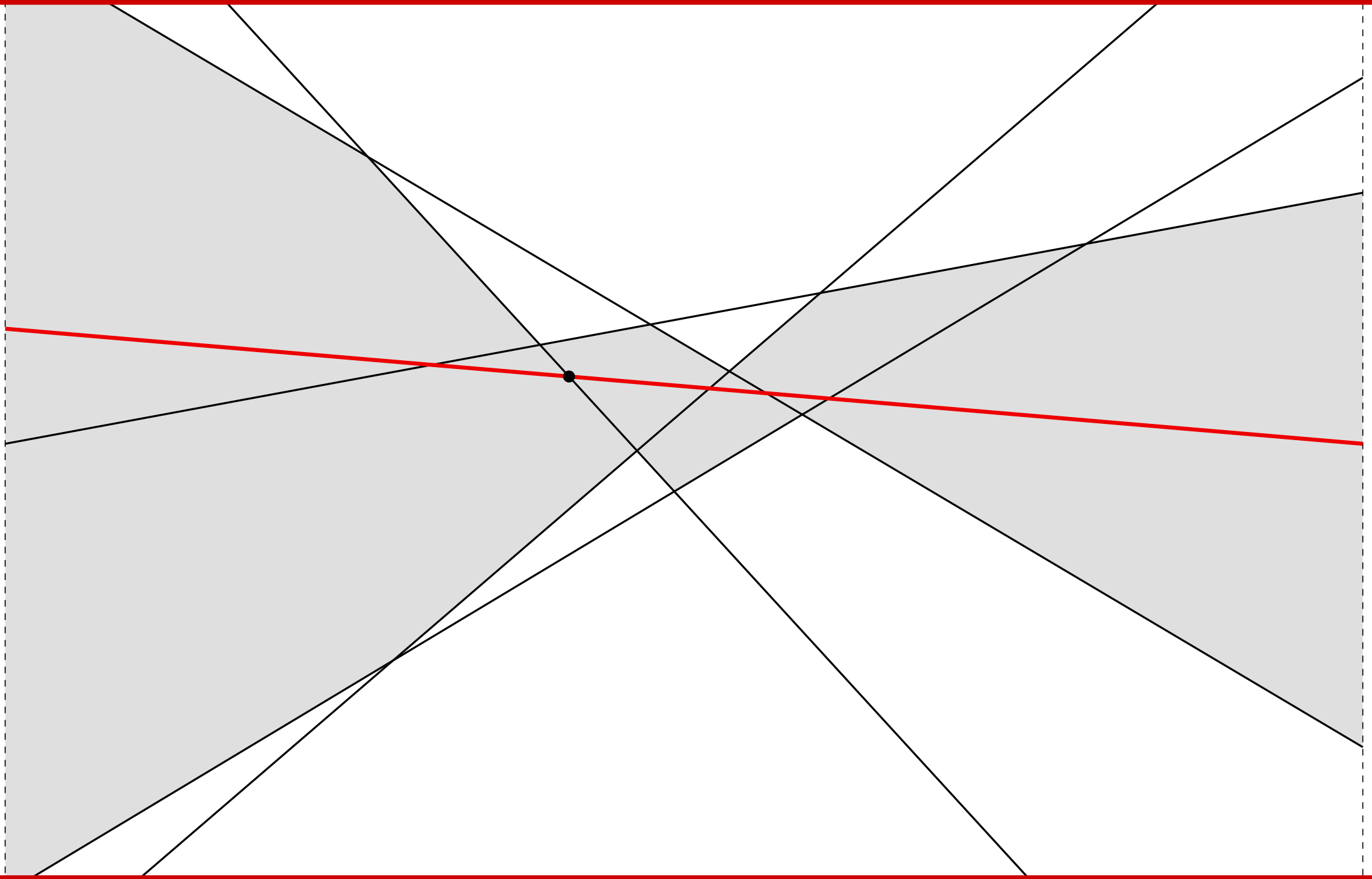
Incremental algorithm

For each new line l:

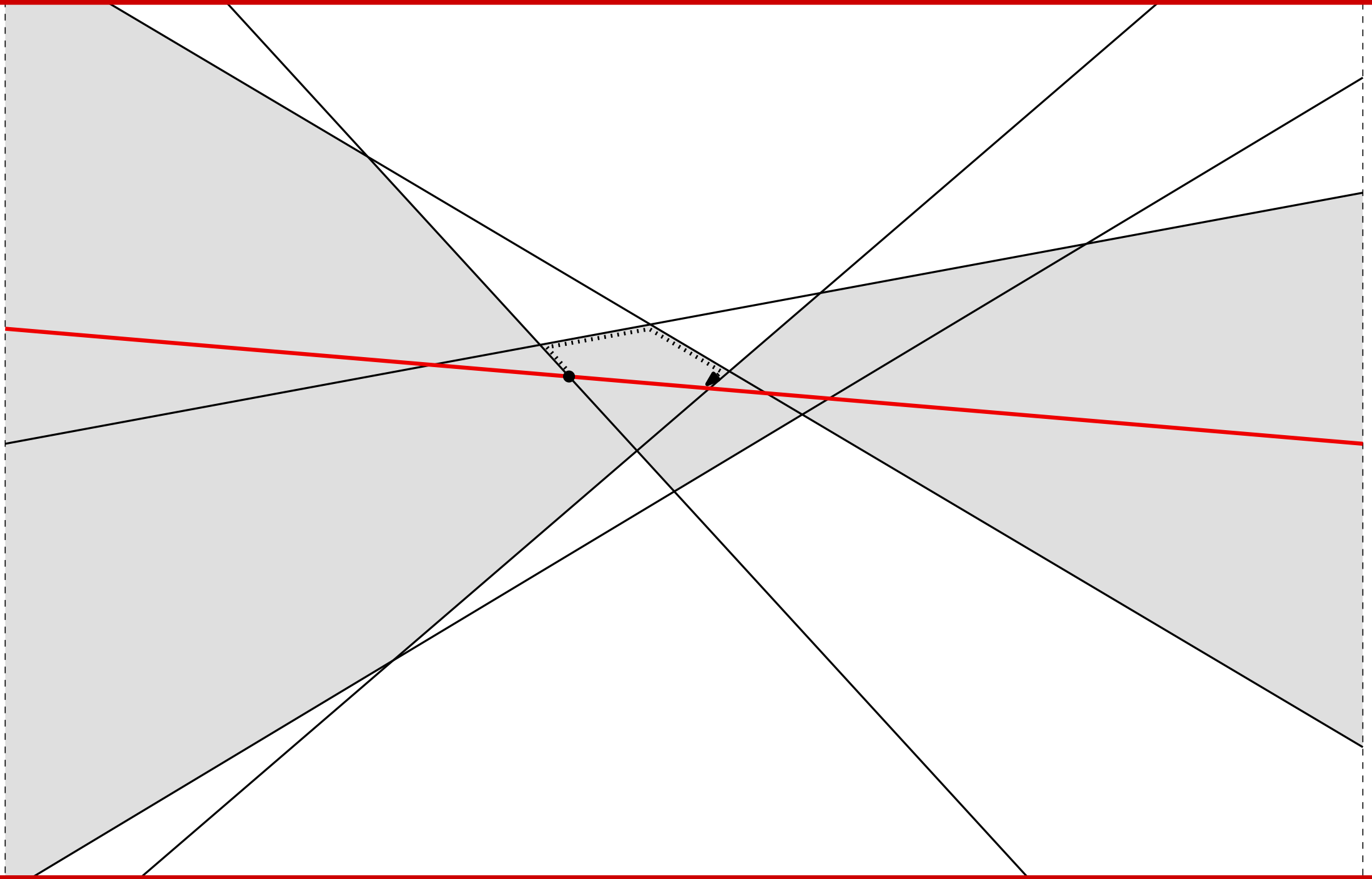
1. Intersect l with any line l_i pre existent in the arrangement.
2. Traverse the zone of l...



