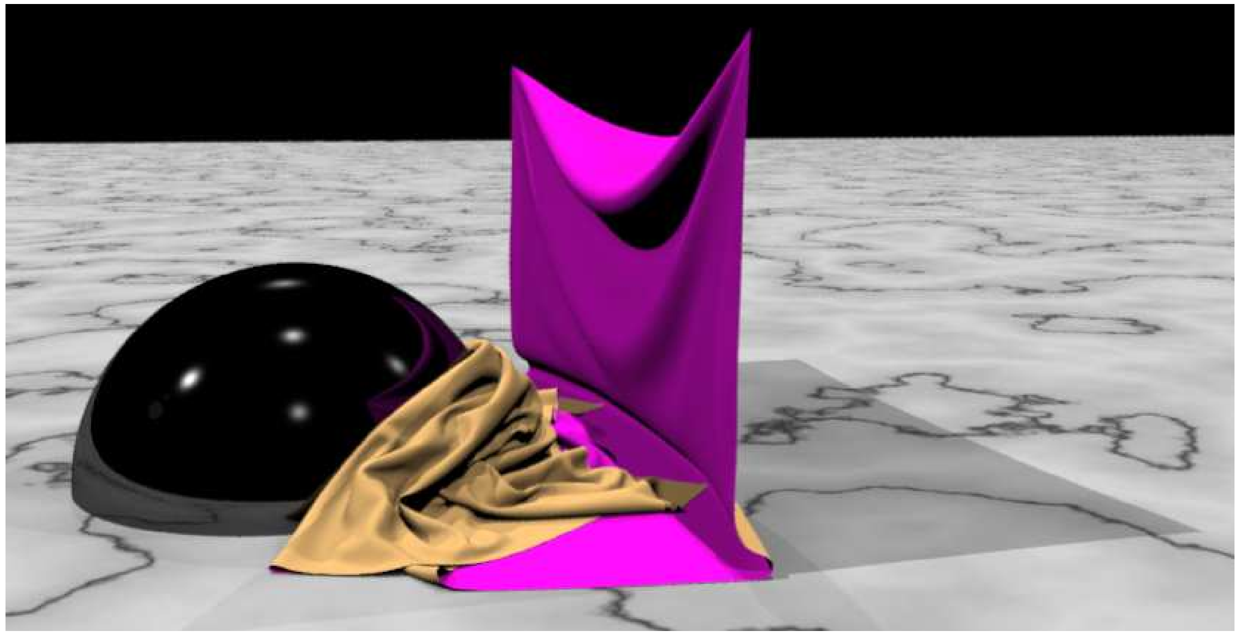
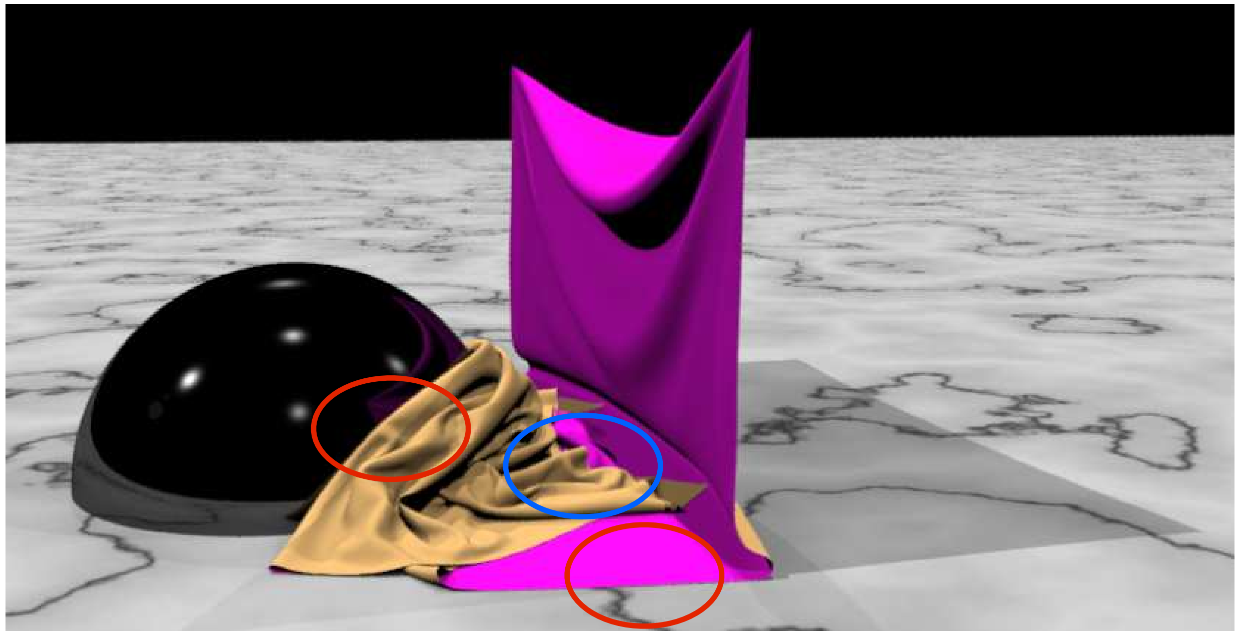
Collision processing

- Different types of collisions
  - Collision of a simulated deformable structure with a kinematic structure (easier)
    - Collision with a rigid moving object, the ground, etc.
    - Collision object can even be *deforming* as long as its deformation is kinematic (i.e. scripted), not simulated
  - Self collision within a deformable structure (harder)
    - Also includes collision between 2+ deformable structures



Collision processing

- Different types of collisions
  - Collision of a simulated deformable structure with a kinematic structure (easier)
    - Collision with a rigid moving object, the ground, etc.
    - Collision object can even be *deforming* as long as its deformation is kinematic (i.e. scripted), not simulated
  - Self collision within a deformable structure (harder)
    - Also includes collision between 2+ deformable structures



Collision processing

- Typically partitioned into two tasks/phases
  - *Collision detection*
    - Detect if an interpenetration event occurred
    - Localize such events, in space and time
    - *(If required)* determine depth and direction of collision
  - *Collision response (or resolution)*
    - Attempt to resolve and fix all collisions, and/or
    - Try to make collision less severe (but tolerate some), and/or
    - Take steps to *prevent* collisions in the imminent future

Collision processing

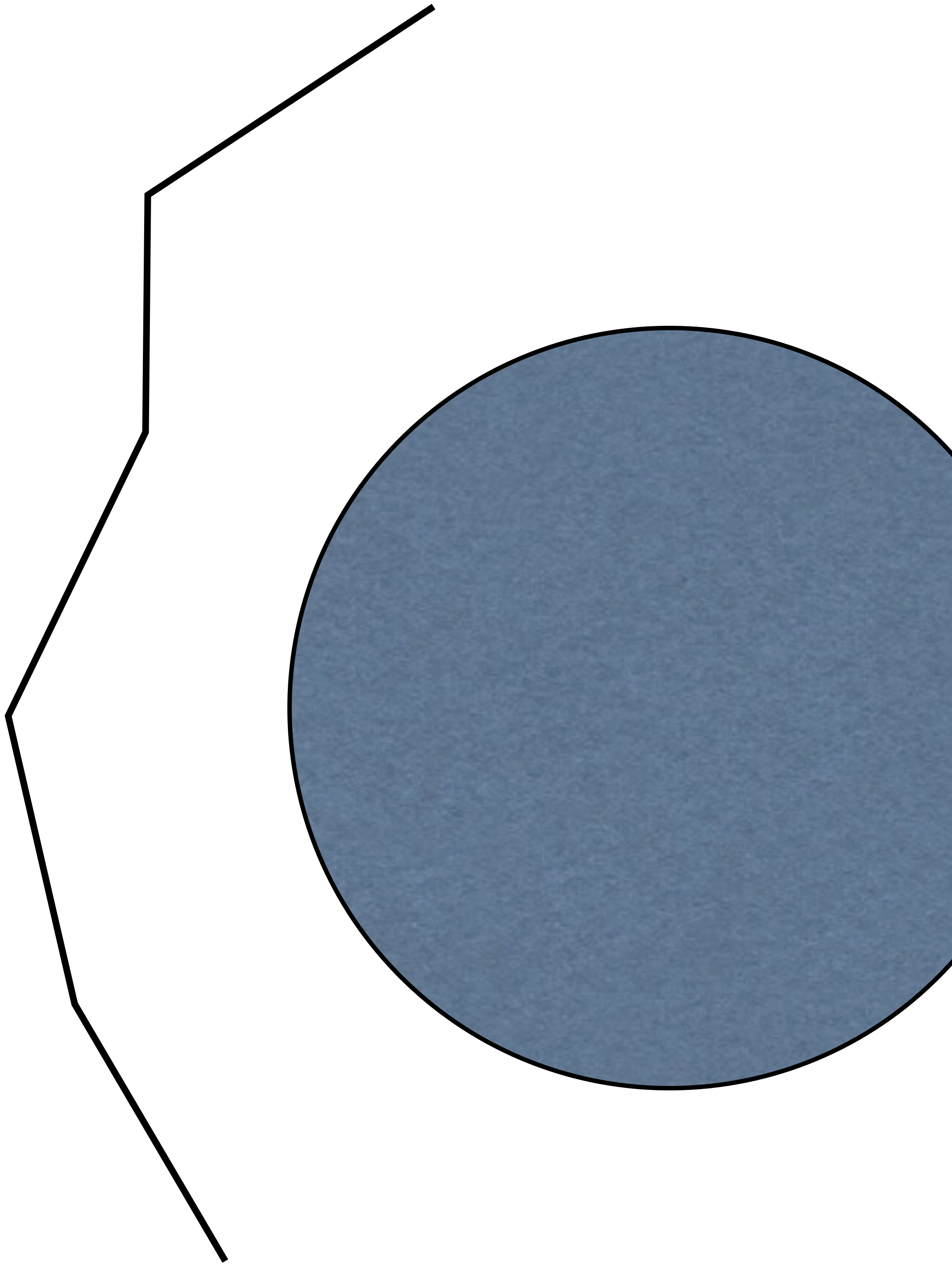
- Typically partitioned into two tasks/phases
  - *Collision detection*
    - Detect if an interpenetration event occurred
    - Localize such events, in space and time
    - *(If required)* determine depth and direction of collision

The exact nature of collision detection depends on how we expect to use that information in the response stage!

  - *Collision response (or resolution)*
    - Attempt to resolve and fix all collisions, and/or
    - Try to make collision less severe (but tolerate some), and/or
    - Take steps to *prevent* collisions in the imminent future

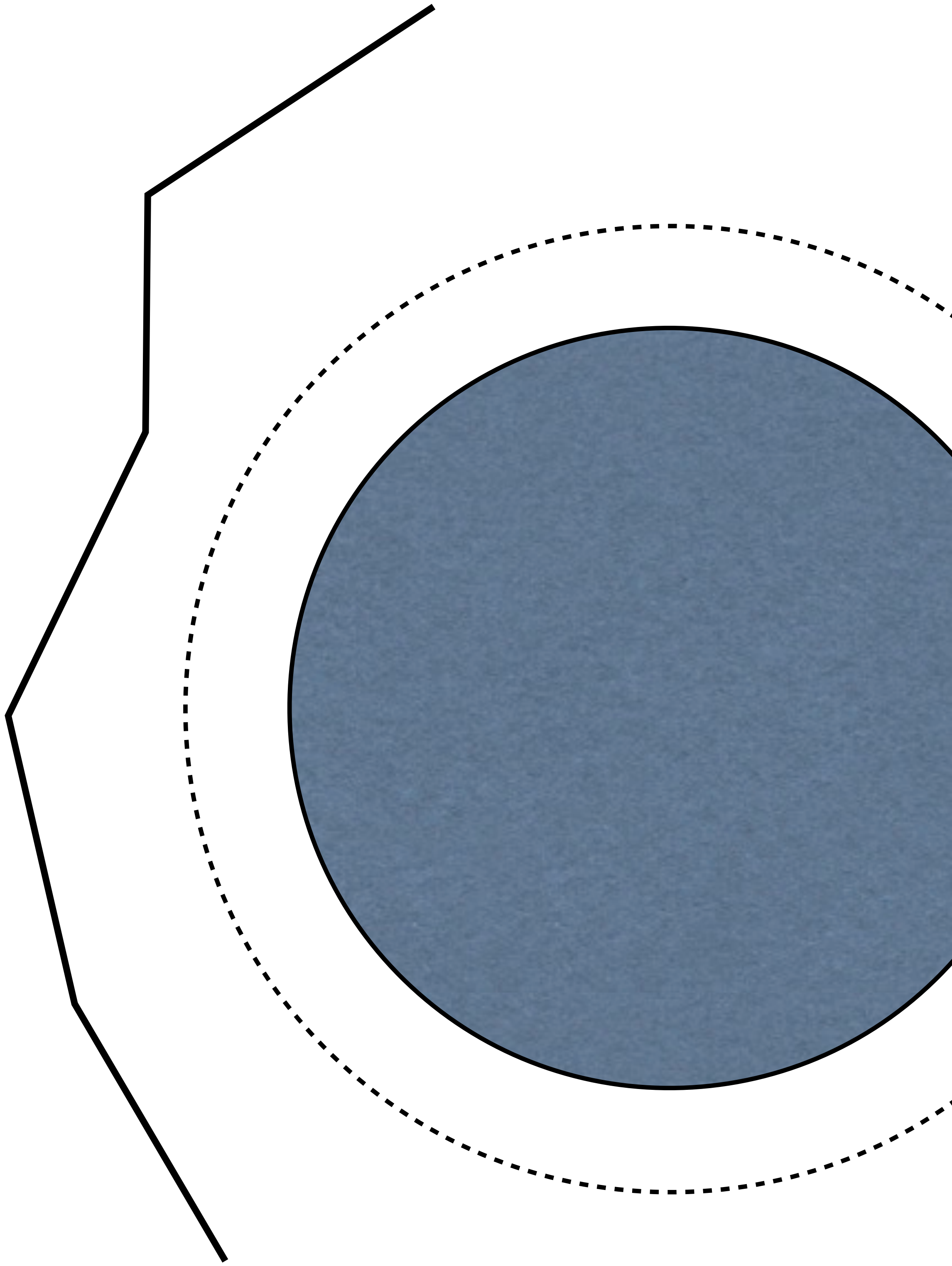
Collision response (general approaches)

- Penalty-based methods
  - Detect proximity to collision objects and apply a repulsive "*penalty*" force when the distance to the collision target is small
  - Increase strength of repulsion force as distance decreases (or as interpenetration starts to occur)



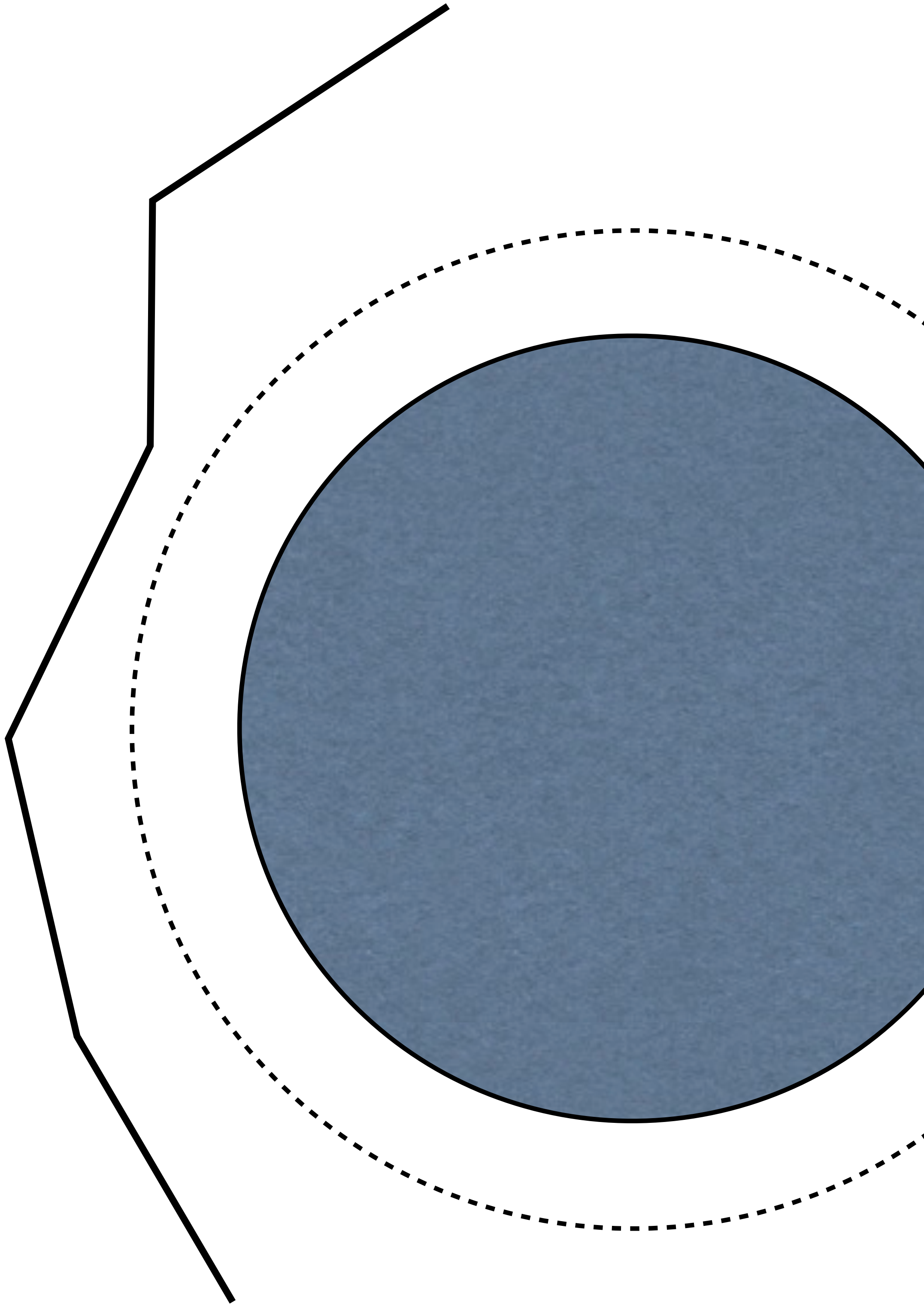
Collision response (general approaches)

- Penalty-based methods
  - Detect proximity to collision objects and apply a repulsive "*penalty*" force when the distance to the collision target is small
  - Increase strength of repulsion force as distance decreases (or as interpenetration starts to occur)