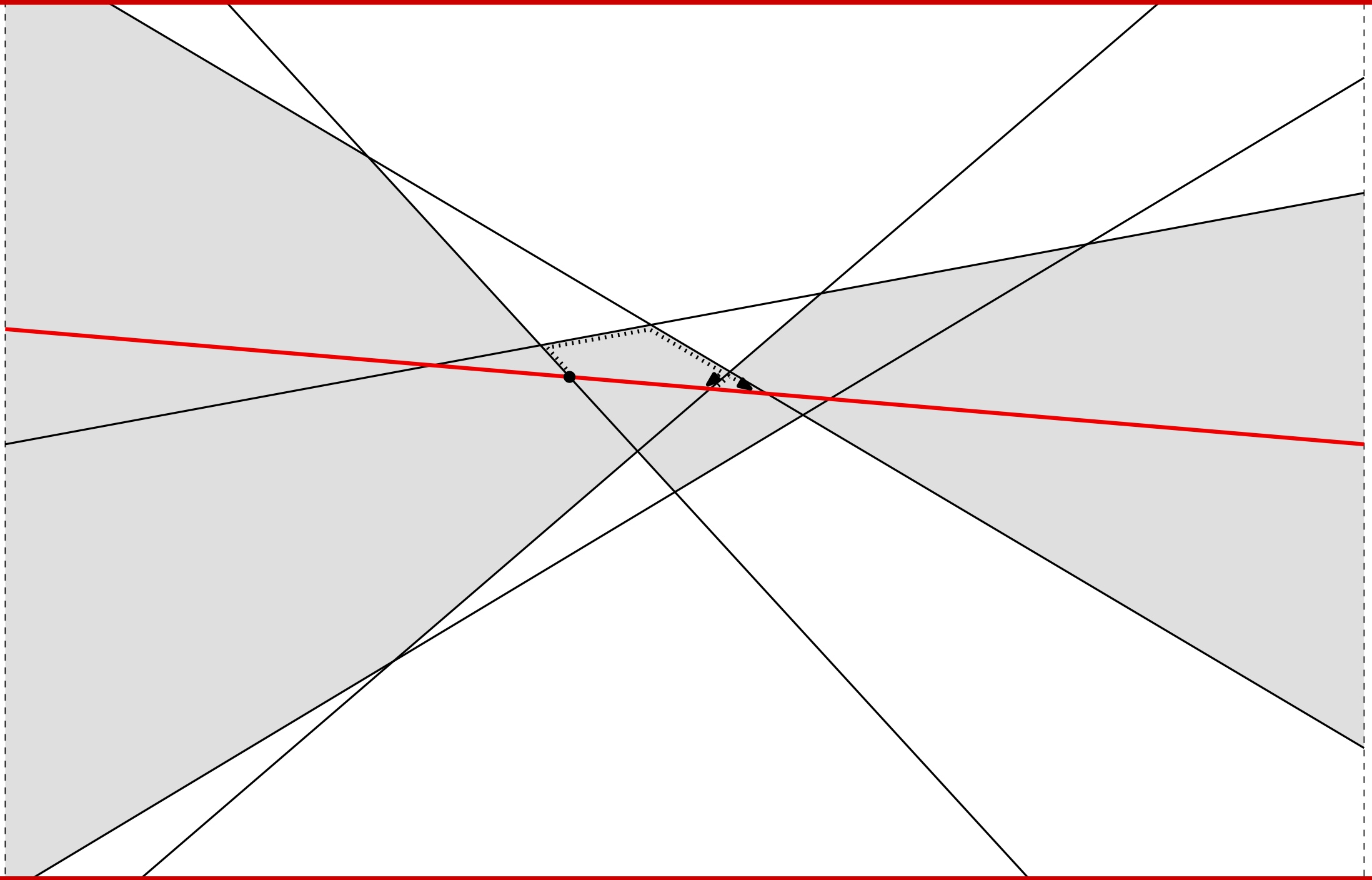
ARRANGEMENTS



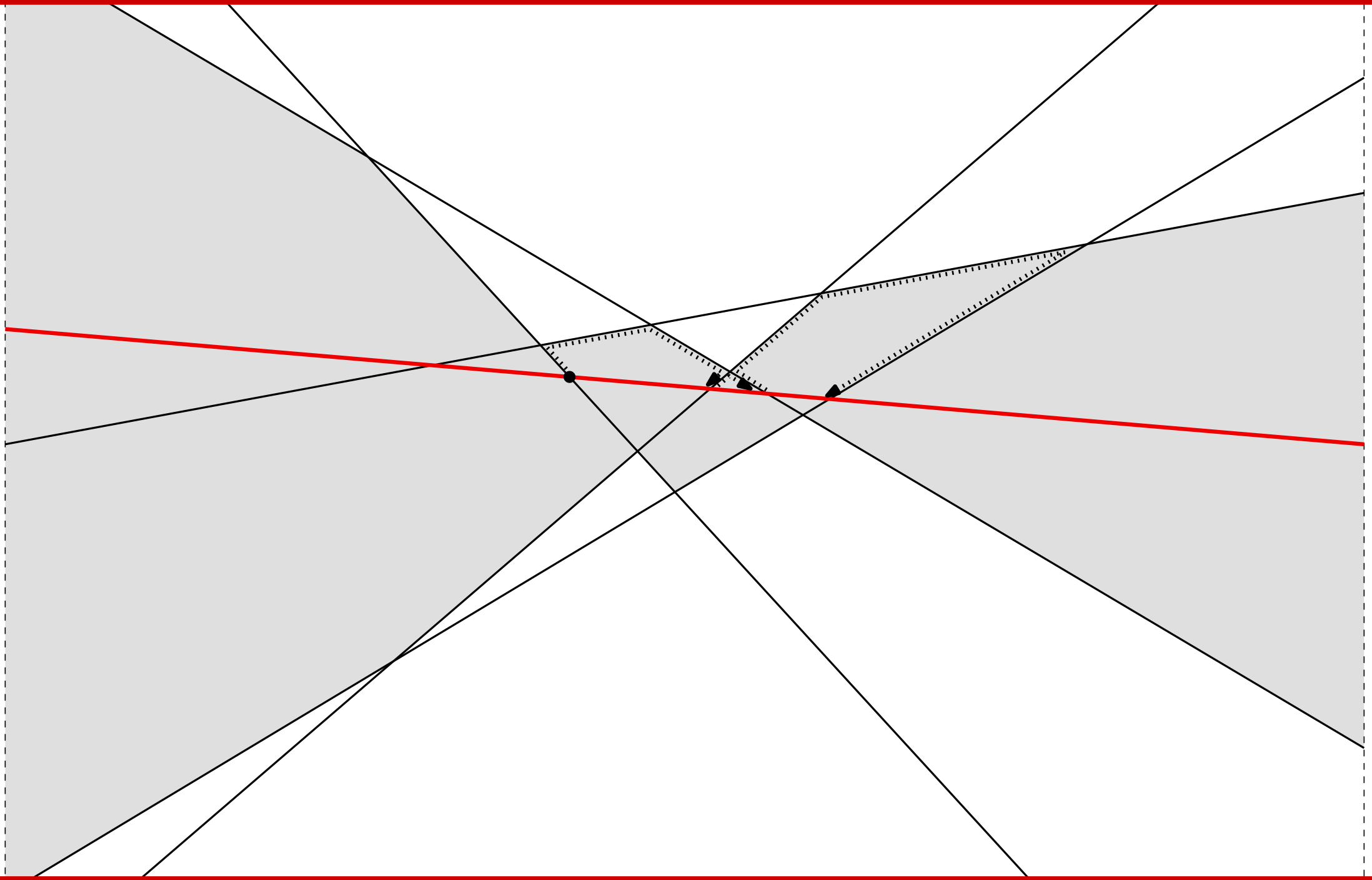
ARRANGEMENTS



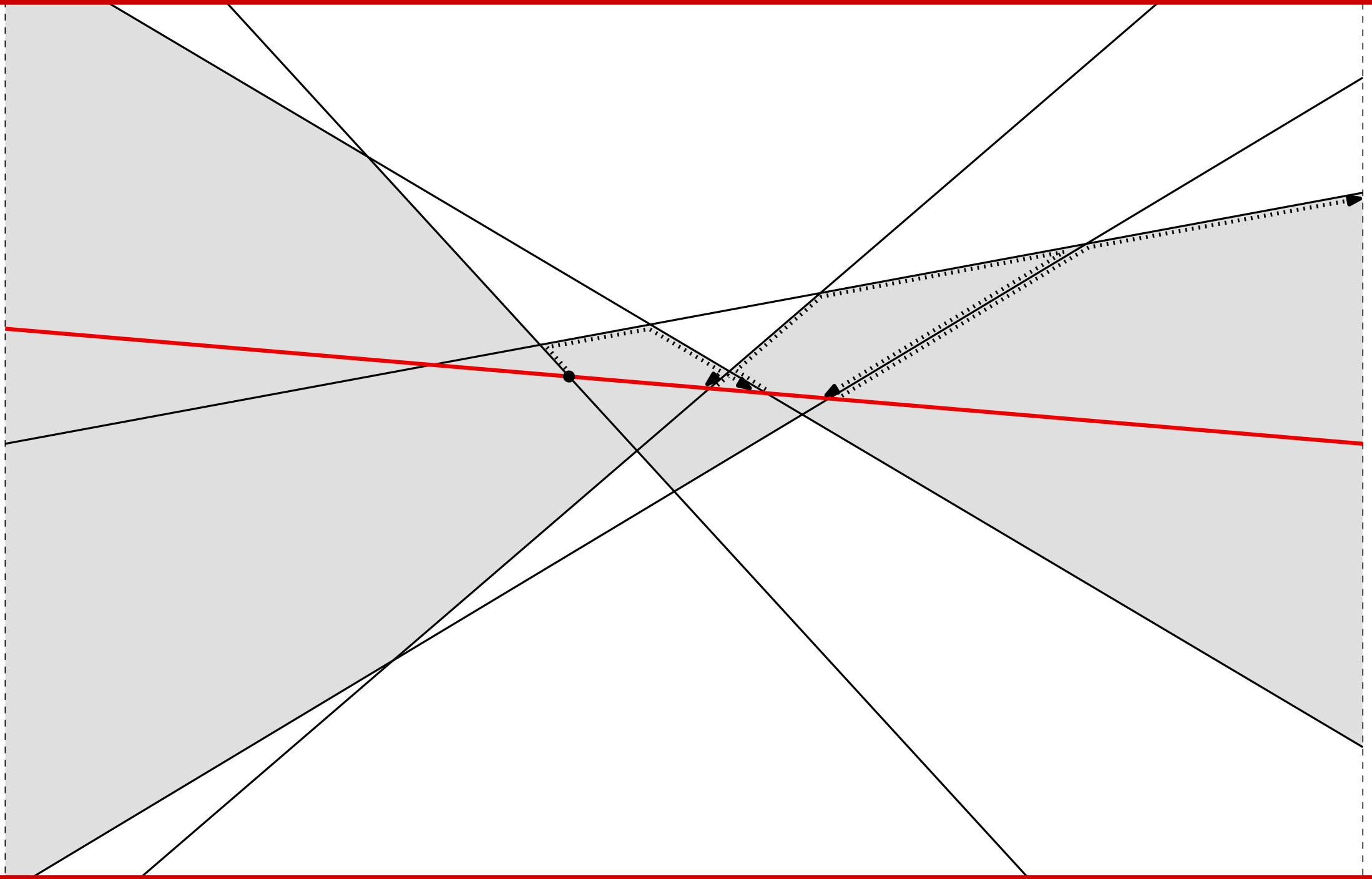
ARRANGEMENTS



ARRANGEMENTS



ARRANGEMENTS

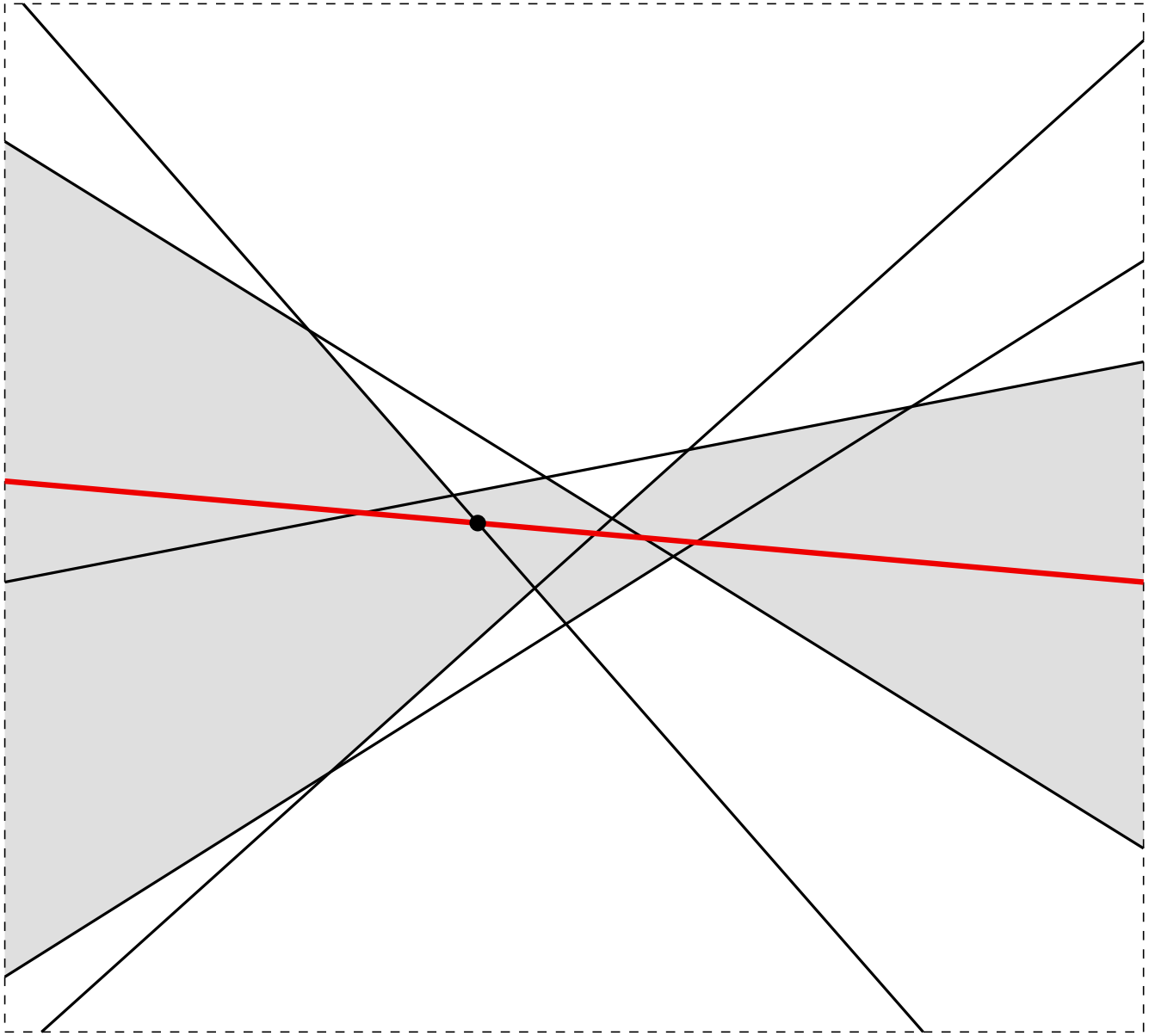


COMPUTATION

Incremental algorithm

For each new line l:

1. Intersect l with any line