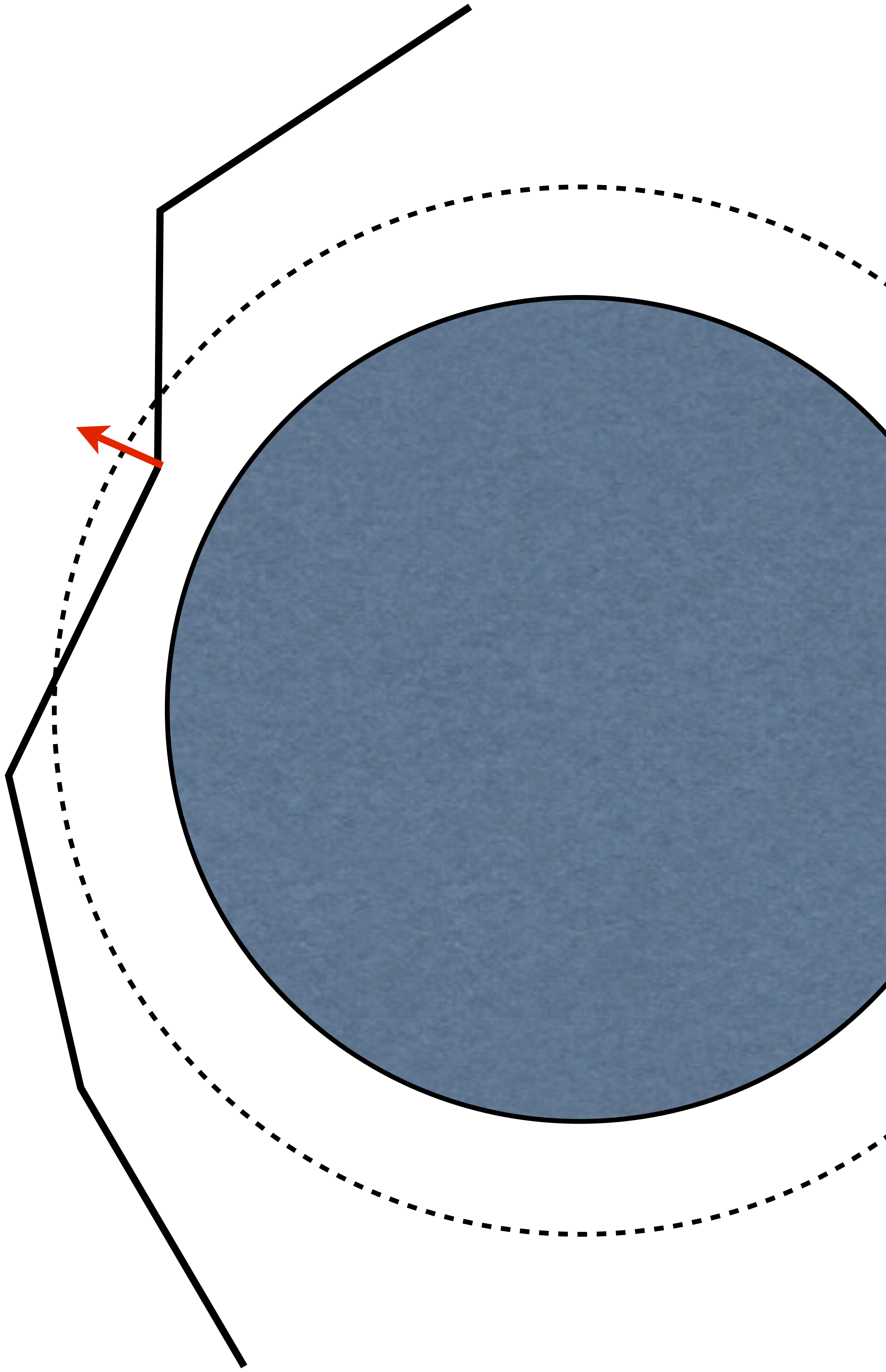
Collision response (general approaches)

- Penalty-based methods
  - Detect proximity to collision objects and apply a repulsive "*penalty*" force when the distance to the collision target is small
  - Increase strength of repulsion force as distance decreases (or as interpenetration starts to occur)



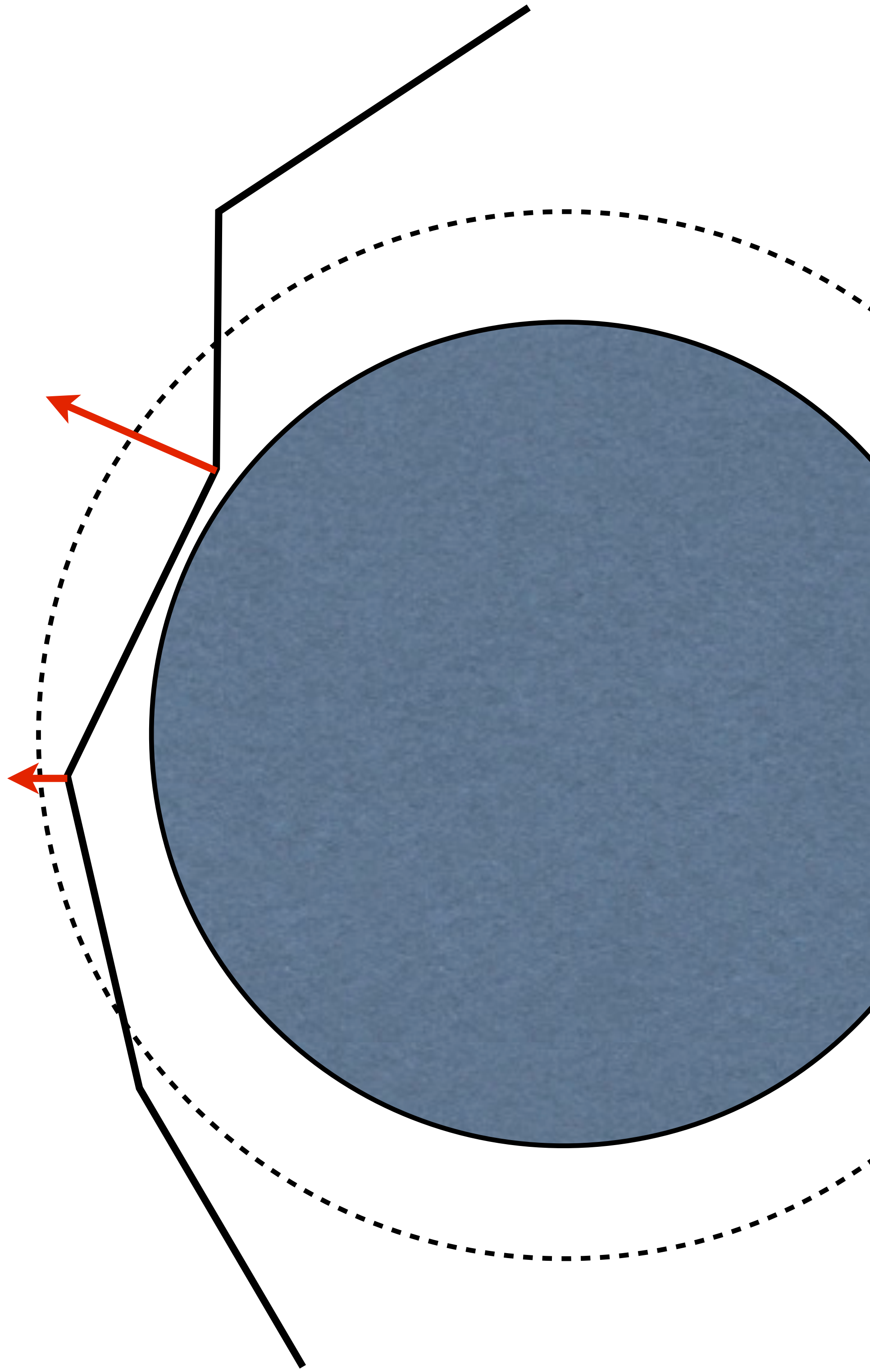
Collision response (general approaches)

- Penalty-based methods
  - Detect proximity to collision objects and apply a repulsive "*penalty*" force when the distance to the collision target is small
  - Increase strength of repulsion force as distance decreases (or as interpenetration starts to occur)



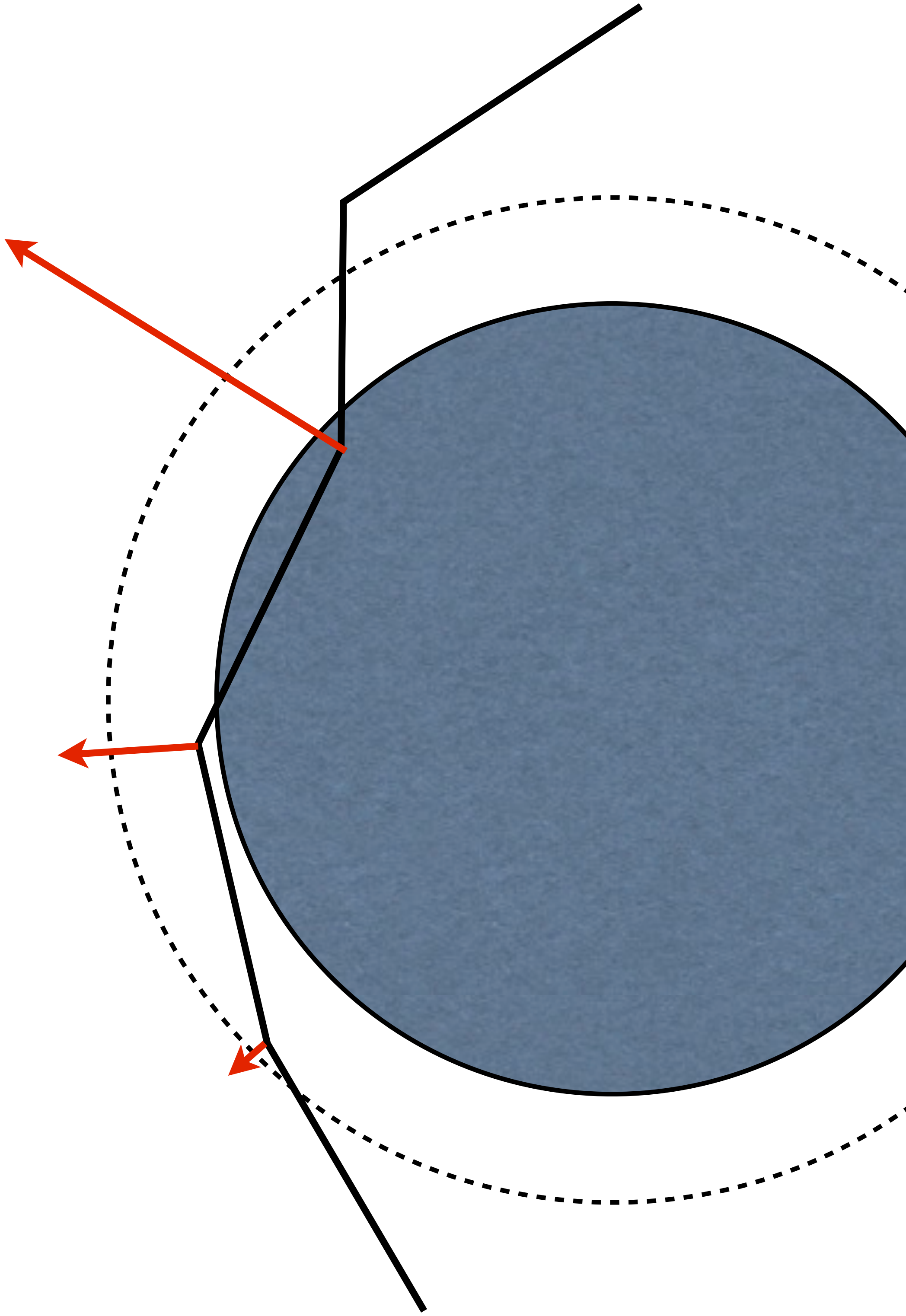
Collision response (general approaches)

- Penalty-based methods
  - Detect proximity to collision objects and apply a repulsive "*penalty*" force when the distance to the collision target is small
  - Increase strength of repulsion force as distance decreases (or as interpenetration starts to occur)



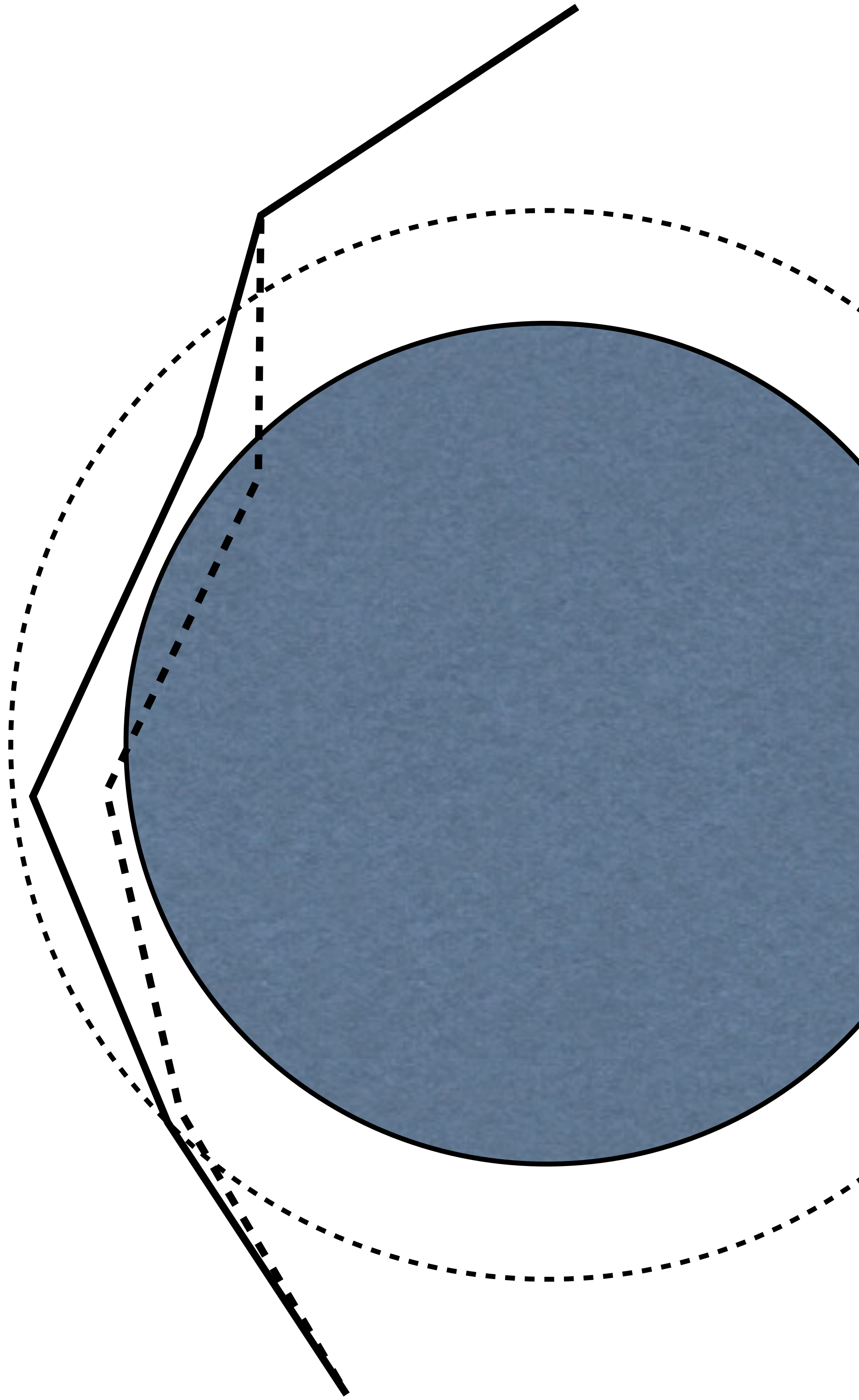
Collision response (general approaches)

- Penalty-based methods
  - Detect proximity to collision objects and apply a repulsive "*penalty*" force when the distance to the collision target is small
  - Increase strength of repulsion force as distance decreases (or as interpenetration starts to occur)



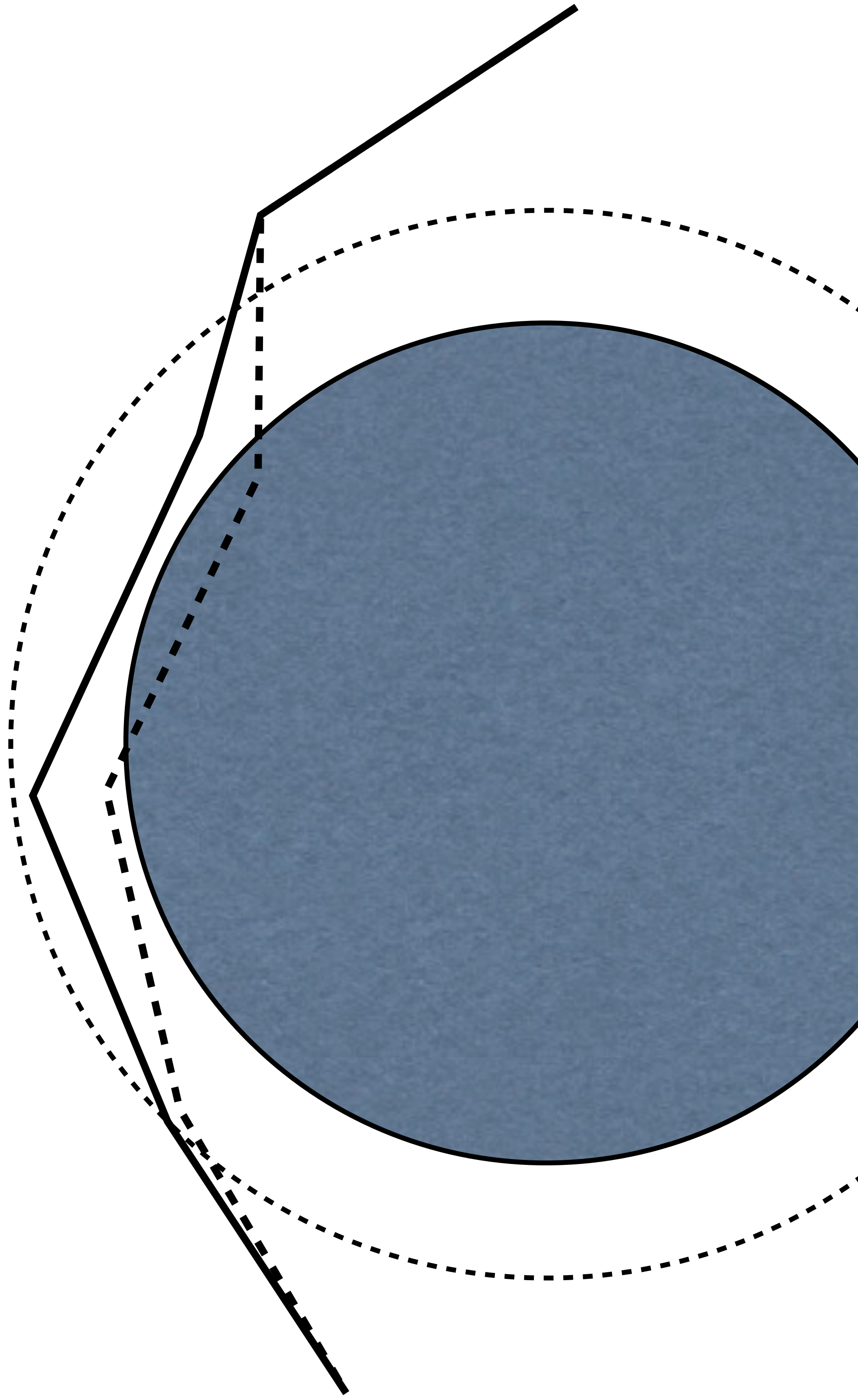
Collision response (general approaches)

- Penalty-based methods
    - Detect proximity to collision objects and apply a repulsive "*penalty*" force when the distance to the collision target is small
    - Increase strength of repulsion force as distance decreases (or as interpenetration starts to occur)