l_i pre existent in the arrangement.
2. Traverse the zone of l...

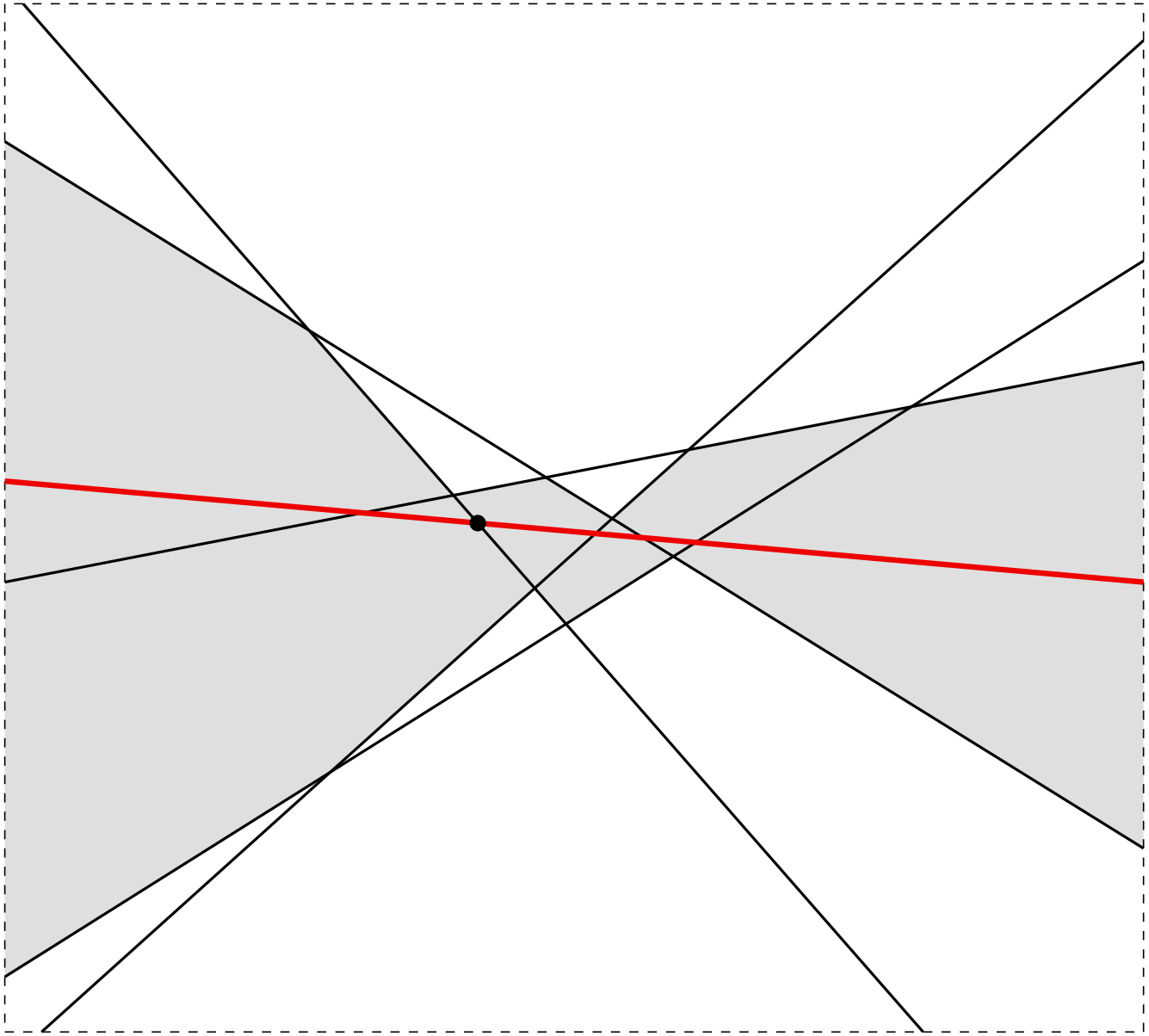


COMPUTATION

Incremental algorithm

For each new line l:

1. Intersect l with any line l_i pre existent in the arrangement.
2. Traverse the zone of l...
   ...updating the DCEL.



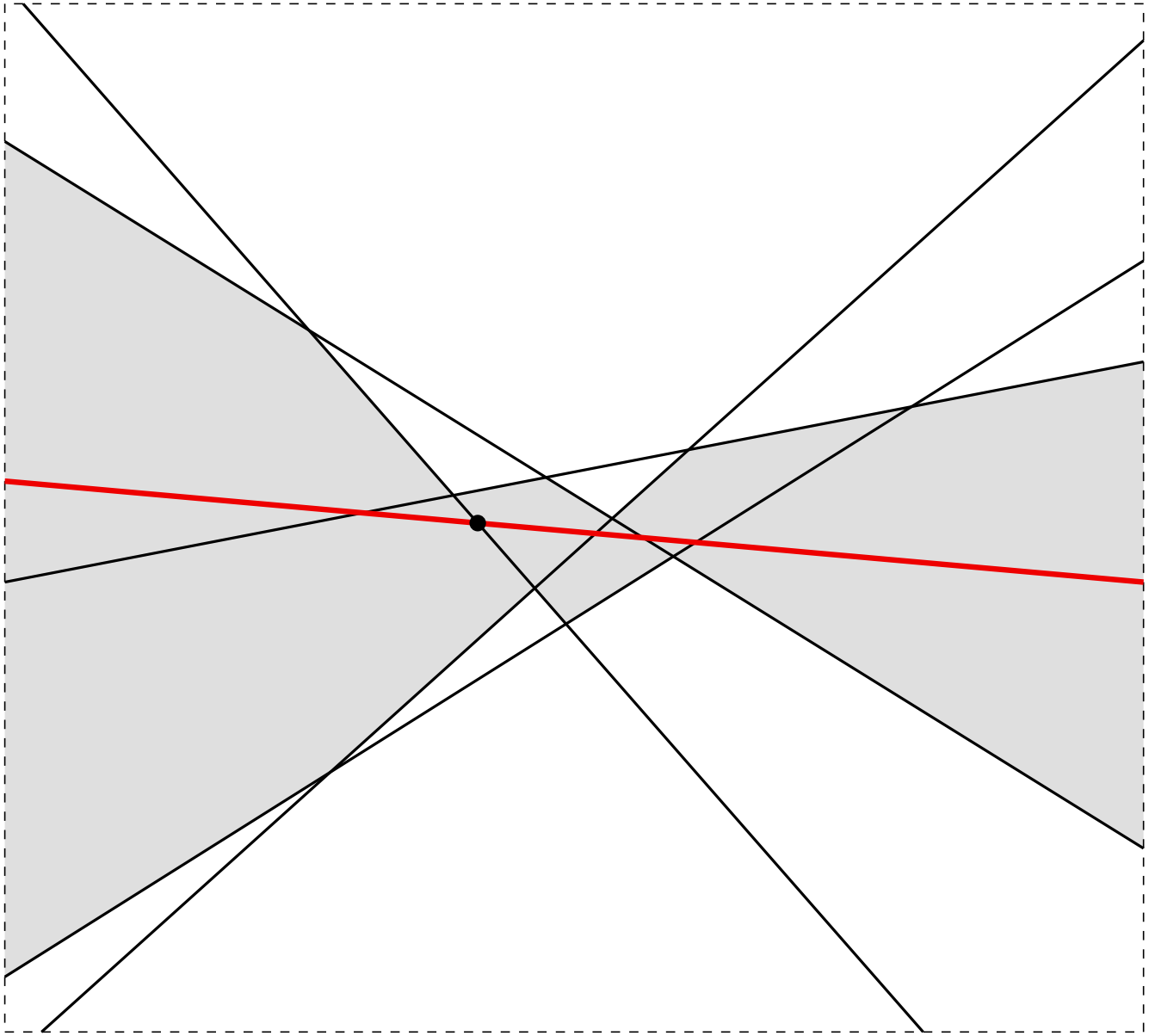
COMPUTATION

Incremental algorithm

For each new line l:

1. Intersect l with any line l_i pre existent in the arrangement.
2. Traverse the zone of l...
   ...updating the DCEL.

Updating the DCEL:



COMPUTATION

Incremental algorithm

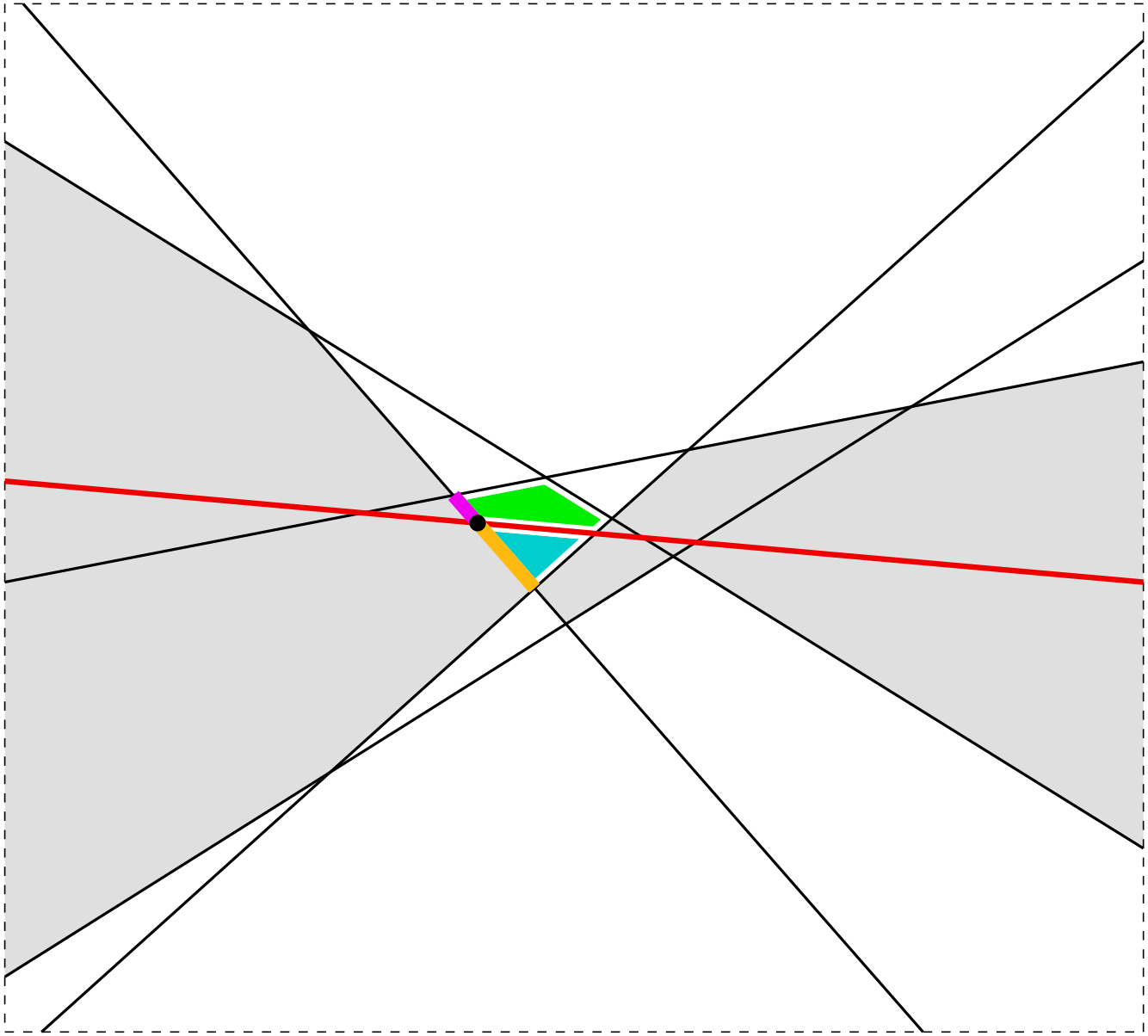
For each new line l:

1. Intersect l with any line l_i pre existent in the arrangement.
2. Traverse the zone of l...
   ...updating the DCEL.

Updating the DCEL:

At each intersection point v, the line l

- partitions an old edge e into two new edges, e_1 and e_2;
- a new edge b_1 ends and a new edge b_2 starts;
- partitions an old face f into two new faces, f_1 and f_2.