Collision response (general approaches)

- Penalty-based methods
  - Detect proximity to collision objects and apply a repulsive "*penalty*" force when the distance to the collision target is small
  - Increase strength of repulsion force as distance decreases (or as interpenetration starts to occur)
  - Does not strictly *enforce* a collision-free state, but attempts to prevent it, and lessen the degree of collision

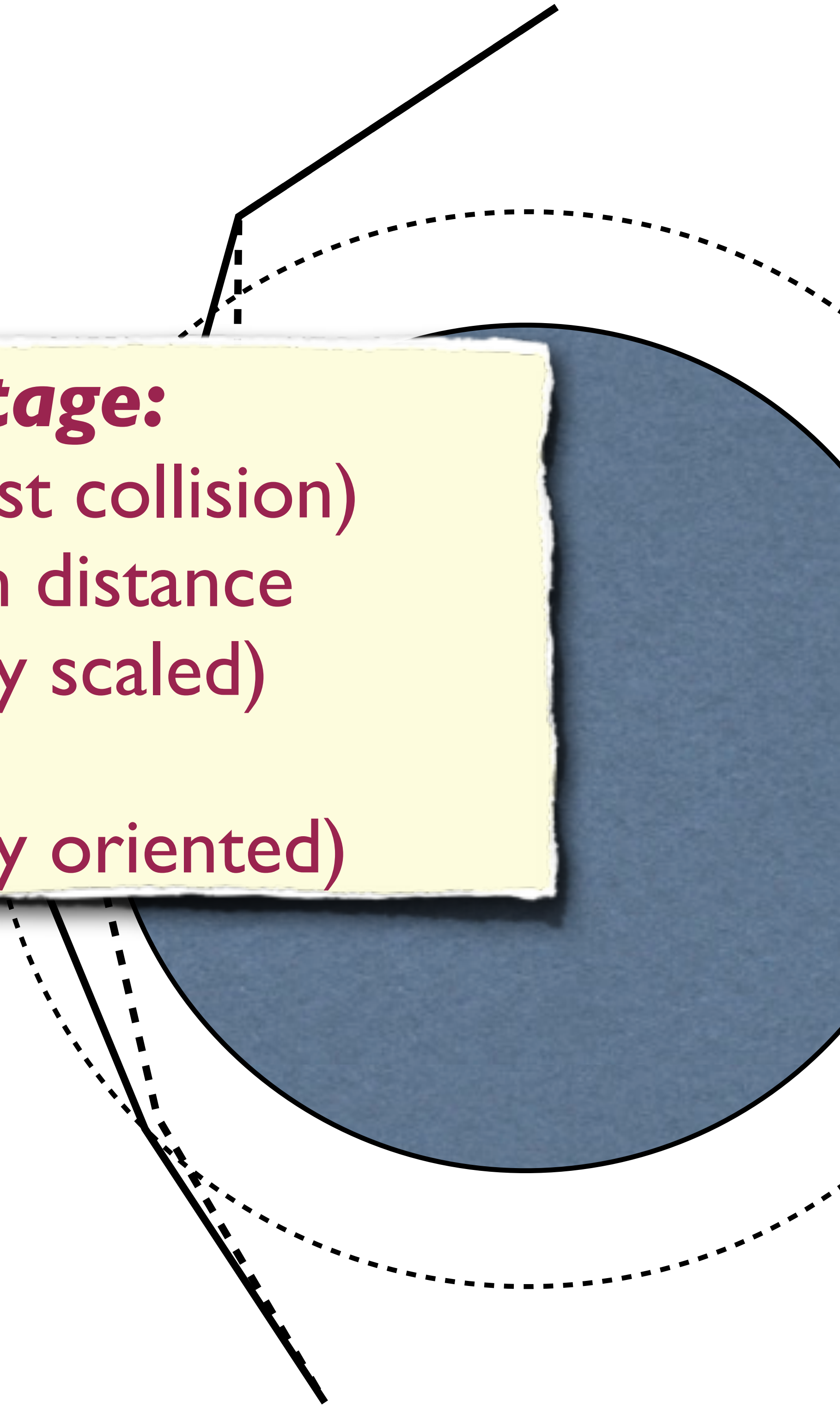


Collision response (general approaches)

- Penalty-based methods
  - Detect proximity to collision objects and apply a repulsive "*penalty*" force when the di ta
  - In fo (c to occur)
  - Does not strictly *enforce* a collision-free state, but attempts to prevent it, and lessen the degree of collision

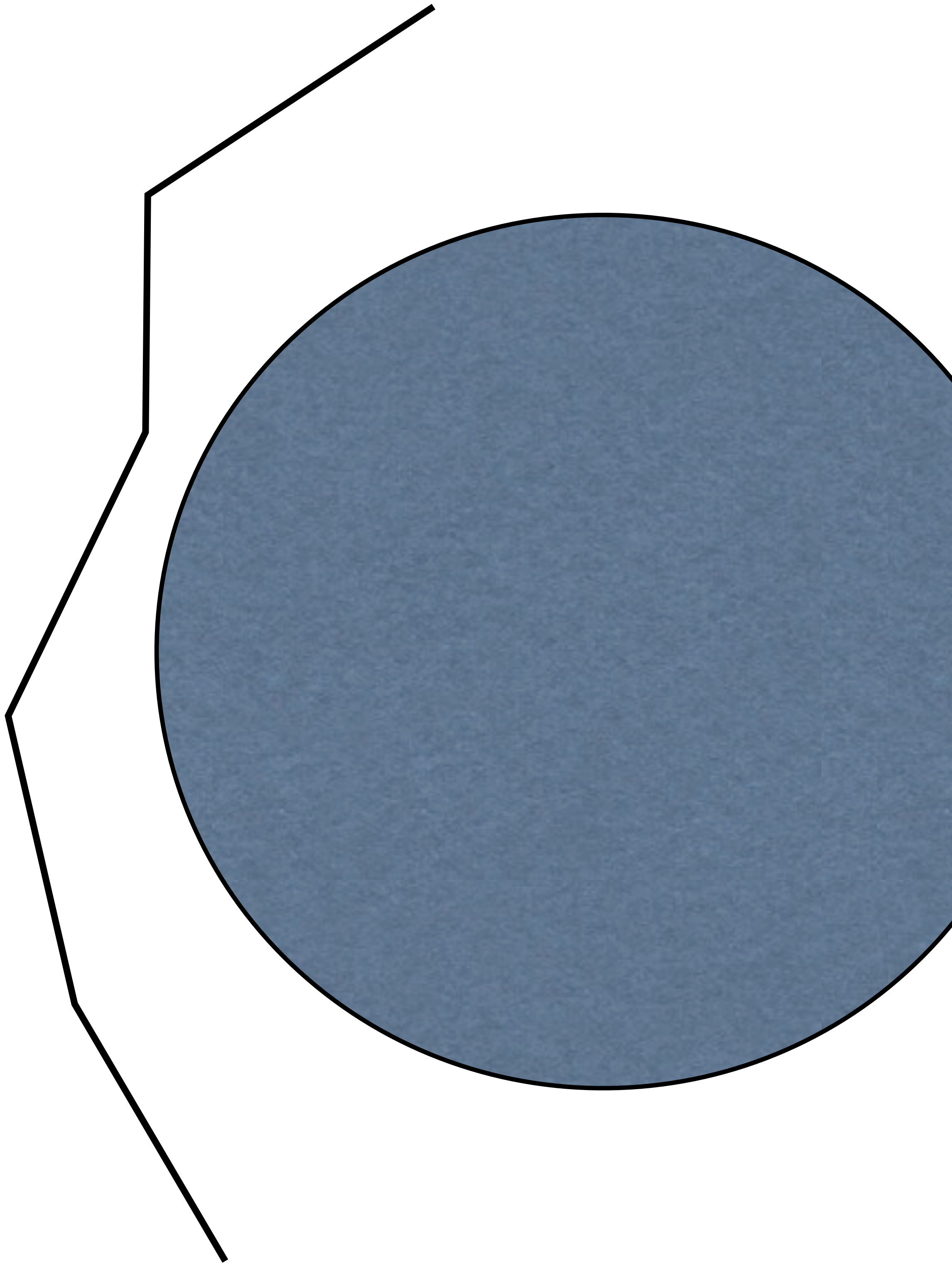
Requirements for the detection stage:

- Detection of *static proximity* (not just collision)
- Estimation of proximity or collision distance (such that forces can be accordingly scaled)
- Estimation of collision *direction* (such that forces can be accordingly oriented)



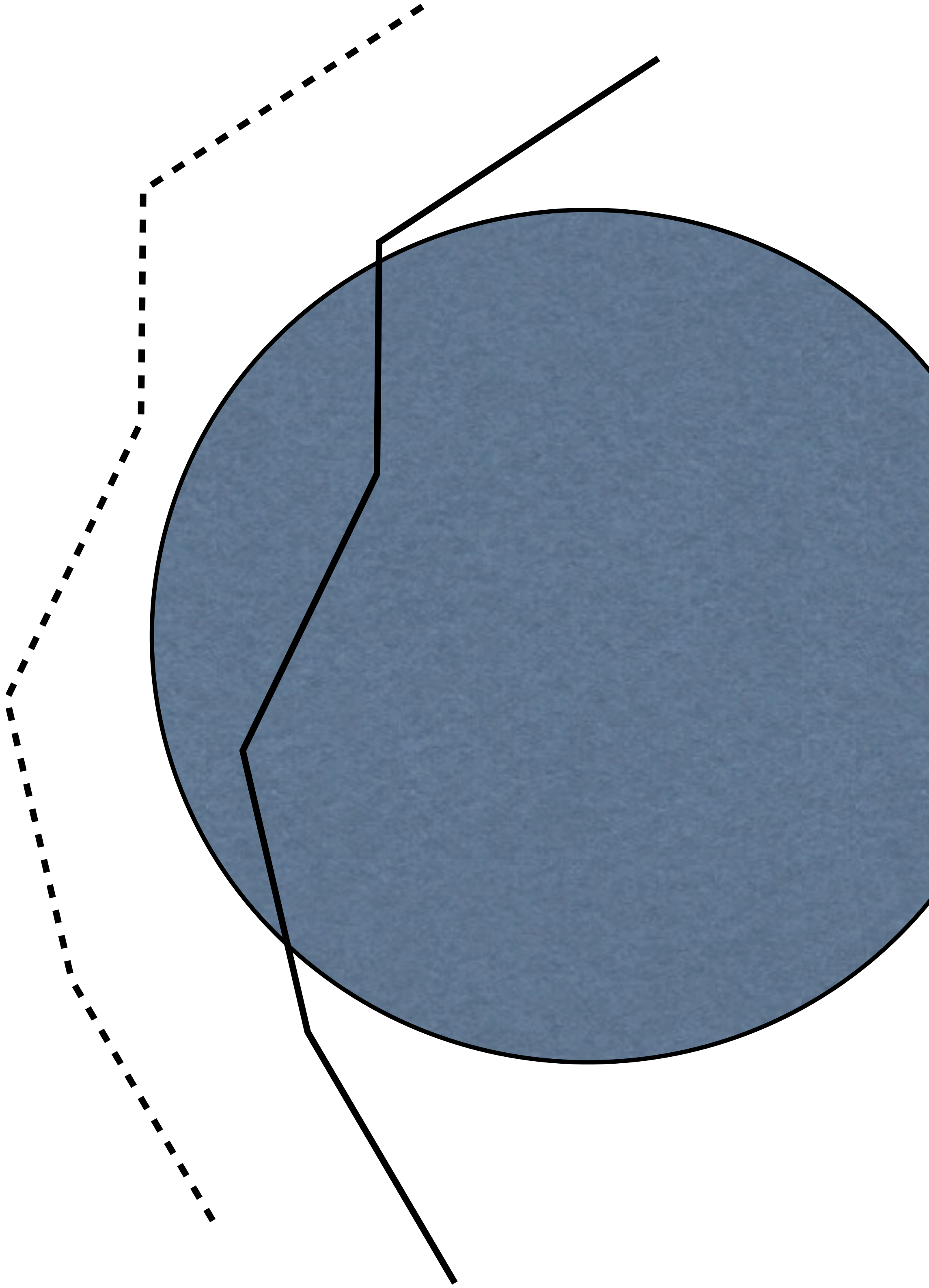
Collision response (general approaches)

- Impulse-based methods
  - Usually attempt to guarantee that *no collision* is produced or left untreated, at any time
  - Starting from a collision-free state at time t*, the system is advanced to time t*+dt
  - Collisions that occurred in the interval [t*,t*+dt] are localized (in space and time)
  - An *impulse* is applied to instantaneously correct the object trajectory and prevent (or fix) any collision events



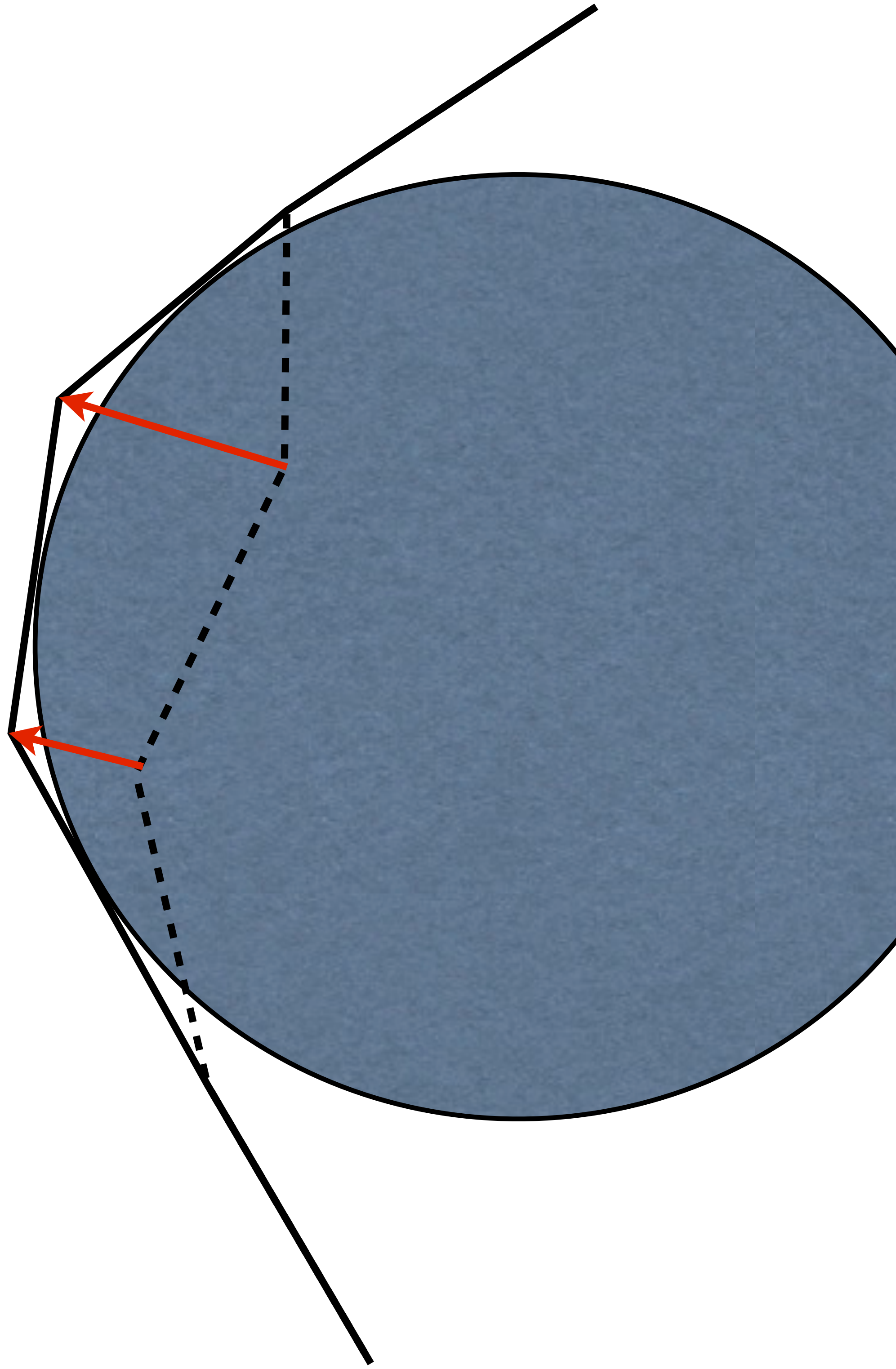
Collision response (general approaches)

- Impulse-based methods
  - Usually attempt to guarantee that *no collision* is produced or left untreated, at any time
  - Starting from a collision-free state at time t*, the system is advanced to time t*+dt
  - Collisions that occurred in the interval [t*,t*+dt] are localized (in space and time)
  - An *impulse* is applied to instantaneously correct the object trajectory and prevent (or fix) any collision events



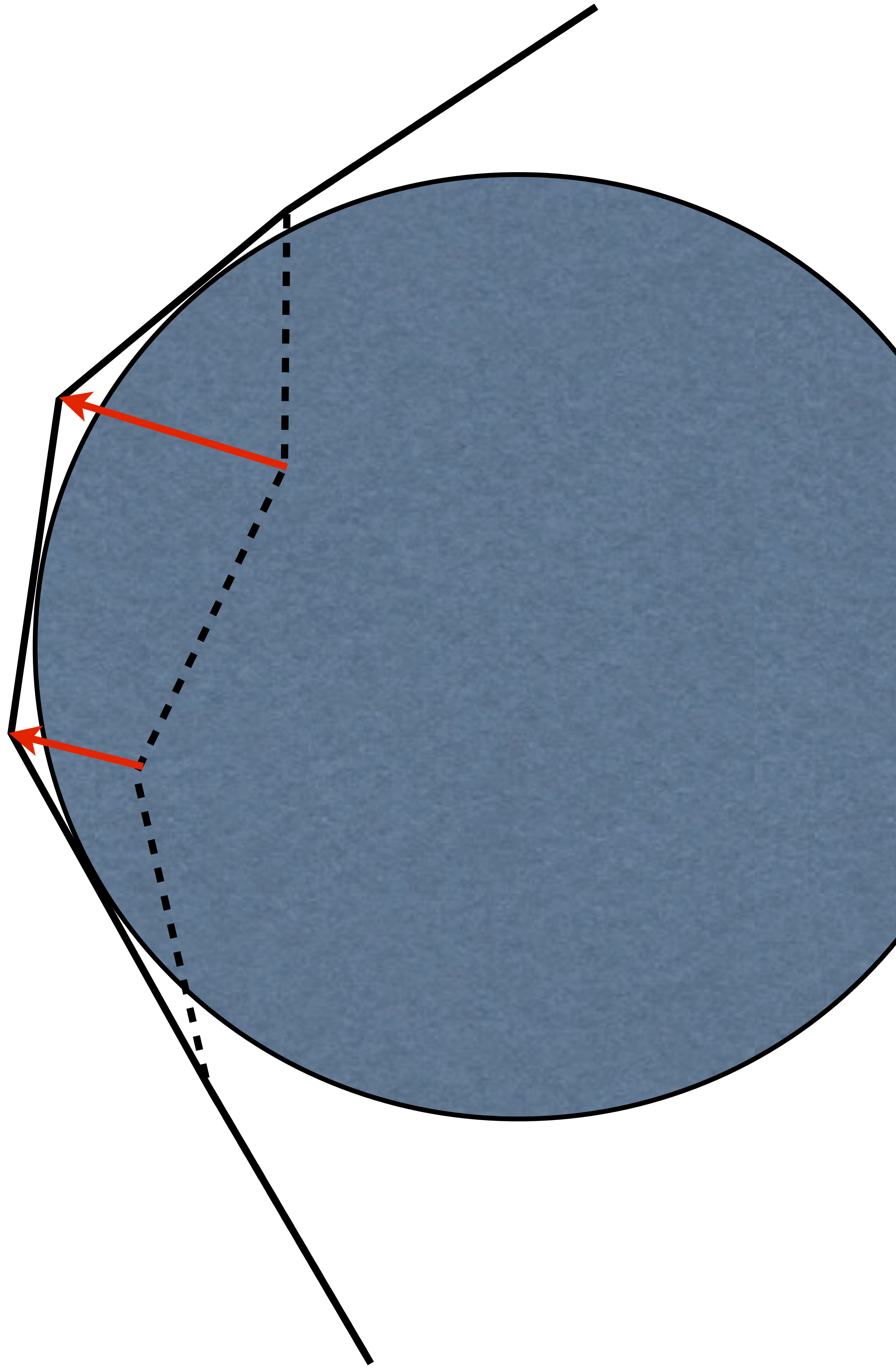
Collision response (general approaches)