COMPUTATION

Incremental algorithm

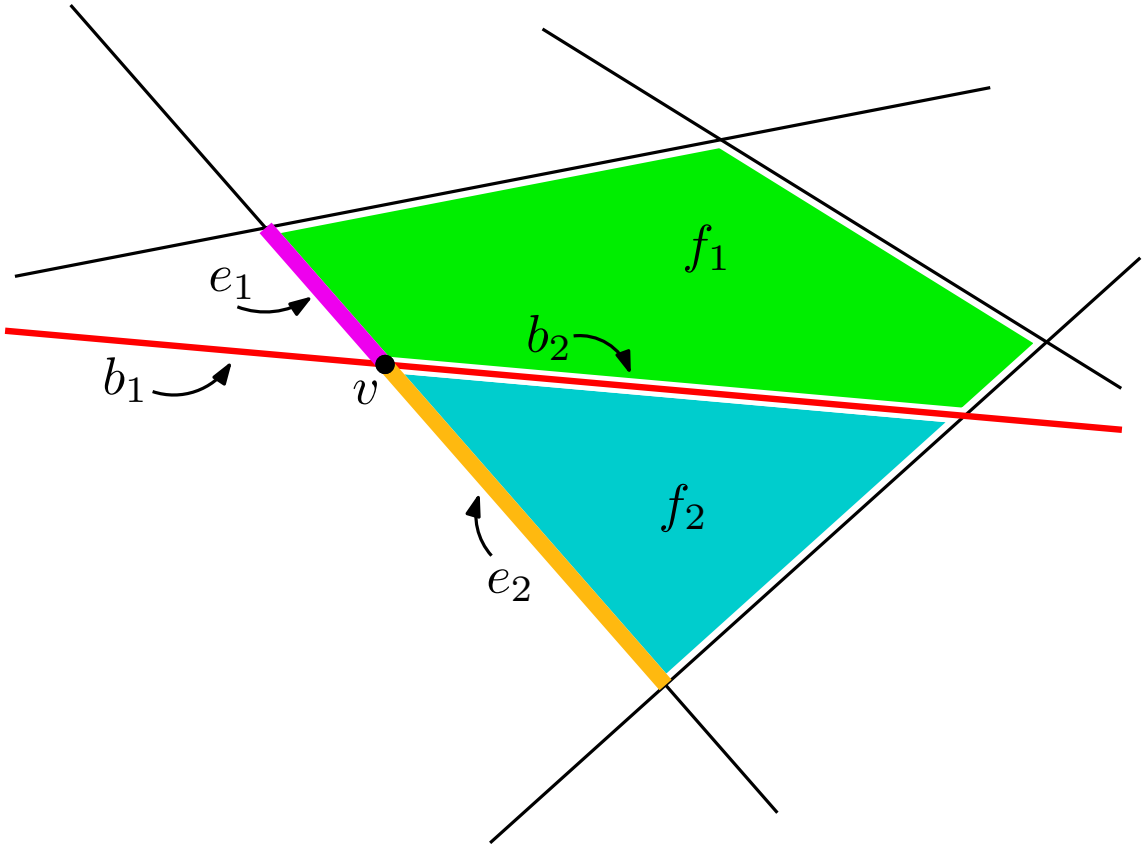
For each new line l:

1. Intersect l with any line l_i pre existent in the arrangement.
2. Traverse the zone of l...
   ...updating the DCEL.

Updating the DCEL:

At each intersection point v, the line l

- partitions an old edge e into two new edges, e_1 and e_2;
- a new edge b_1 ends and a new edge b_2 starts;
- partitions an old face f into two new faces, f_1 and f_2.

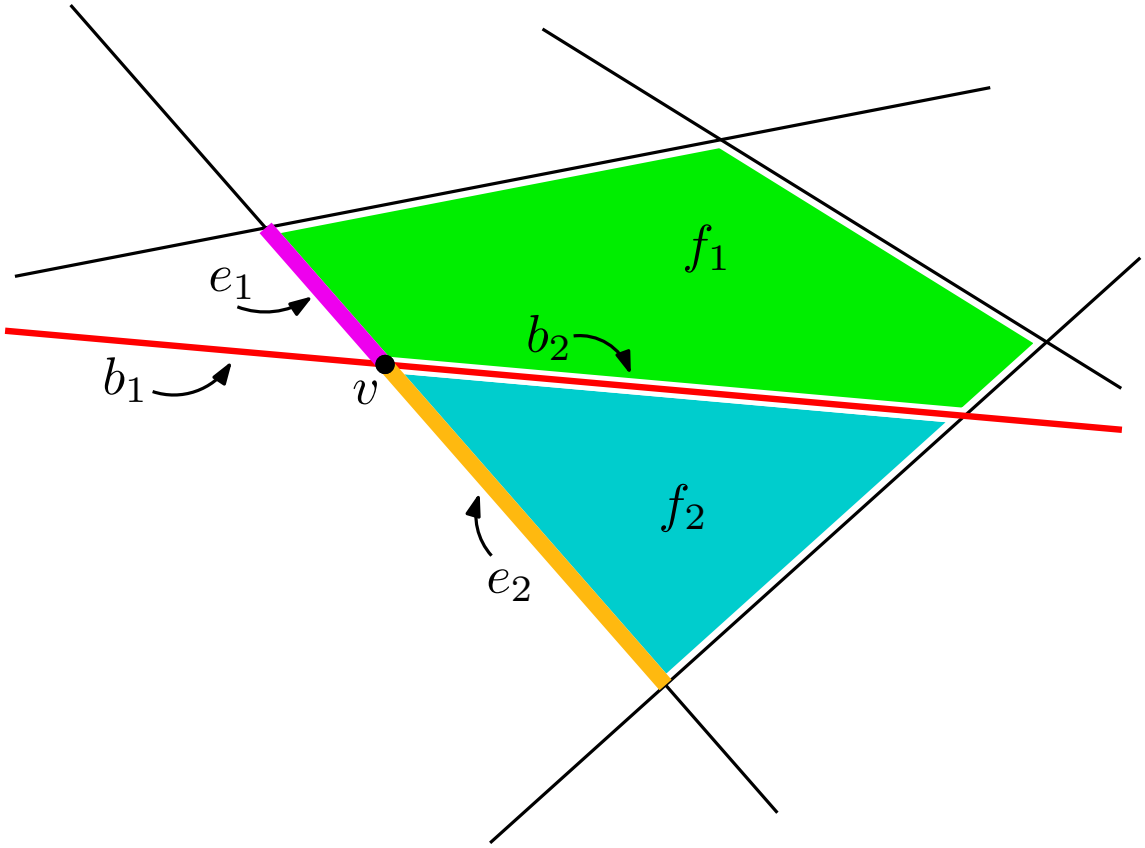


COMPUTATION

Incremental algorithm

At each intersection point v,

- Add v to the table, pointing at b_2.
- Delete f and add f_1 and f_2, pointing at b_2.
- Add v_E and e_N for b_1.
- Add b_2 with all its pointers, excluding v_E and e_N.
- Add e_1 and e_2 with all their pointers, and delete e.