- Impulse-based methods
  - Usually attempt to guarantee that *no collision* is produced or left untreated, at any time
  - Starting from a collision-free state at time t*, the system is advanced to time t*+dt
  - Collisions that occurred in the interval [t*,t*+dt] are localized (in space and time)
  - An *impulse* is applied to instantaneously correct the object trajectory and prevent (or fix) any collision events



Collision response (general approaches)

- Impulse-based methods
  - Can be structured to provide guarantees of non-interpenetration (makes other parts of the simulation simpler and easier)
  - Capable of enforcing *tight* contact, instead of modeling a large, artificial "*thickness*" for the collision object
  - Not guaranteed to succeed, especially with conflicting nonphysical constraints
  - Relatively slow and expensive



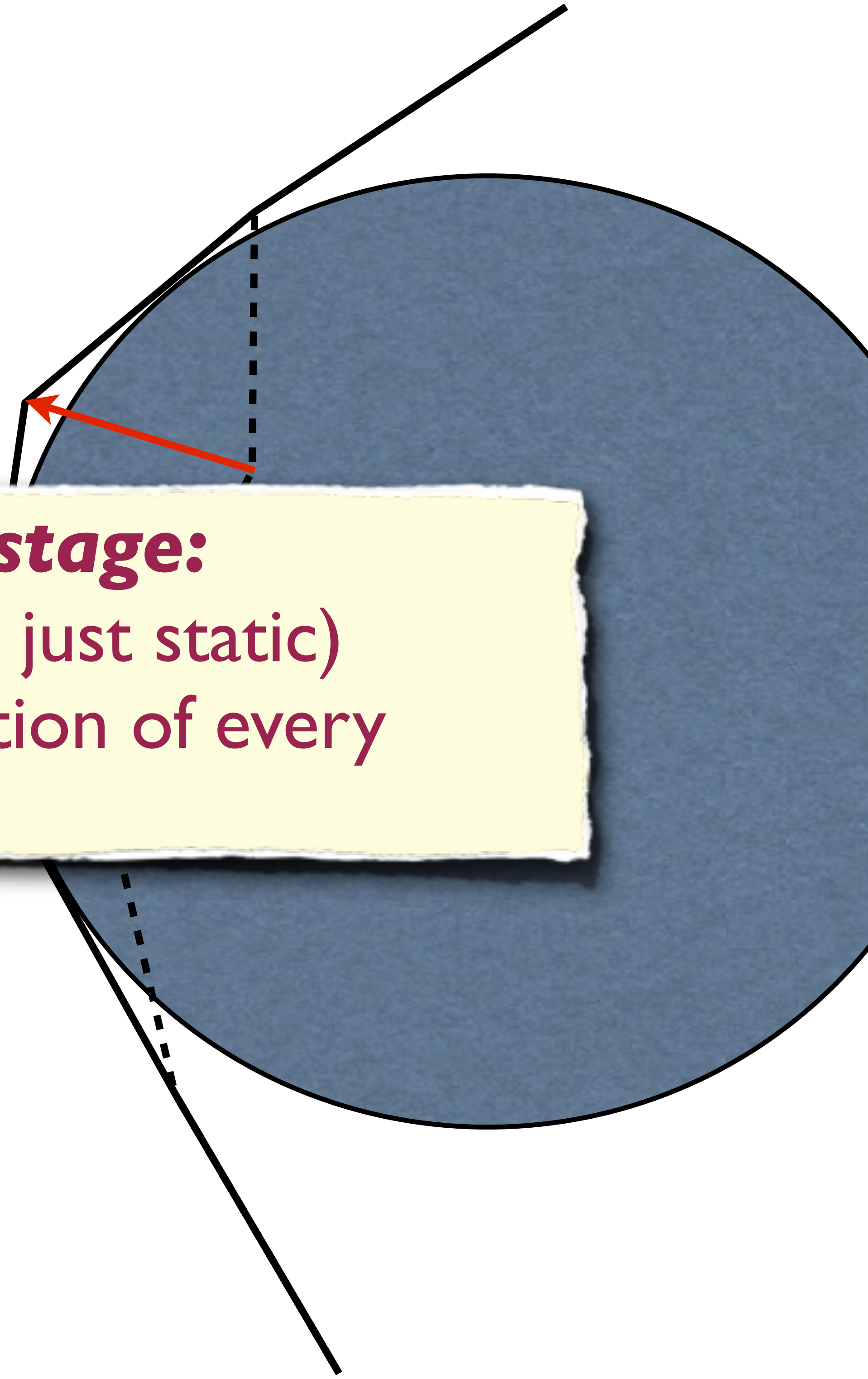
Collision response (general approaches)

- Impulse-based methods
  - Can be structured to provide guarantees of non-interpenetration (makes other parts of the simulation simpler and easier)
  - C co la th

Requirements for the detection stage:

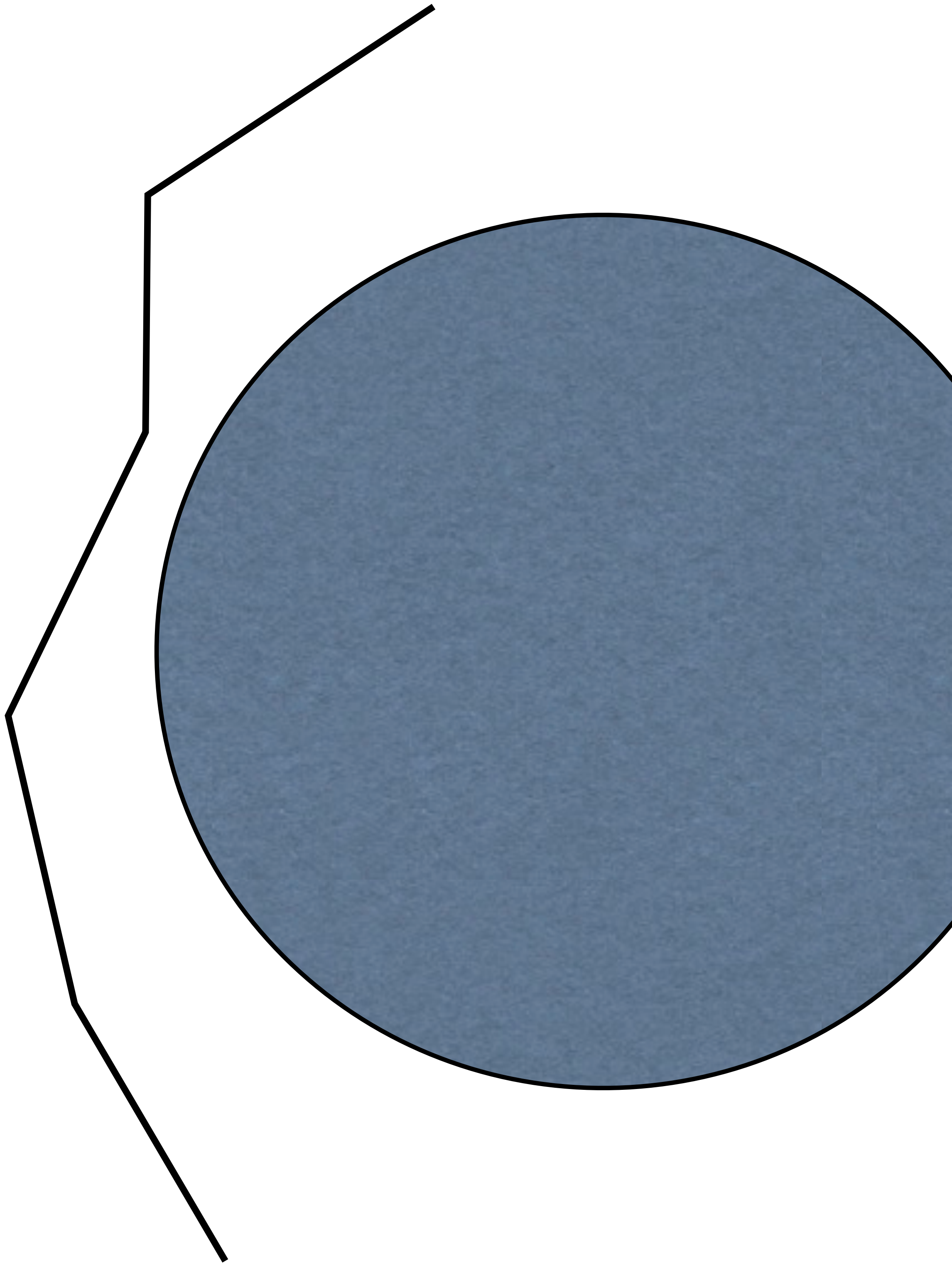
- Detection of *moving collisions* (not just static)
- Estimation of the exact time/location of every impact event

  - Not guaranteed to succeed, especially with conflicting nonphysical constraints
  - Relatively slow and expensive



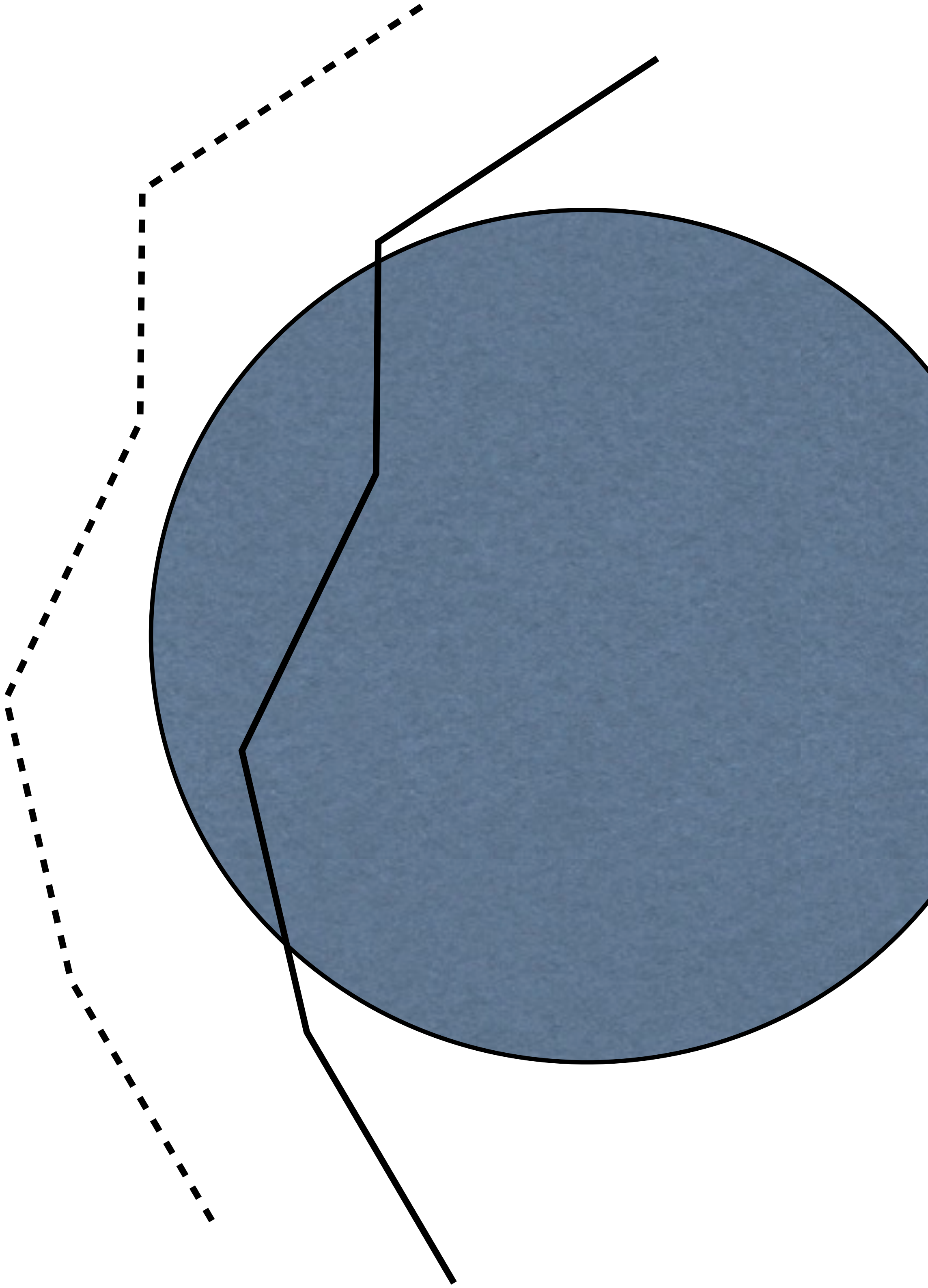
Collision response (general approaches)

- Continuous time collisions
  - Most "*physically justified*" technique : handle one collision event at a time, in the order they occur
  - Can be structured to pursue full avoidance of collisions, while not requiring collision objects to be thickened
  - Response can be formulated in terms of simple and intuitive penalty forces
  - Disadvantage : May lead to very small time steps



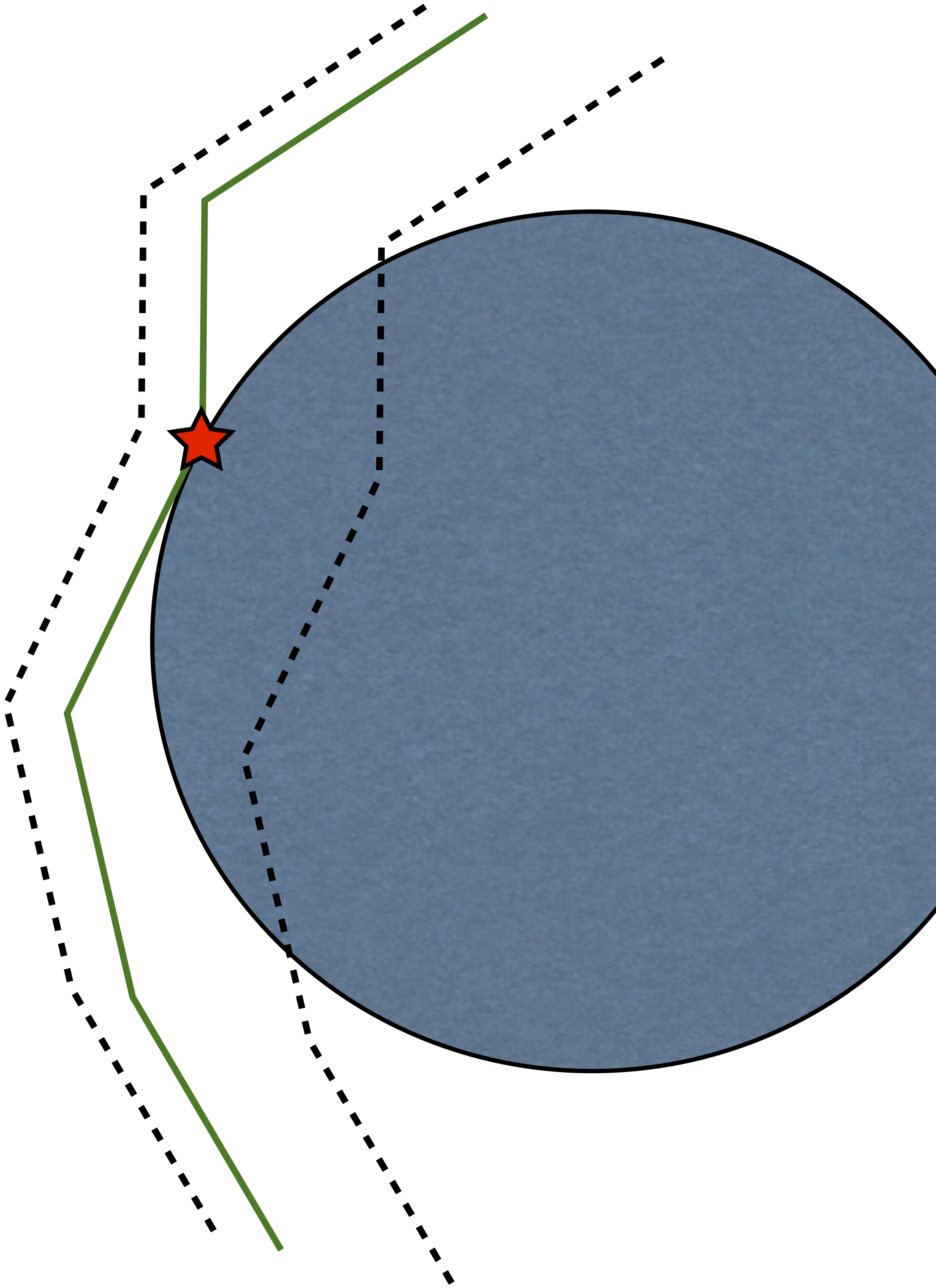
Collision response (general approaches)

- Continuous time collisions
  - Most "*physically justified*" technique : handle one collision event at a time, in the order they occur
  - Can be structured to pursue full avoidance of collisions, while not requiring collision objects to be thickened
  - Response can be formulated in terms of simple and intuitive penalty forces
  - Disadvantage : May lead to very small time steps



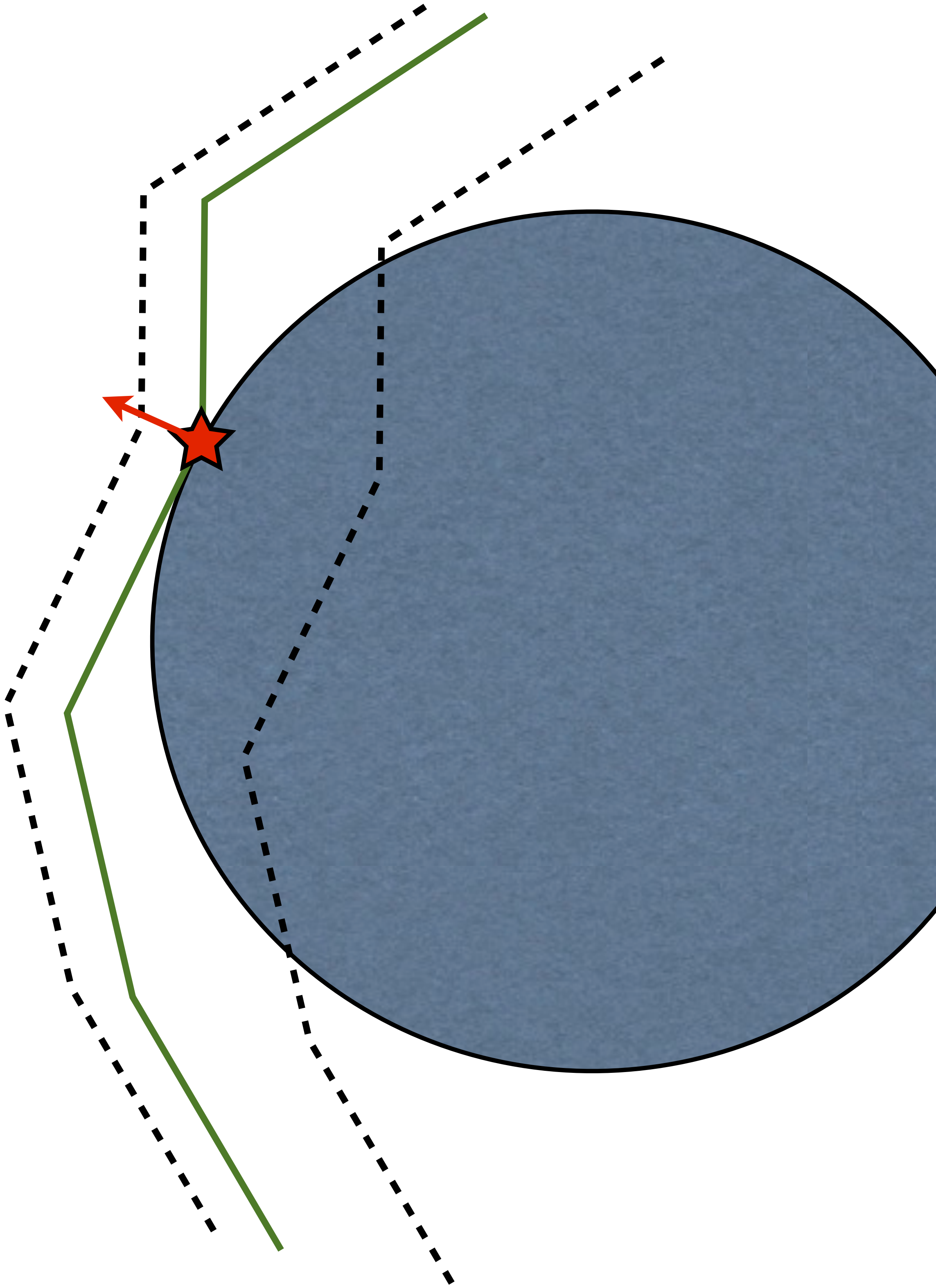
Collision response (general approaches)

- Continuous time collisions
  - Most "*physically justified*" technique : handle one collision event at a time, in the order they occur
  - Can be structured to pursue full avoidance of collisions, while not requiring collision objects to be thickened
  - Response can be formulated in terms of simple and intuitive penalty forces
  - Disadvantage : May lead to very small time steps



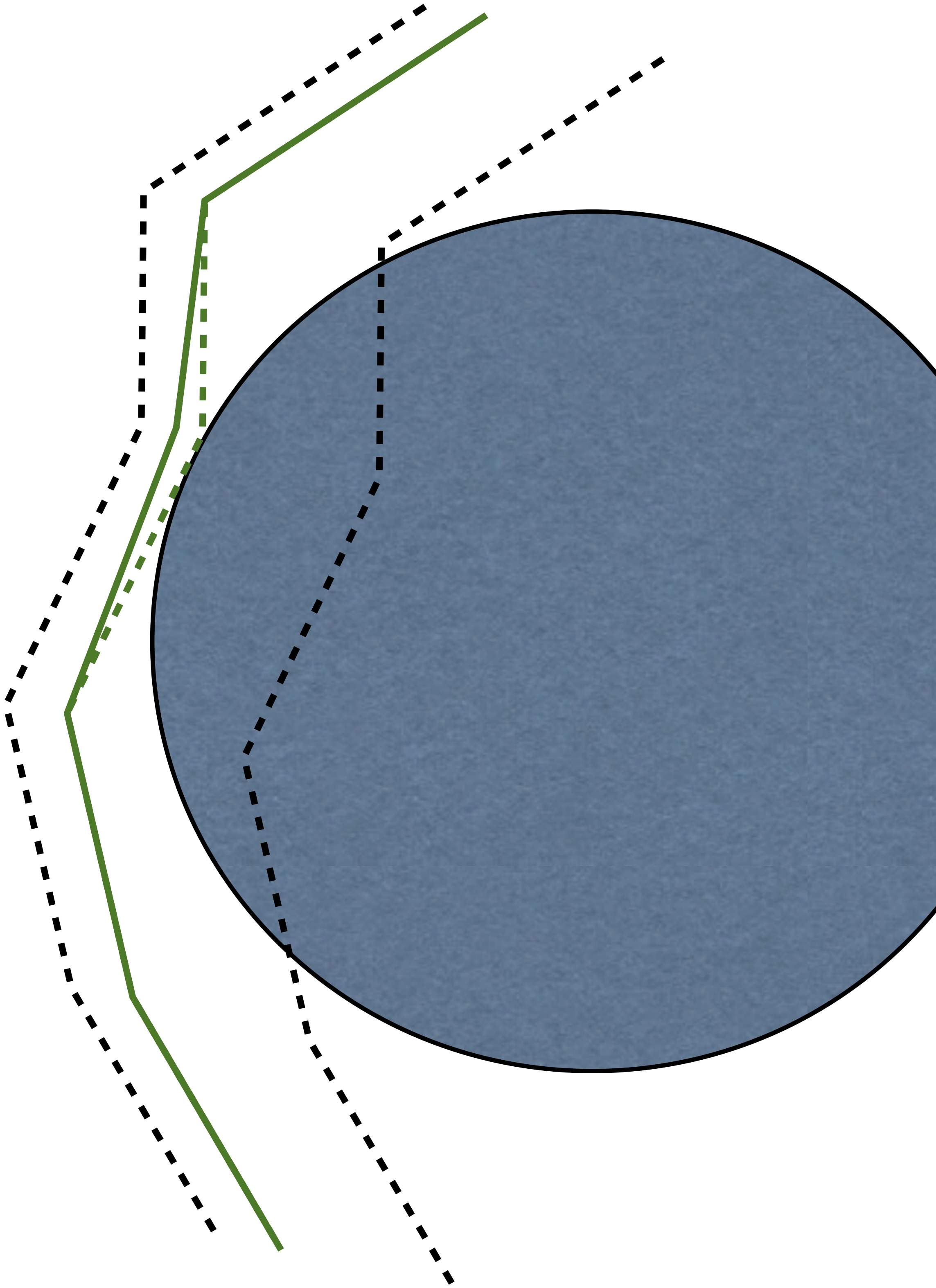
Collision response (general approaches)

- Continuous time collisions
  - Most "*physically justified*" technique : handle one collision event at a time, in the order they occur
  - Can be structured to pursue full avoidance of collisions, while not requiring collision objects to be thickened
  - Response can be formulated in terms of simple and intuitive penalty forces
  - Disadvantage : May lead to very small time steps



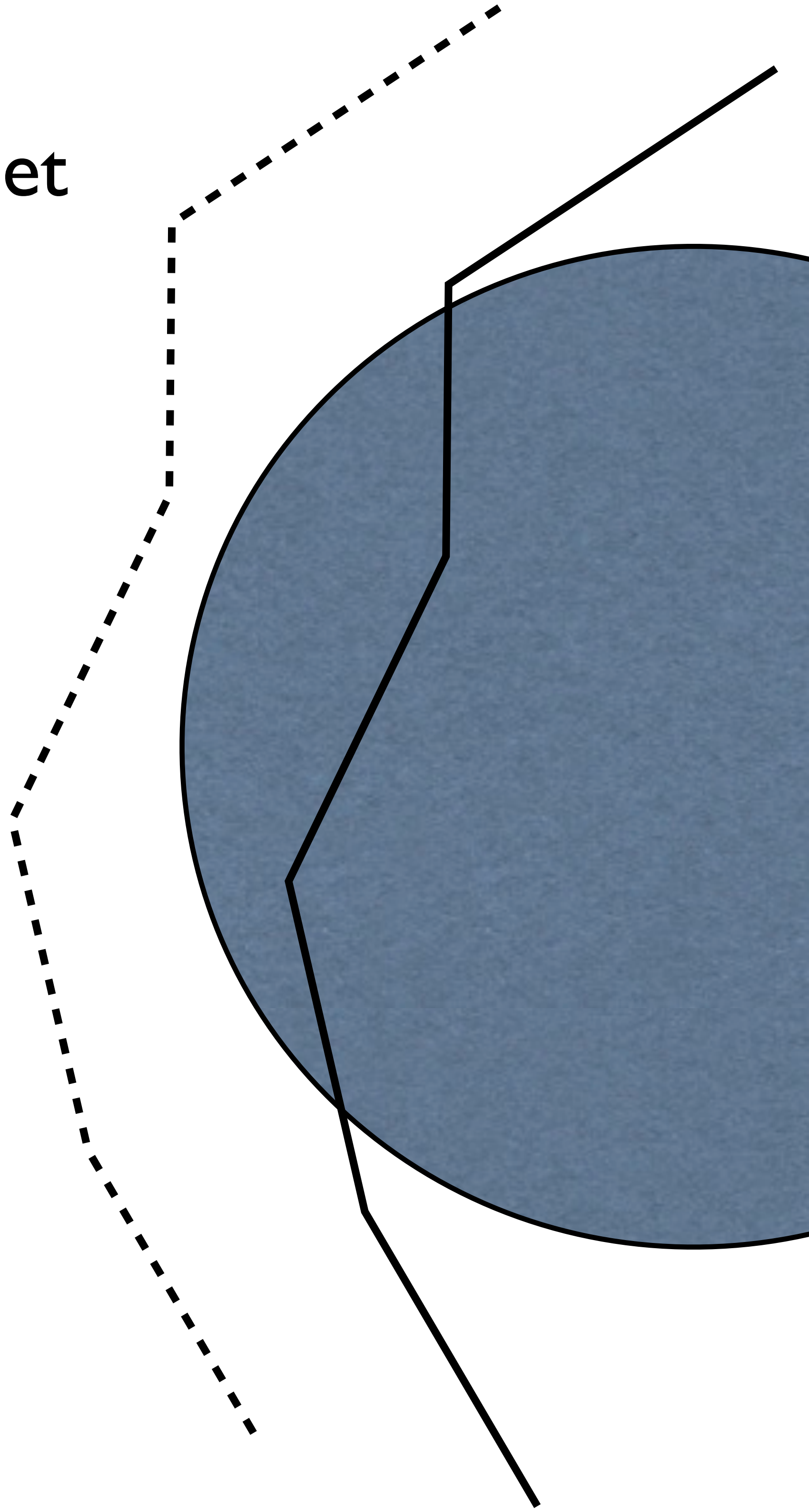
Collision response (general approaches)

- Continuous time collisions
  - Most "*physically justified*" technique : handle one collision event at a time, in the order they occur
  - Can be structured to pursue full avoidance of collisions, while not requiring collision objects to be thickened
  - Response can be formulated in terms of simple and intuitive penalty forces
  - Disadvantage : May lead to very small time steps



Collision detection (for kinematic objects)

- Simplest case: *Collision object is rigid*
  - We can pre-process the object into a levelset



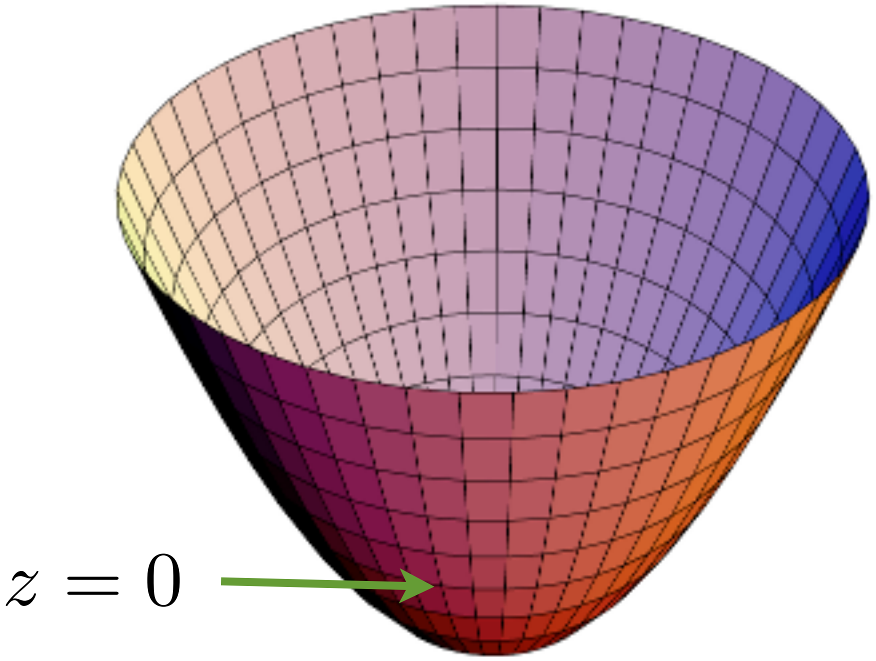
Represent a curve in 2D (or, a surface in 3D) as the zero isocontour of a (continuous) function, i.e.

$$\mathcal{C} = \{(x.y) \in \mathbf{R}^2 : \phi(x,y) = 0\}$$

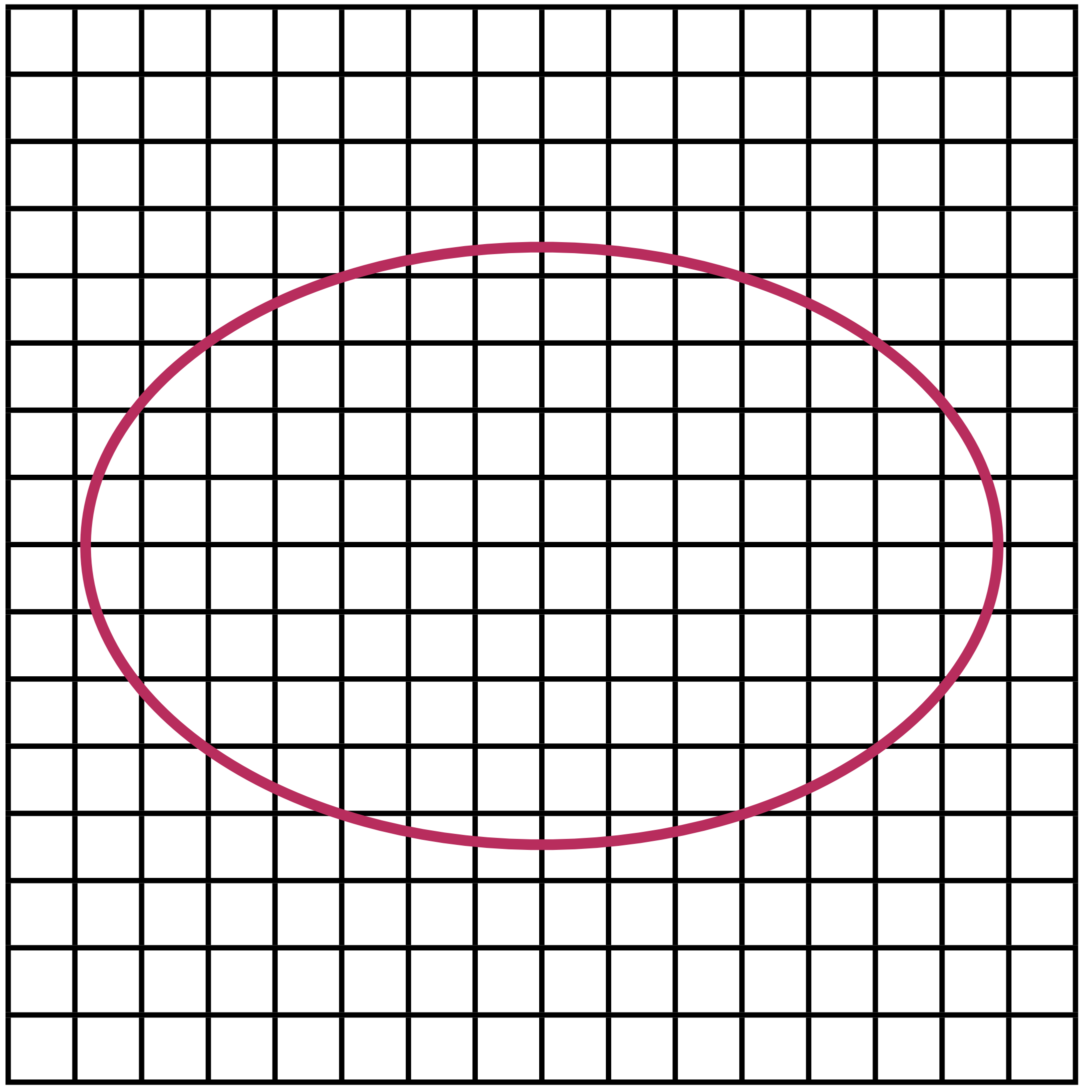
e.g.