Between two intersection points,

- Modify the pointer of each visited edge to the newly created face.

COMPUTATION

Incremental algorithm

Running time:

For each line (there are n lines):

1. Compute the intersection point in $O(1)$ time and locate, in $O(n)$ time, which edge of the current arrangement contains it.
2. Traverse the zone, which has complexity $O(n)$, updating the DCEL in $O(1)$ time at
   - each intersection point and
   - at each edge of the traversed zone.

Total running time: $O(n^2)$

COMPUTATION

Incremental algorithm

Running time:

For each line (there are n lines):

1. Compute the intersection point in $O(1)$ time and locate, in $O(n)$ time, which edge of the current arrangement contains it.
2. Traverse the zone, which has complexity $O(n)$, updating the DCEL in $O(1)$ time at
   - each intersection point and
   - at each edge of the traversed zone.

Total running time: $O(n^2)$

The description of the algorithm we have shown assumes that the arrangement is simple.

An easy modification allows adapting it to the general case.

EXTENSIONS

The arrangement $\mathcal{A}(L)$ of a finite set of *lines* in the *plane* is the decomposition of the plane into faces, edges and vertices produced by L.

EXTENSIONS

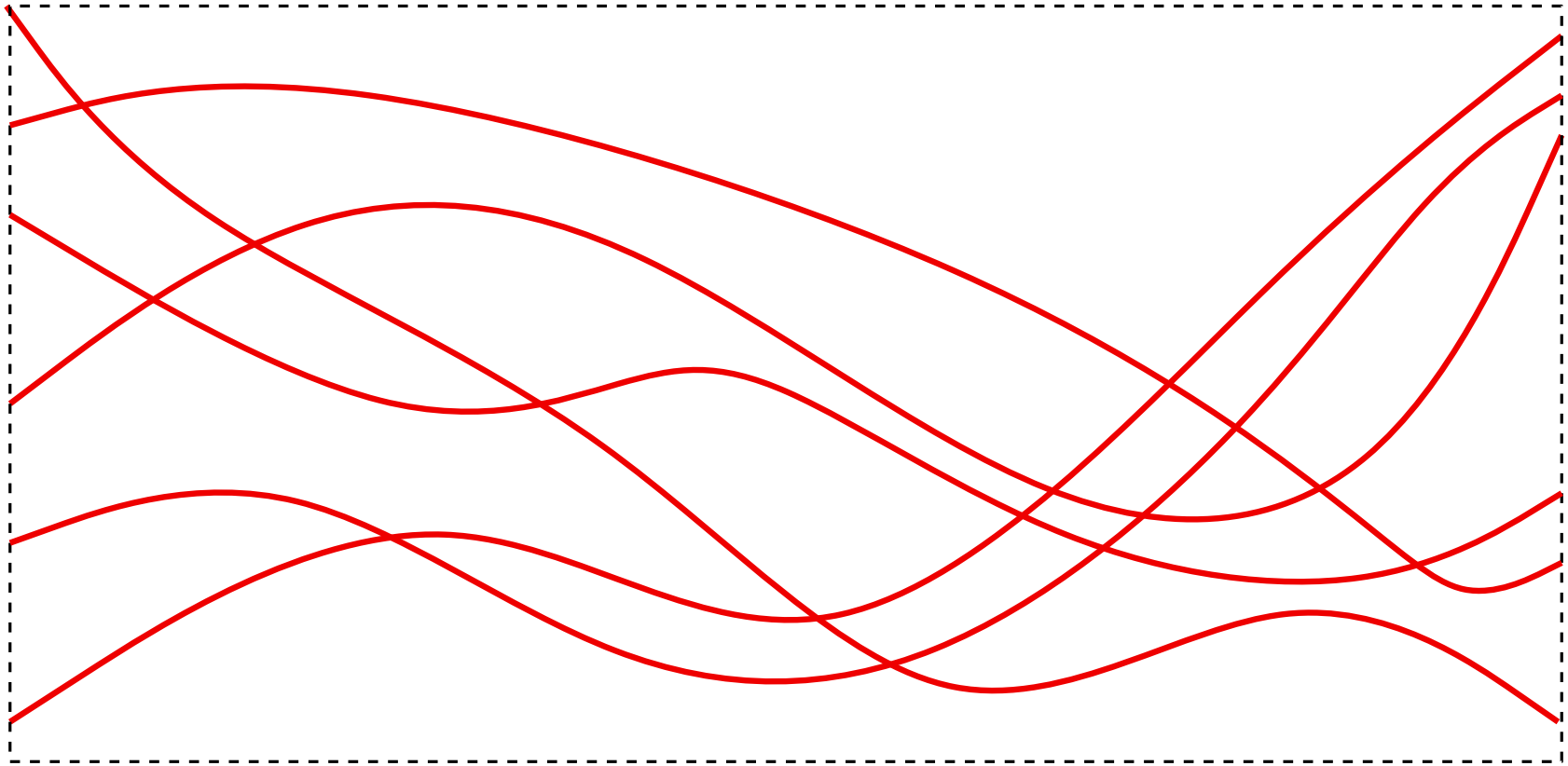
The arrangement $\mathcal{A}(L)$ of a finite set of ~~*lines*~~ in the *plane* is the decomposition of the plane into faces, edges and vertices produced by L.

curves

EXTENSIONS

The arrangement $\mathcal{A}(L)$ of a finite set of ~~*lines*~~ in the *plane* is the decomposition of the plane into faces, edges and vertices produced by L.

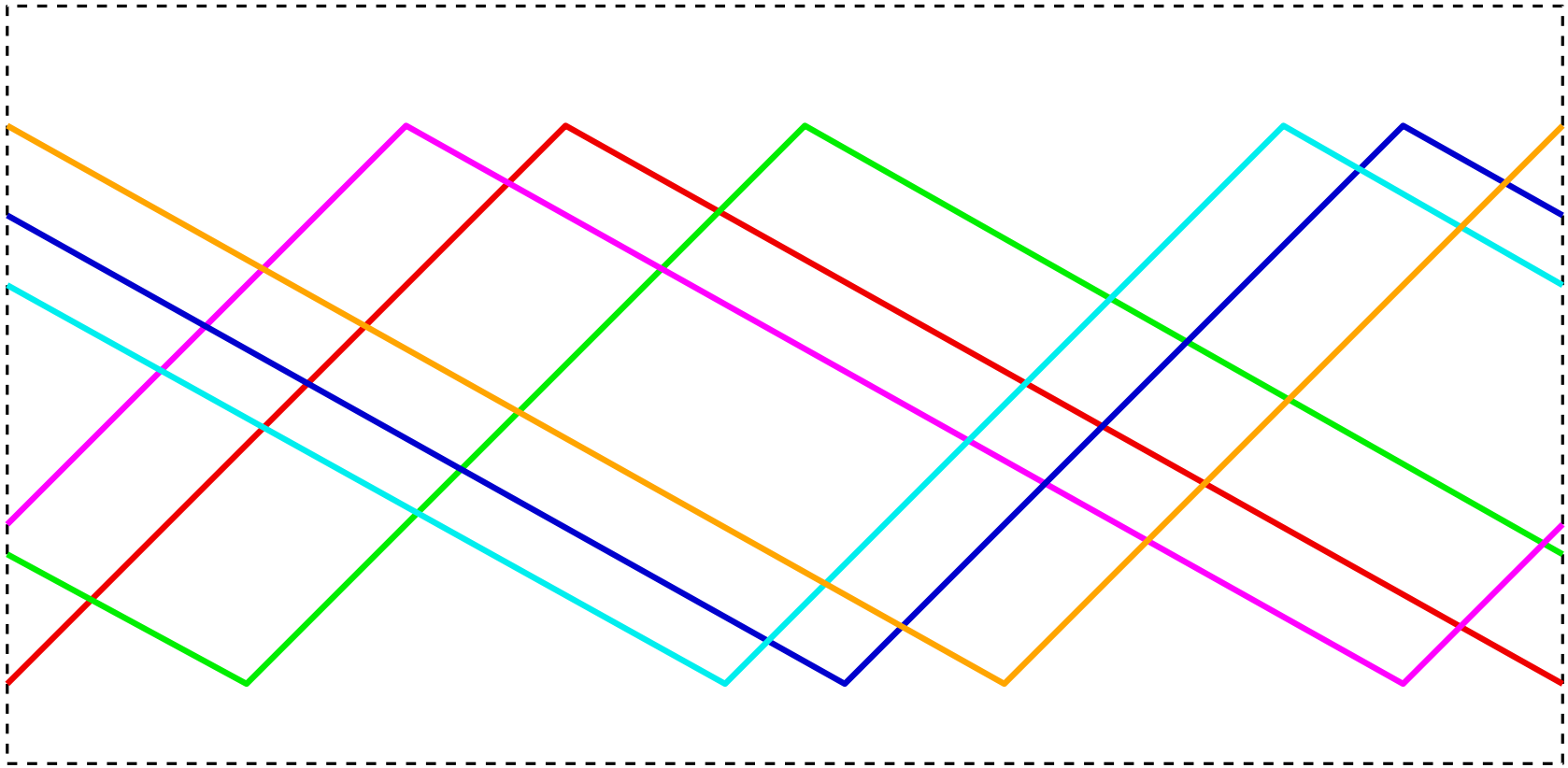
curves



EXTENSIONS

The arrangement $\mathcal{A}(L)$ of a finite set of ~~*lines*~~ in the *plane* is the decomposition of the plane into faces, edges and vertices produced by L.

curves

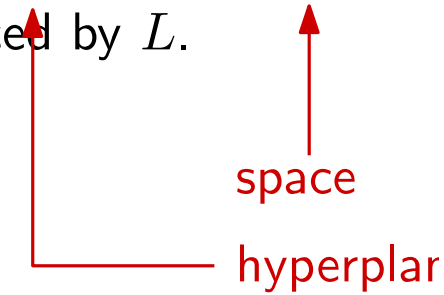


EXTENSIONS

The arrangement $\mathcal{A}(L)$ of a finite set of *lines* in the ~~*plane*~~ is the decomposition of the plane into faces, edges and vertices produced by L.

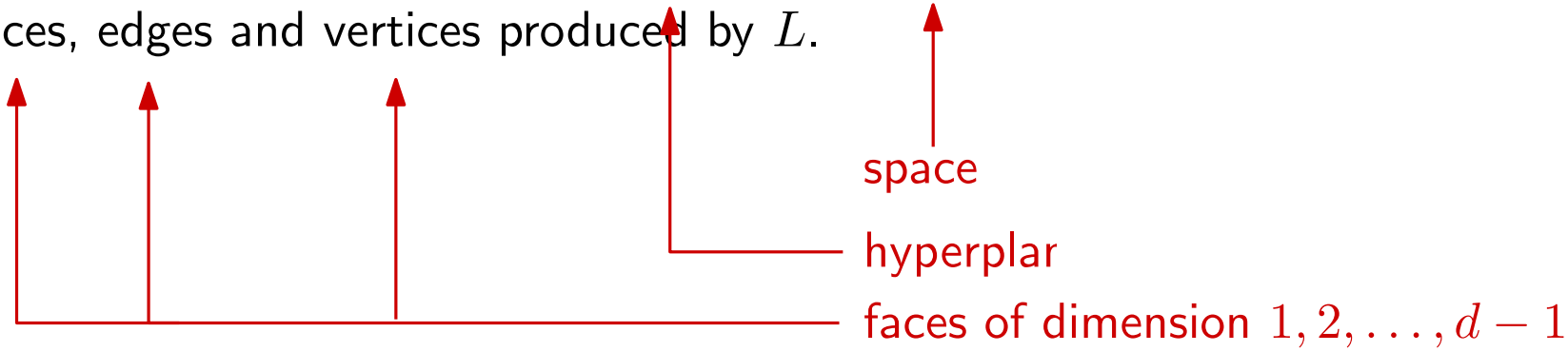
EXTENSIONS

The arrangement $\mathcal{A}(L)$ of a finite set of *lines* in the ~~*plane*~~ is the decomposition of the plane into faces, edges and vertices produced by L.



EXTENSIONS

The arrangement $\mathcal{A}(L)$ of a finite set of *lines* in the ~~*plane*~~ is the decomposition of the plane into faces, edges and vertices produced by L.



ARRANGEMENTS

APPLICATIONS

ARRANGEMENTS

APPLICATIONS

- Arrangements and Voronoi diagrams
- Arrangements of lines as duals of point sets
  (with much more structure)
  - k-sets
  - ham-sandwich cuts
- Elimination of hidden surfaces
  (for n disjoint polygons in space, solved in $O(n^2)$ time by topological sweep)
- Aspect graphs
- Motion planning (visibility graphs)
- And many more! ... even for checking whether there exist 3 or more colinear points in a set of n points.

ARRANGEMENTS

MORE ABOUT IT

- J. O'Rourke: *Computational Geometry in C*, Cambridge University Press.
- M. de Berg, O. Cheong, M. van Kreveld, M. Overmars: *Computational Geometry: Alhorithms and Applications*, Springer.
- H. Edelsbrunner: *Algorithms in Combinatorial Geometry*, Springer.
- L. J. Guibas, M. Sharir: Combinatorics and Algorithms of Arrangements, in *New Trends in Discrete and Computational Geometry. Algorithms and Combinatorics*, J. Pach ed., Springer.
- M. Sharir, P. K. Agarwal: *Davenport-Shinzel Sequences and Their Geometric Applications*, Cambridge University Press.