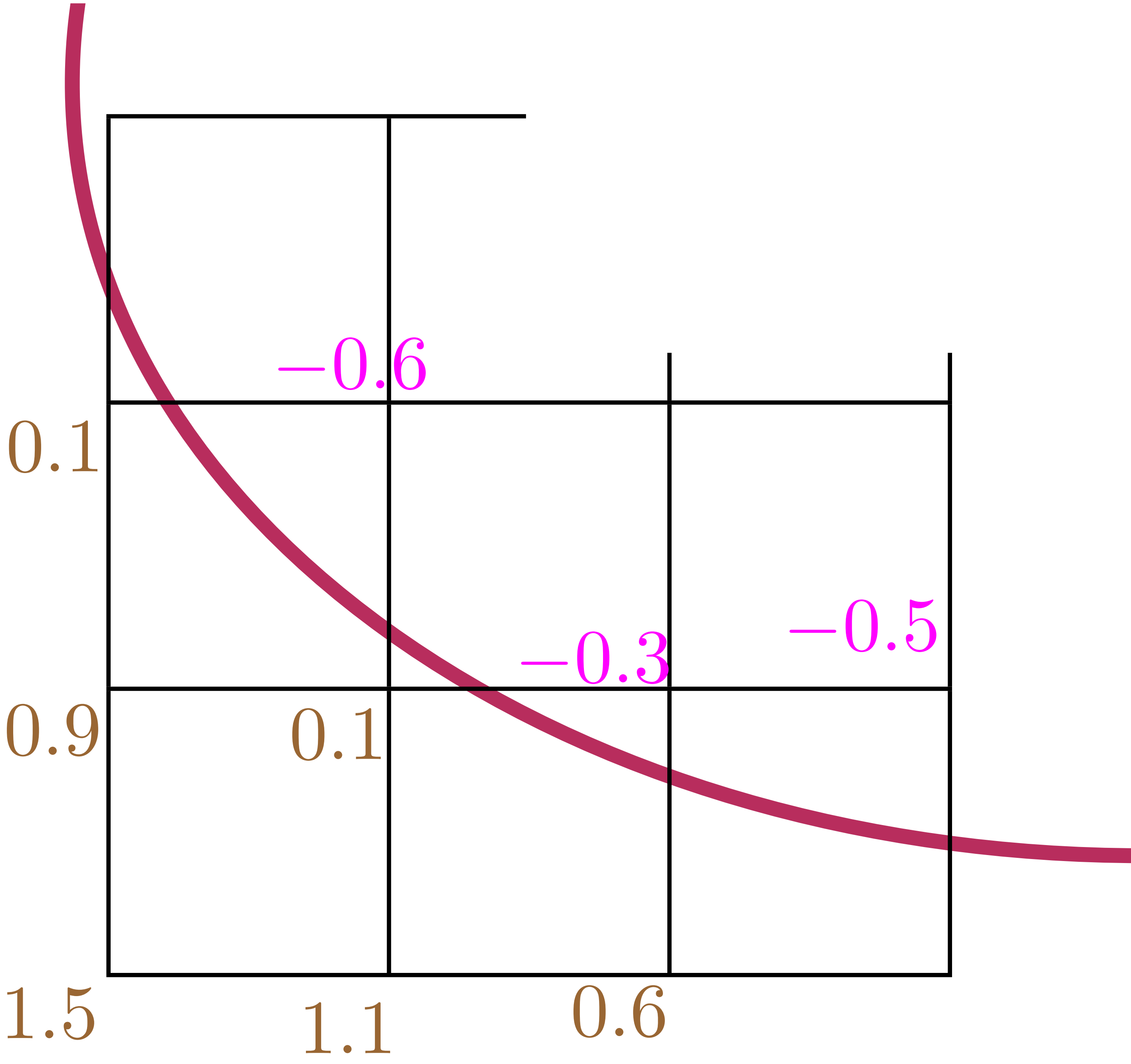
$$\text{circle } x^2 + y^2 = R^2 \equiv \{(x,y) : \phi(x,y) = 0\}$$

$$where\ \phi(x,y) = x^2 + y^2 - R^2$$



$$\left.\begin{array}{c} \phi(x,y) < 0, \text{ if } (x,y) \text{ is inside } \mathcal{C} \\ \phi(x,y) > 0, \text{ if } (x,y) \text{ is outside } \mathcal{C} \\ \phi(x,y) = 0, \text{ if } (x,y) \text{ is on } \mathcal{C} \end{array}\right\} \text{ and } |\phi(x,y)| = \text{distance of } (x,y) \text{ from } \mathcal{C}$$





Collision detection (for kinematic objects)

- Simplest case: *Collision object is rigid*
    - We can pre-process the object into a levelset
    - Query : Is a point x* *colliding* with the object?
        - ➡ Yes, if and only if $\phi(x^*) < 0$

Collision detection (for kinematic objects)

- Simplest case: *Collision object is rigid*
  - We can pre-process the object into a levelset
  - Query : Is a point x* *colliding* with the object?
    - ➡ Yes, if and only if $\phi(x^*) < 0$
  - Query : Is a point x* *within a distance D* of the object?
    - ➡ Yes, if and only if $\phi(x^*) < D$

Collision detection (for kinematic objects)

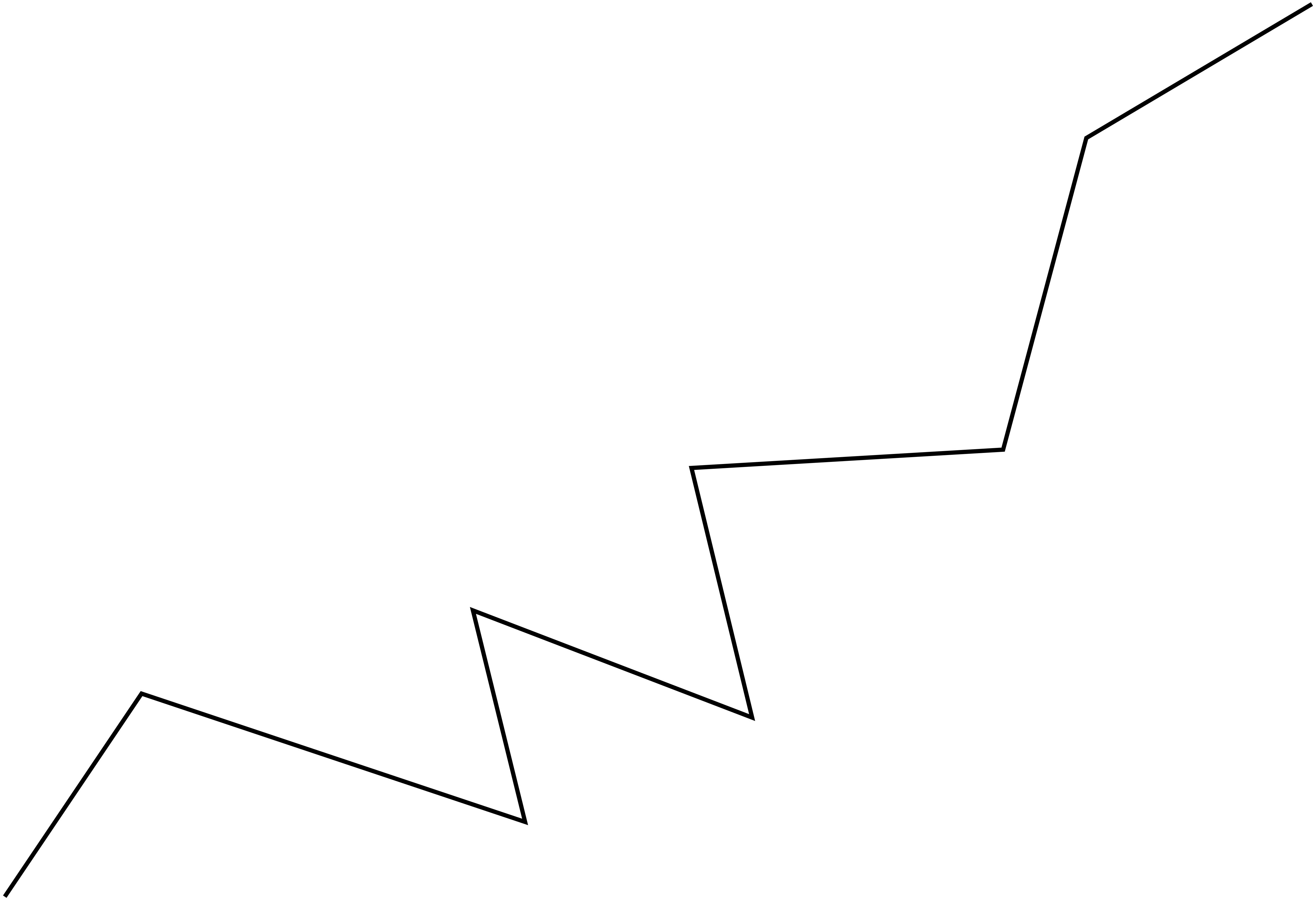
- Simplest case: *Collision object is rigid*
  - We can pre-process the object into a levelset
  - Query : Is a point x* *colliding* with the object?
    - ➡ Yes, if and only if $\phi(x^*) < 0$
  - Query : Is a point x* *within a distance D* of the object?
    - ➡ Yes, if and only if $\phi(x^*) < D$
  - Query : What is the direction of the collision (i.e. what is the "shortest way out"?)
    - ➡ Given by the vector $\nabla\phi(x^*)$

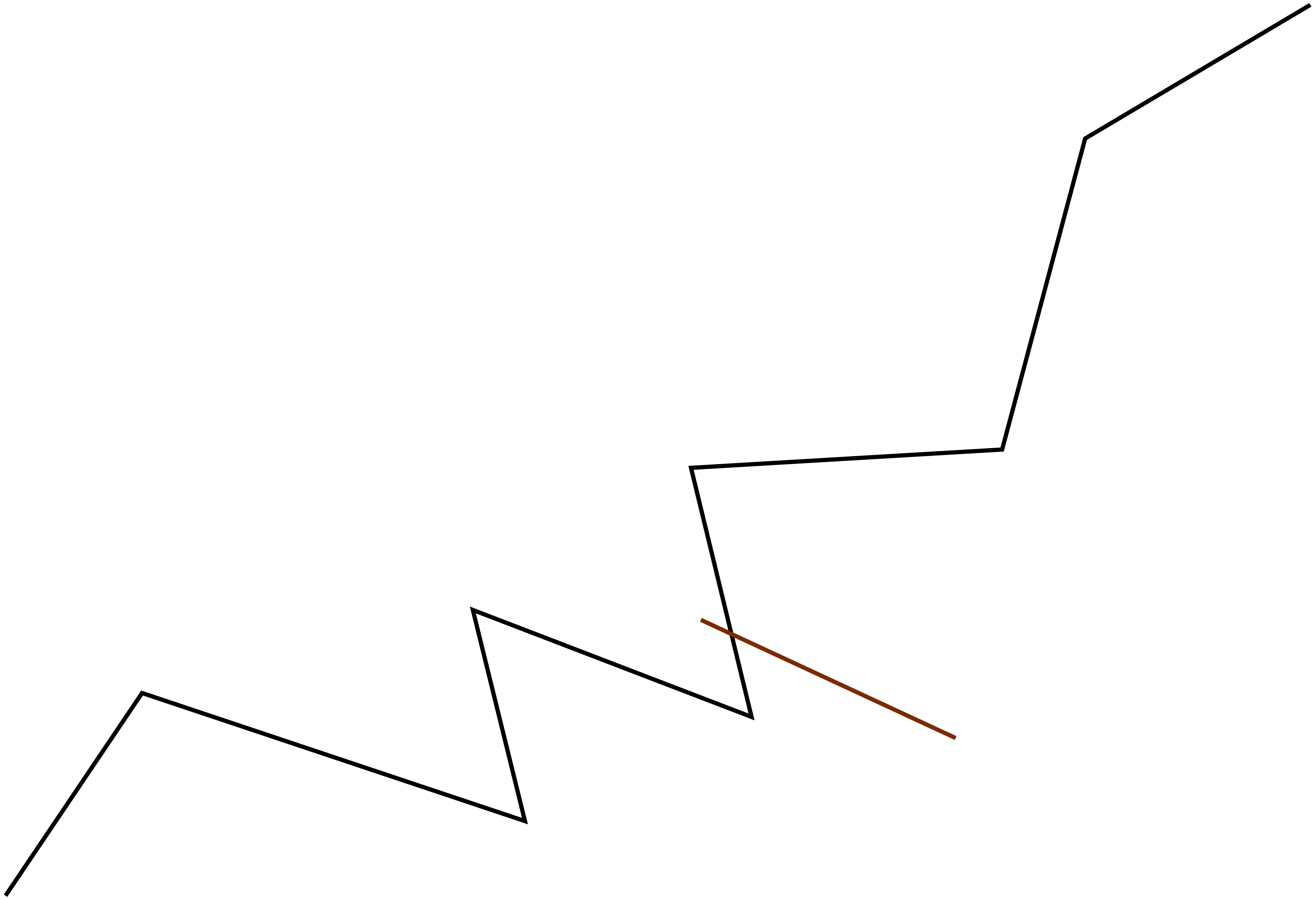
Collision detection (for kinematic objects)

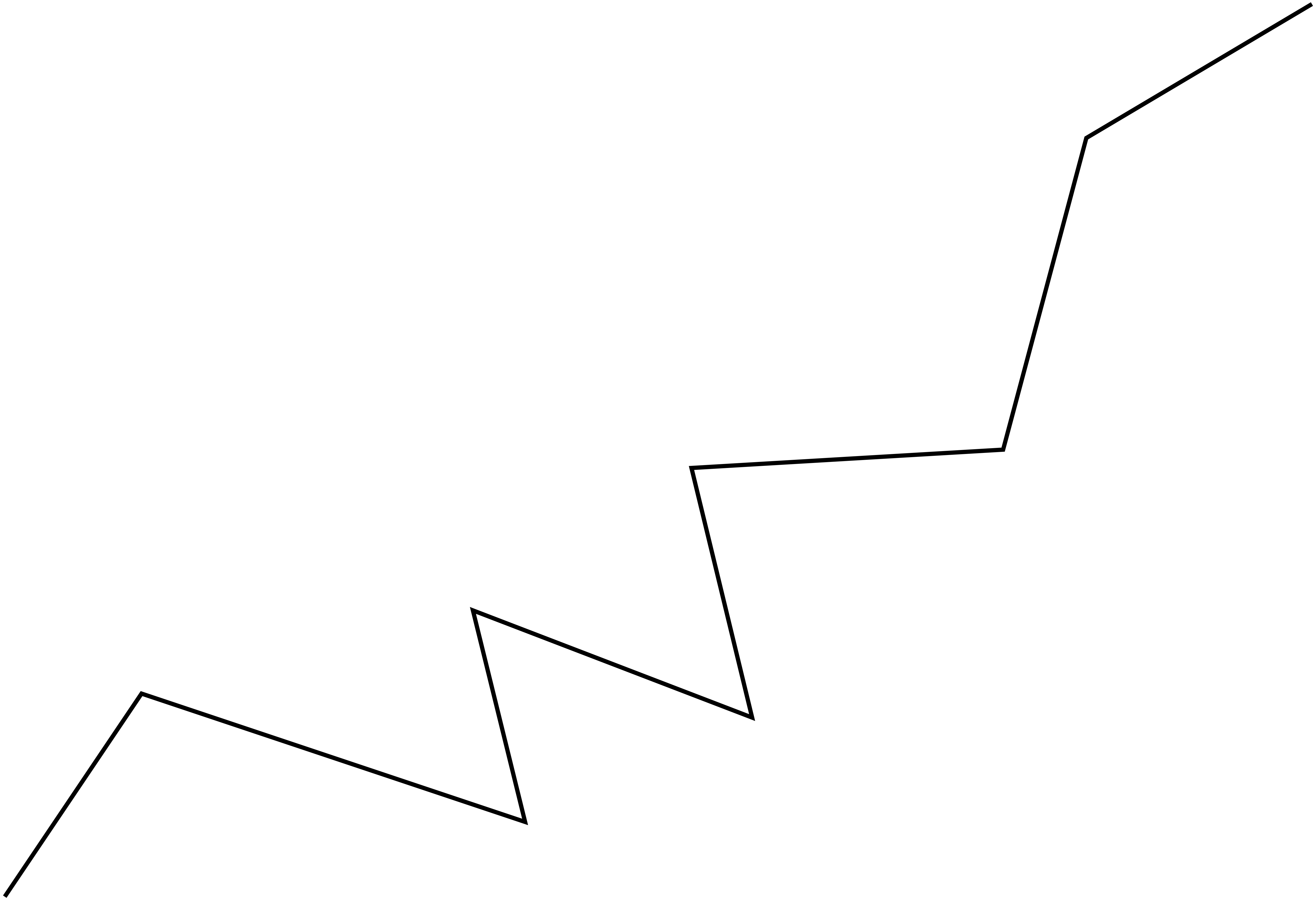
- Simplest case: *Collision object is rigid*
  - We can pre-process the object into a levelset
  - Query : Is a point x* *colliding* with the object?
    - ➡ Yes, if and only if $\phi(x^*) < 0$
  - Query : Is a point x* *within a distance D* of the object?
    - ➡ Yes, if and only if $\phi(x^*) < D$
  - Query : What is the direction of the collision (i.e. what is the "shortest way out"?)
    - ➡ Given by the vector $\nabla\phi(x^*)$
  - Query : What point on the surface of the object is closest to x*?
    - ➡ Given as $x_{\text{surface}} = x^* - \phi(x^*)\nabla\phi(x^*)$

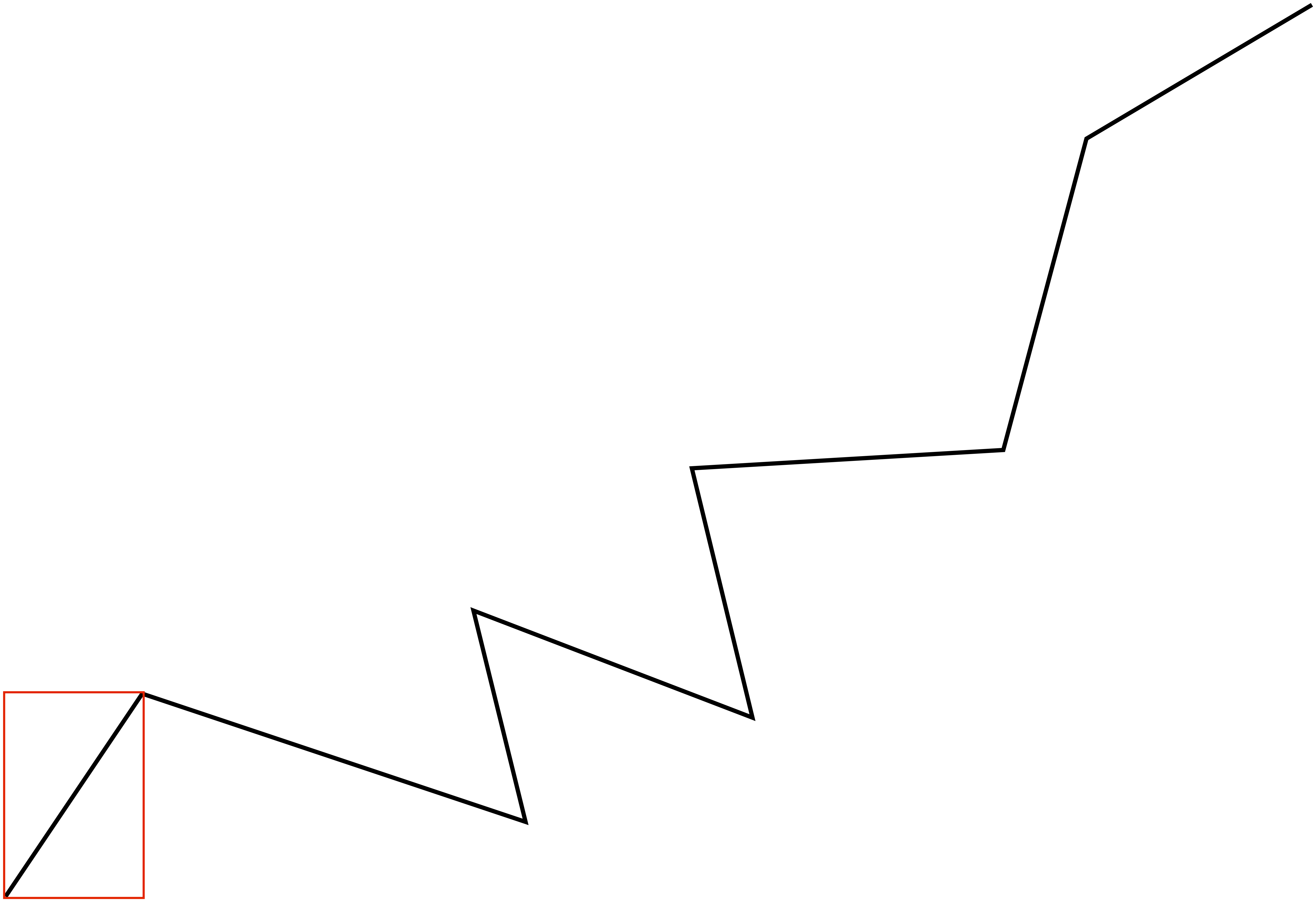
Collision detection (for simulated objects)

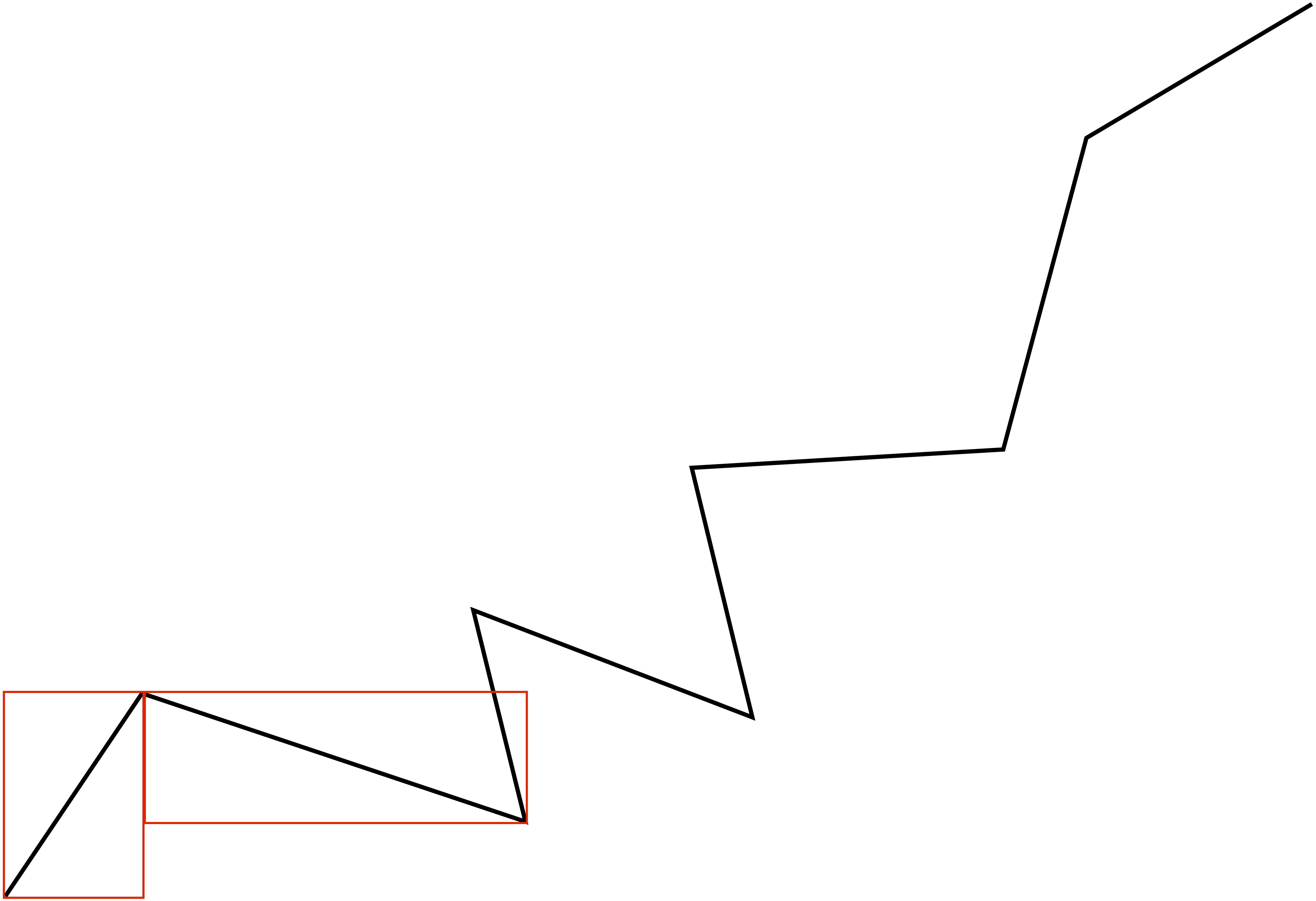
- Cannot (easily and efficiently) convert into levelsets to facilitate O(1) collision queries
  - Sometimes we seek collisions between open surfaces, which do not have an *"interior"* to describe as a levelset
- If simulation contains N primitives (particles, segments, triangles, etc) there is a potential for O(N^2) *"candidate"* intersection pairs
  - Brute force check would require O(N^2) cost
  - Every simulation step ideally requires O(N) effort (e.g. with Forward Euler, or BE with fixed CG iterations)
  - Ideally the detection cost should not exceed O(N) by much
- Popular approach : Using axis-aligned bounding box (AABB) queries to accelerate collision detection

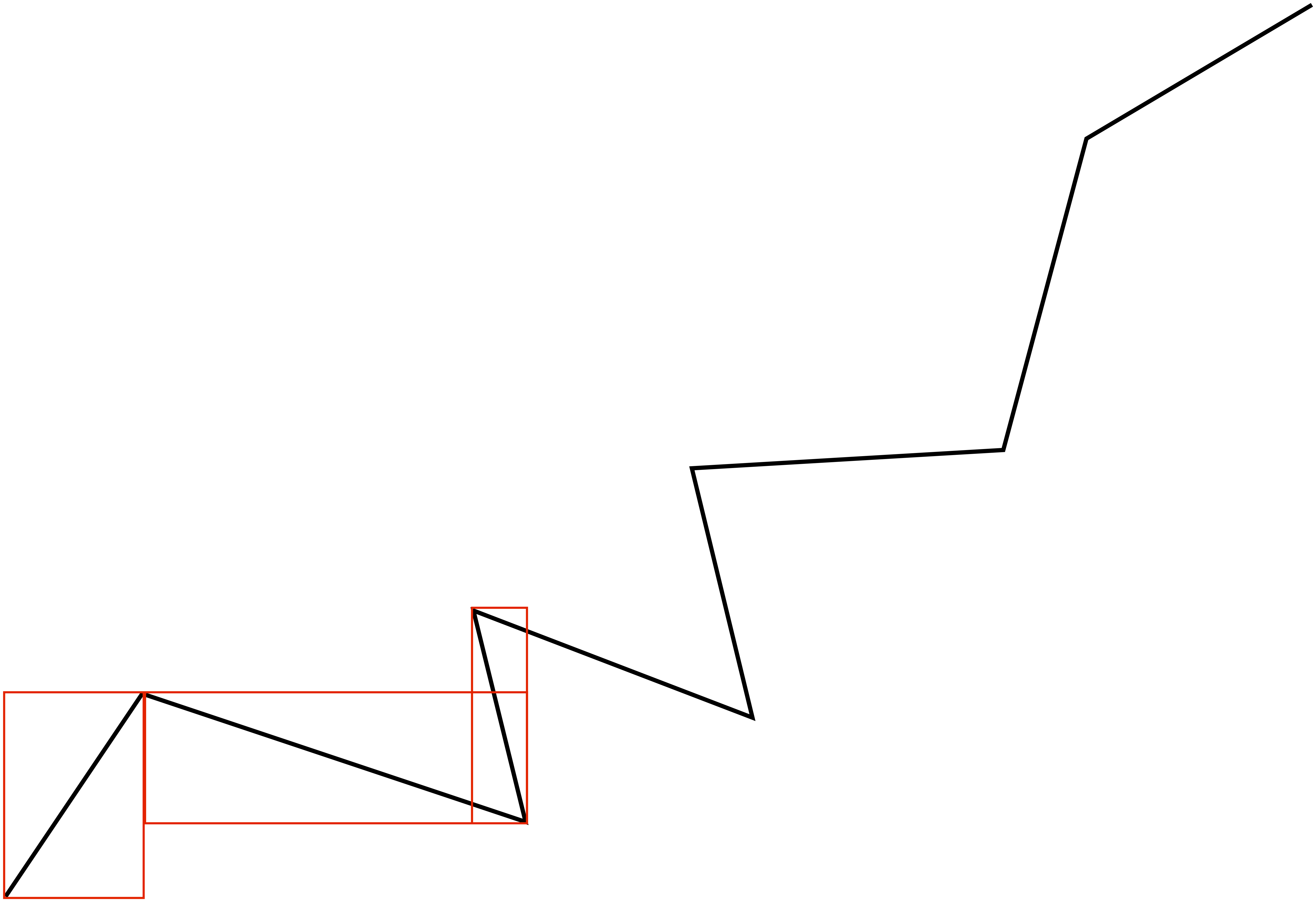


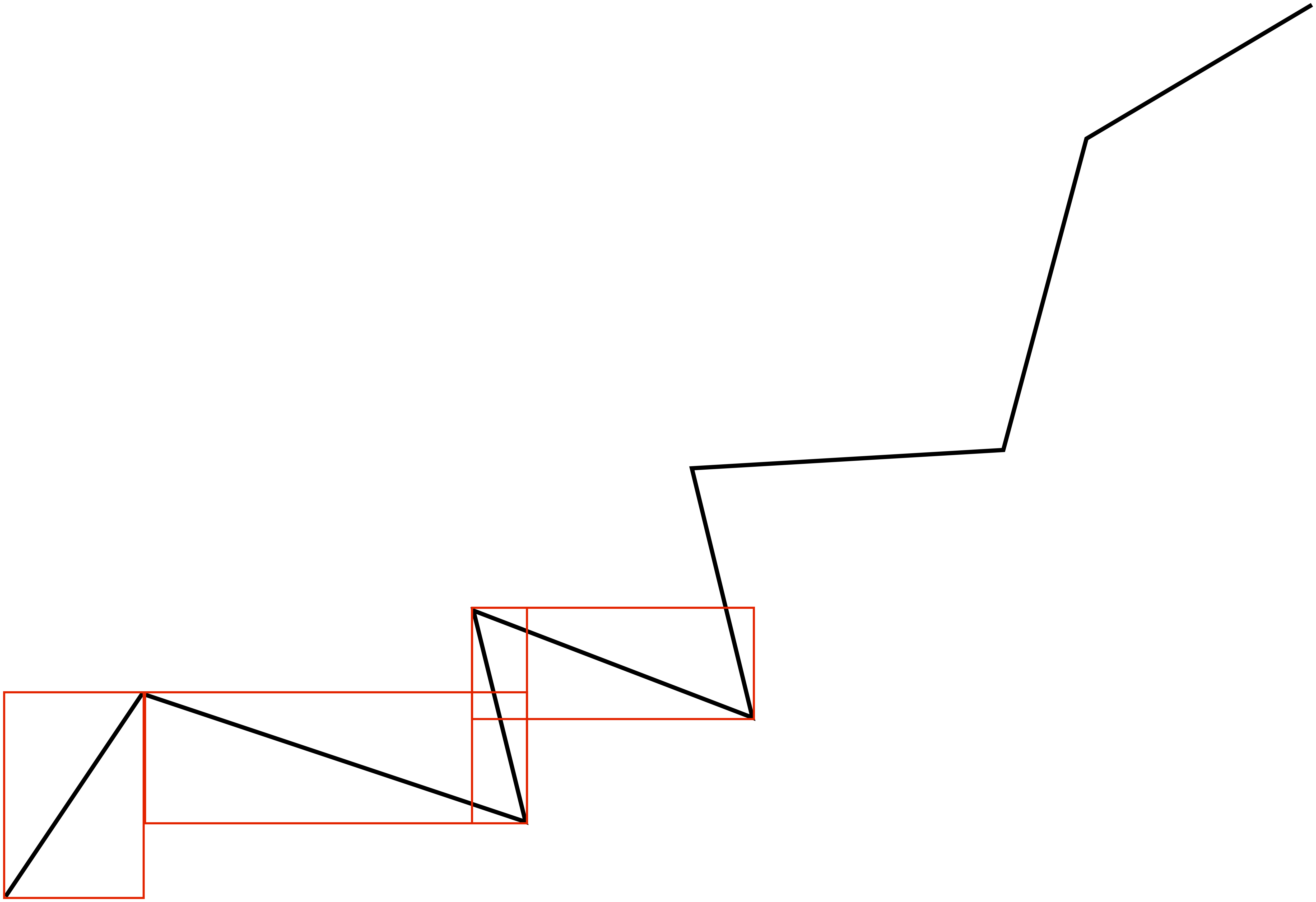


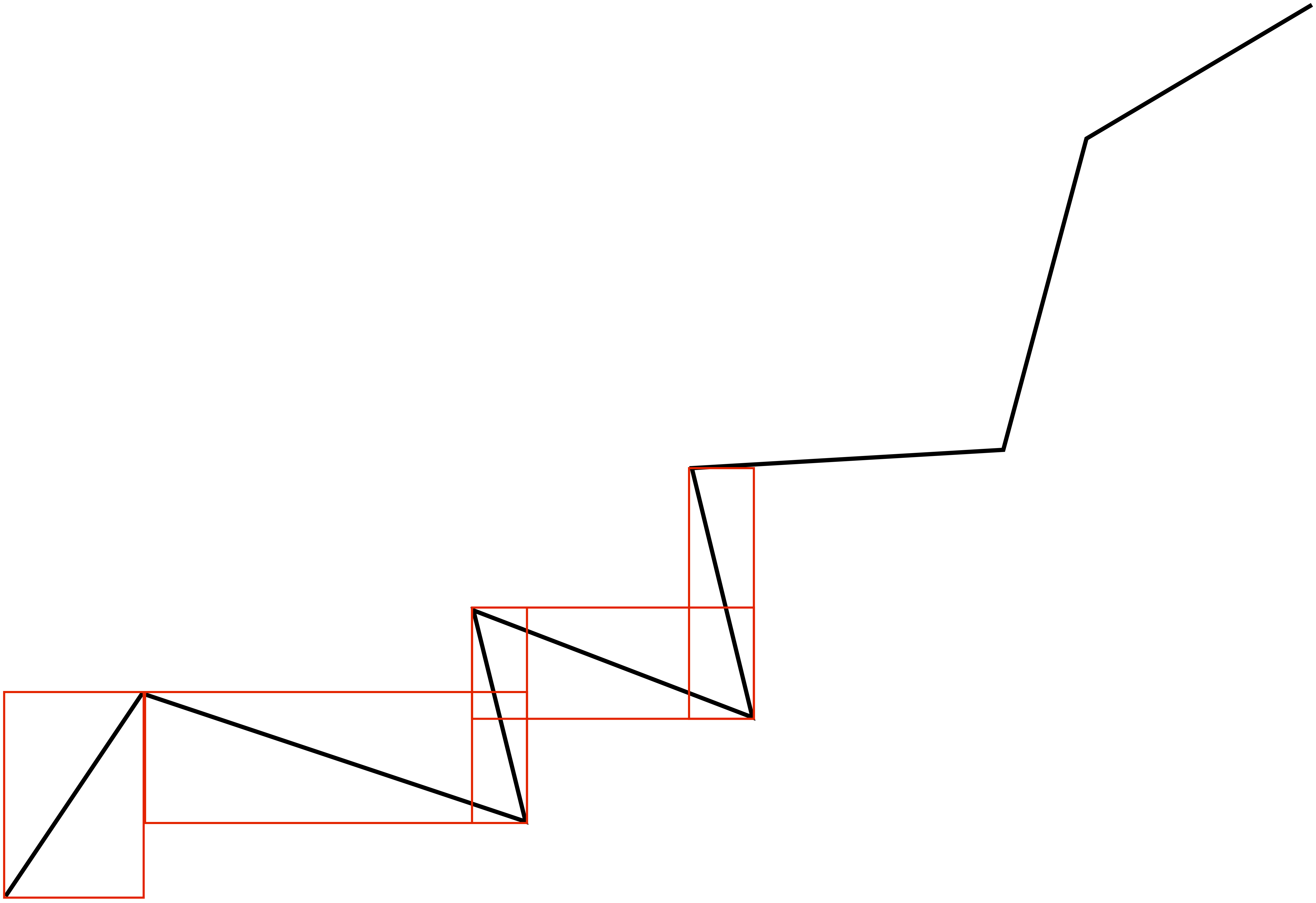


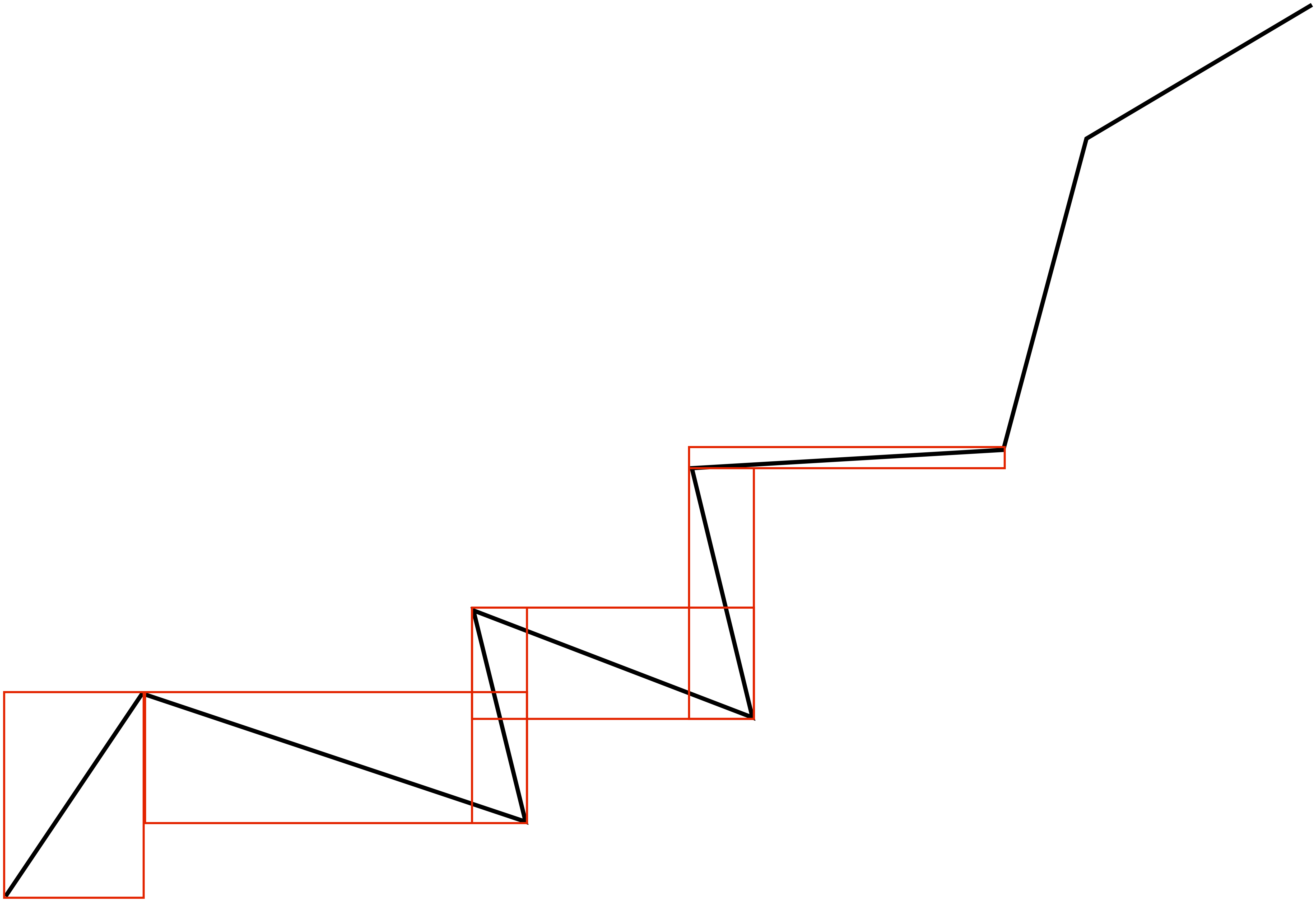


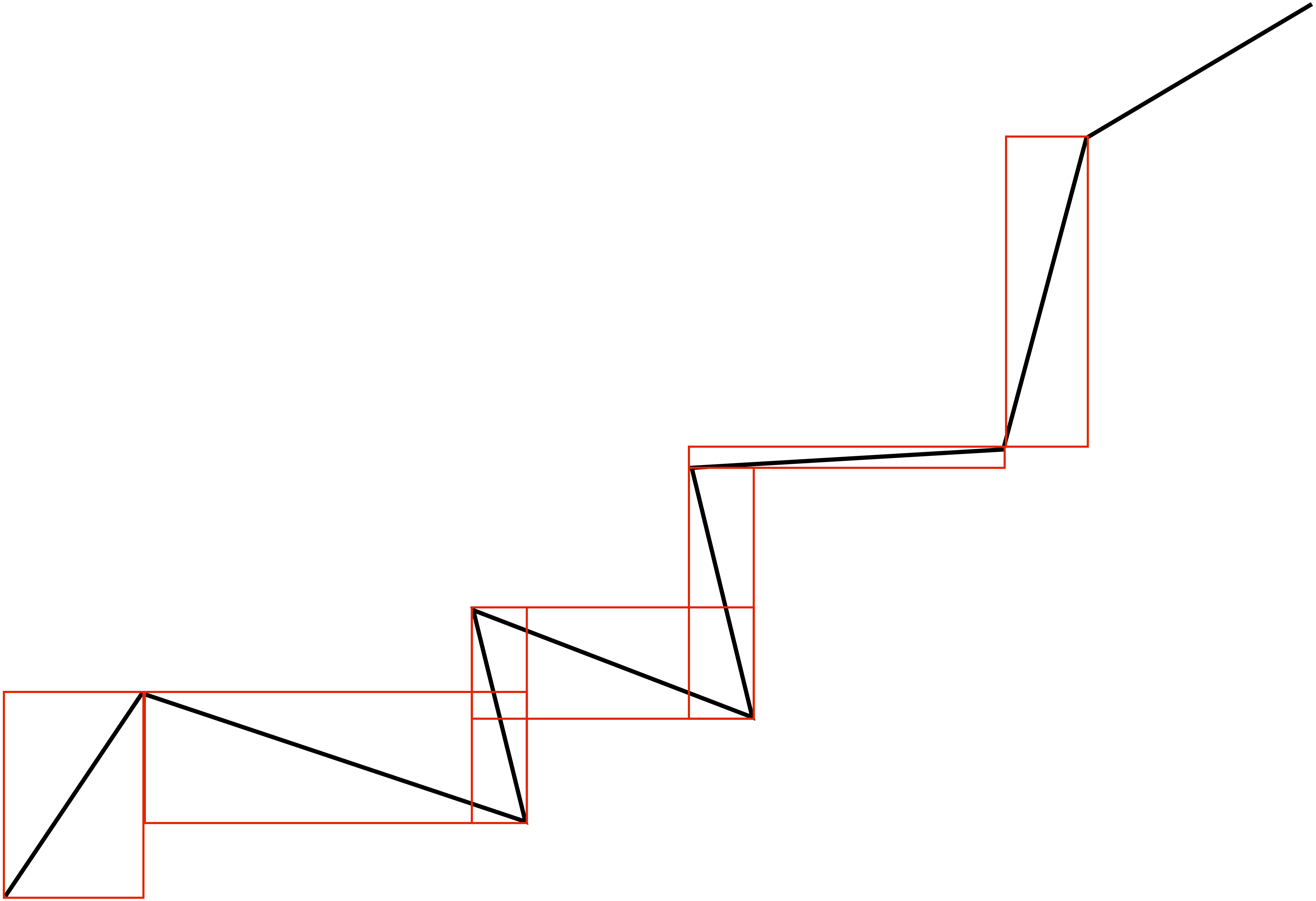


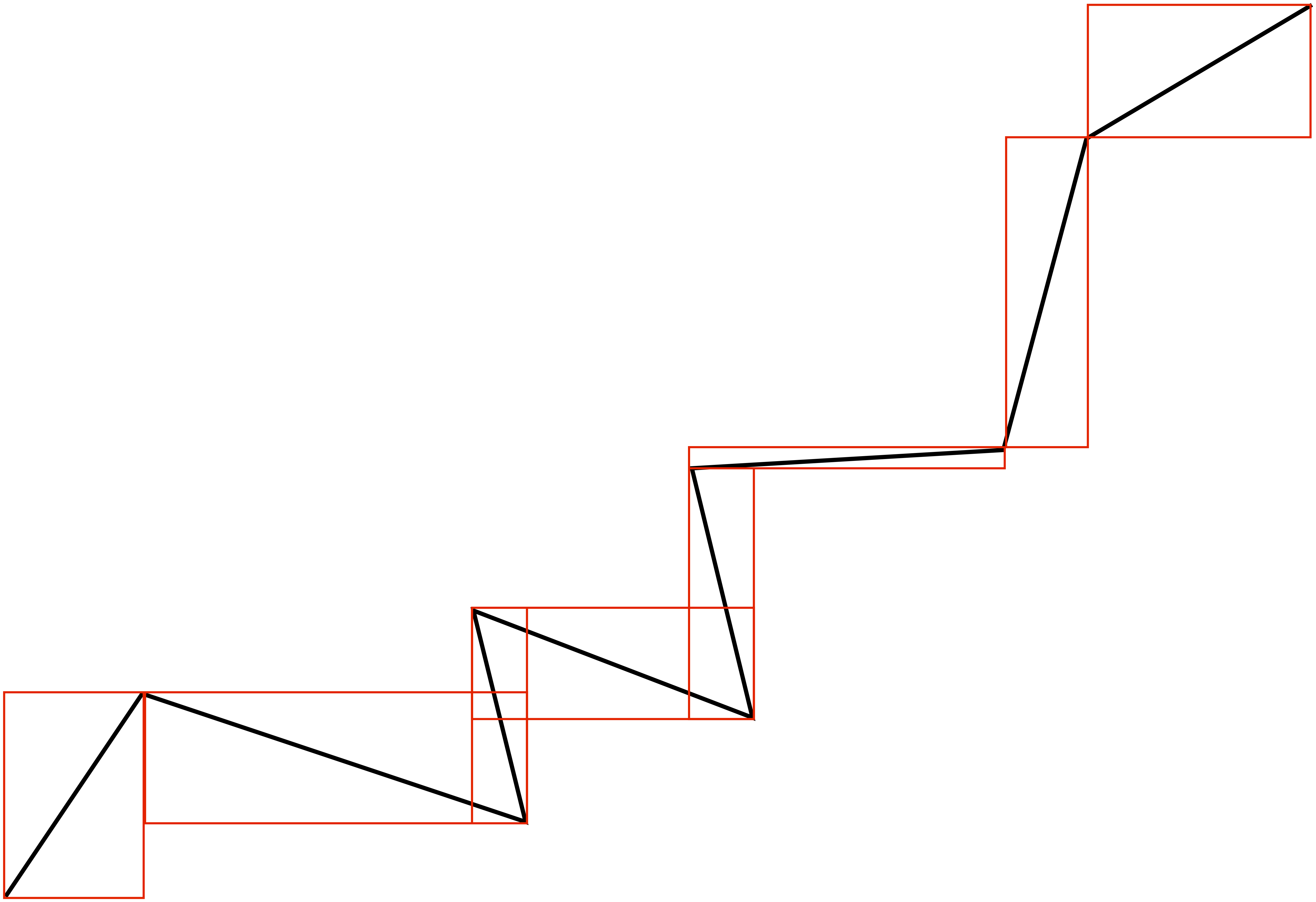


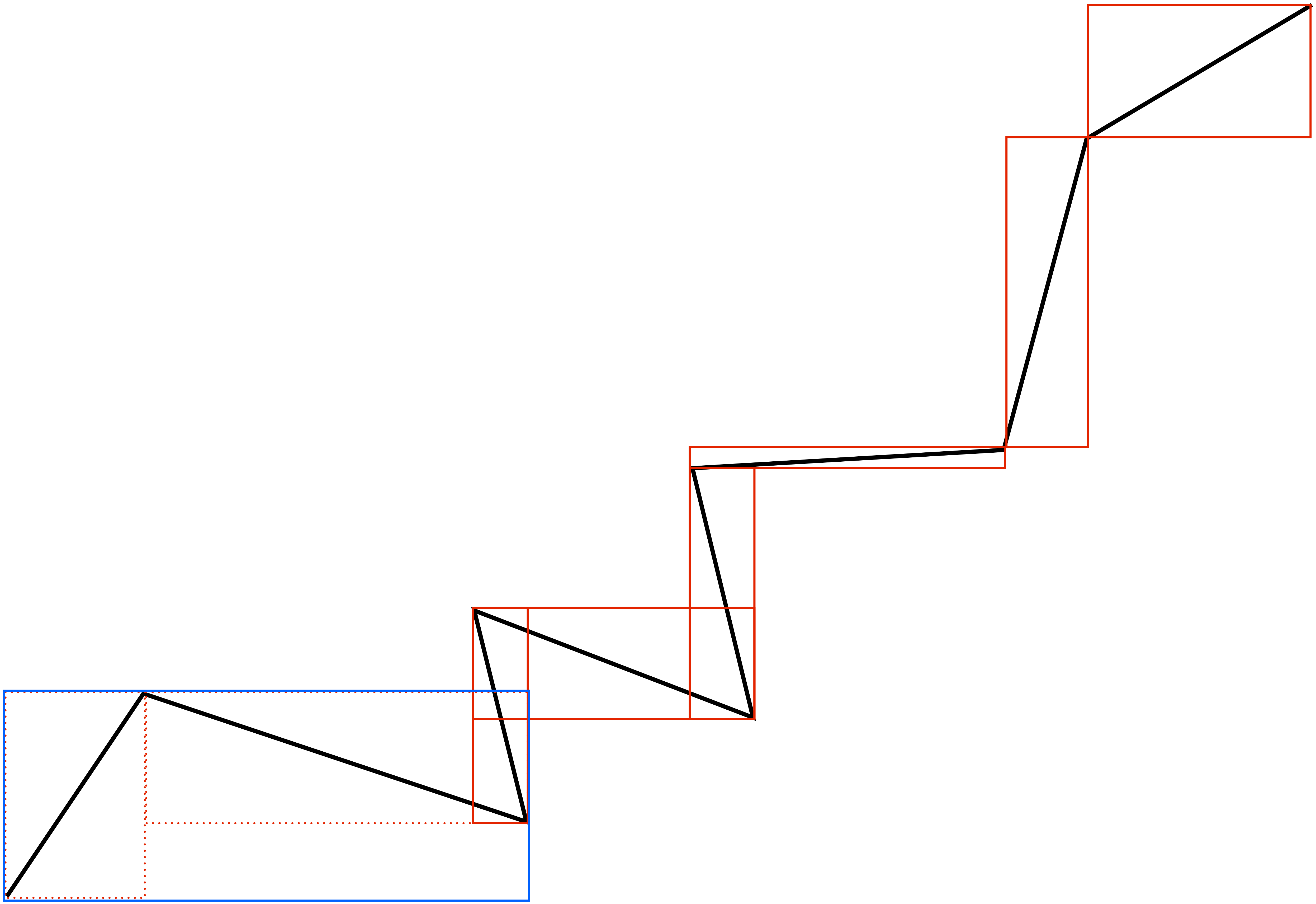


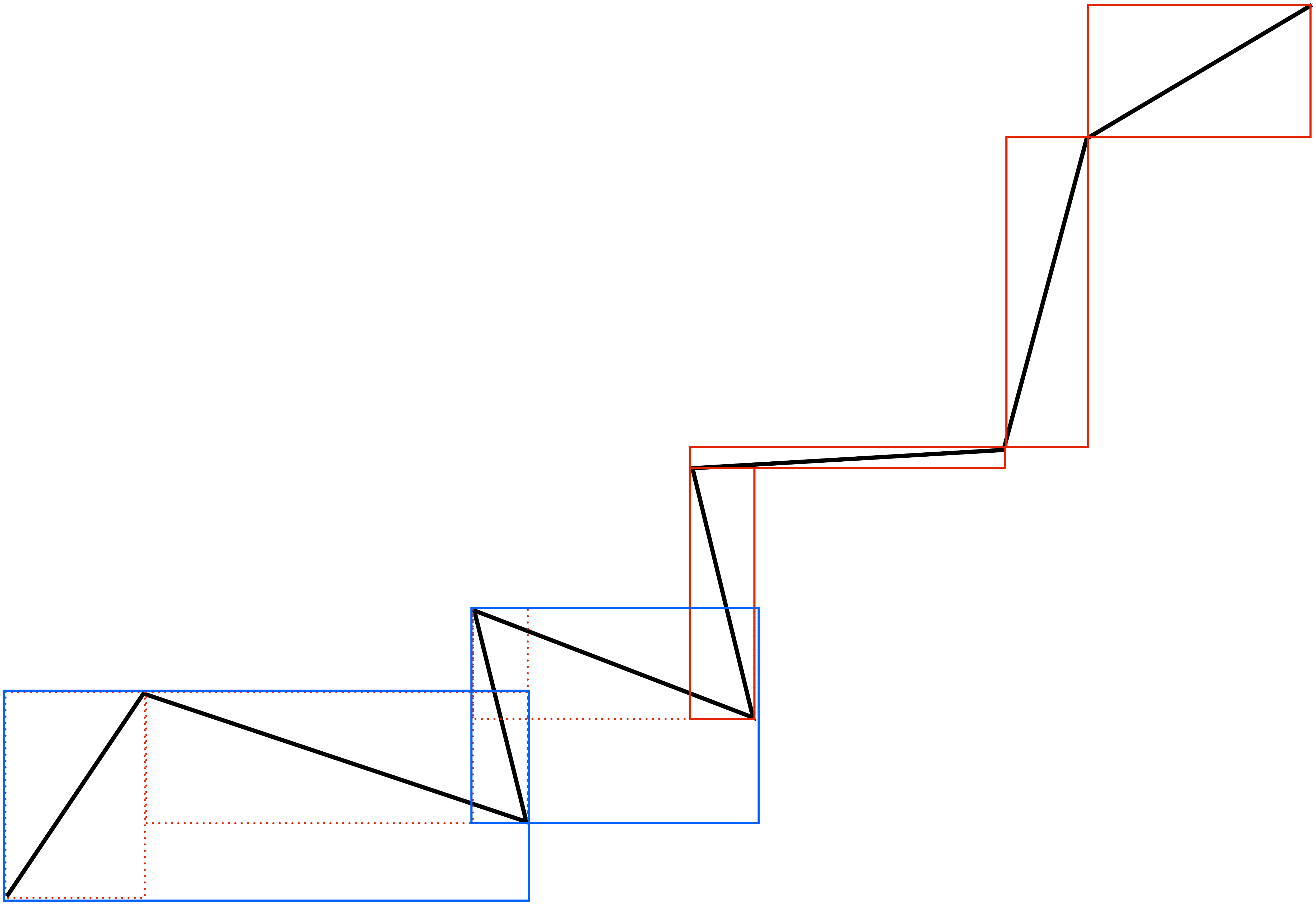


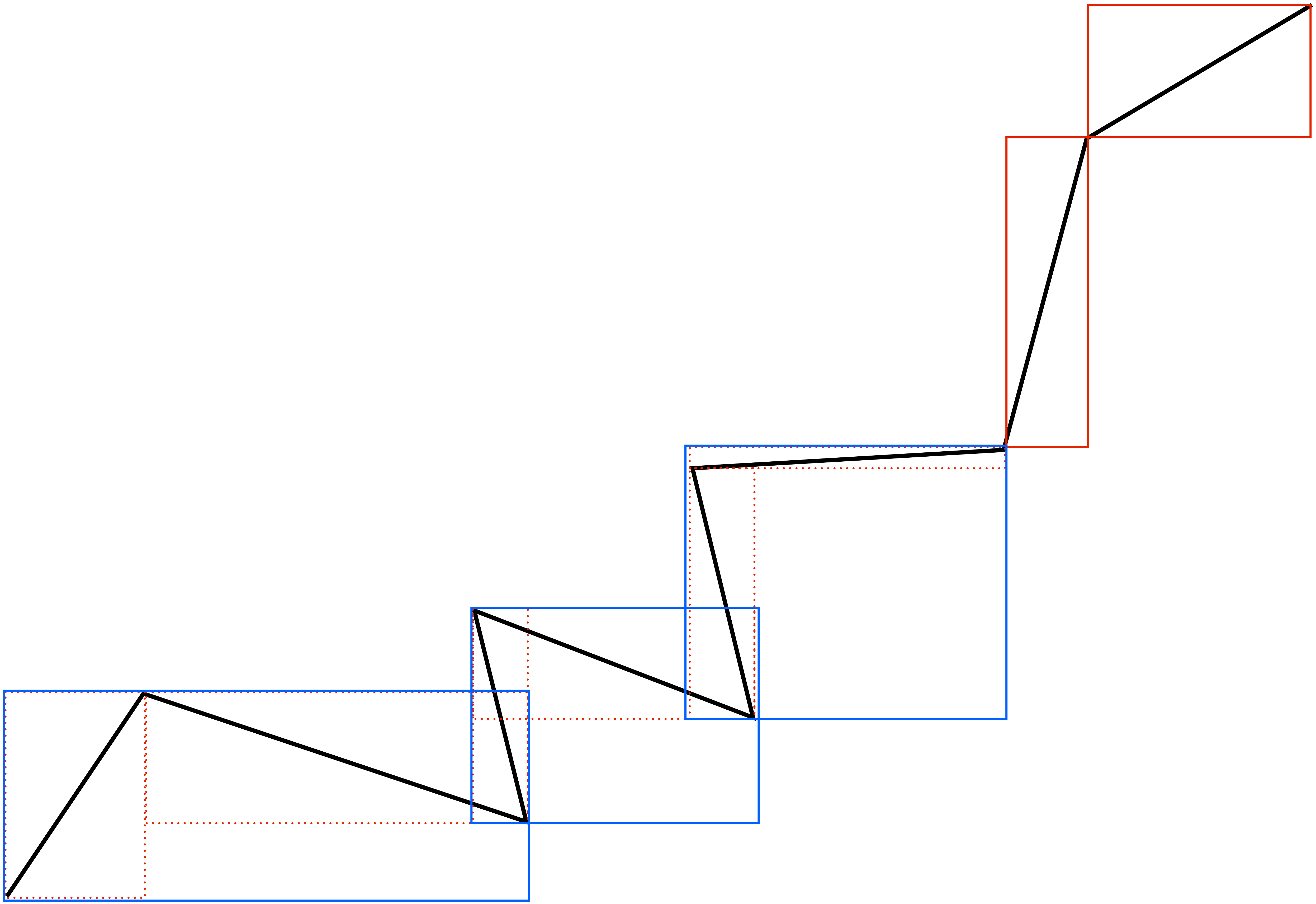


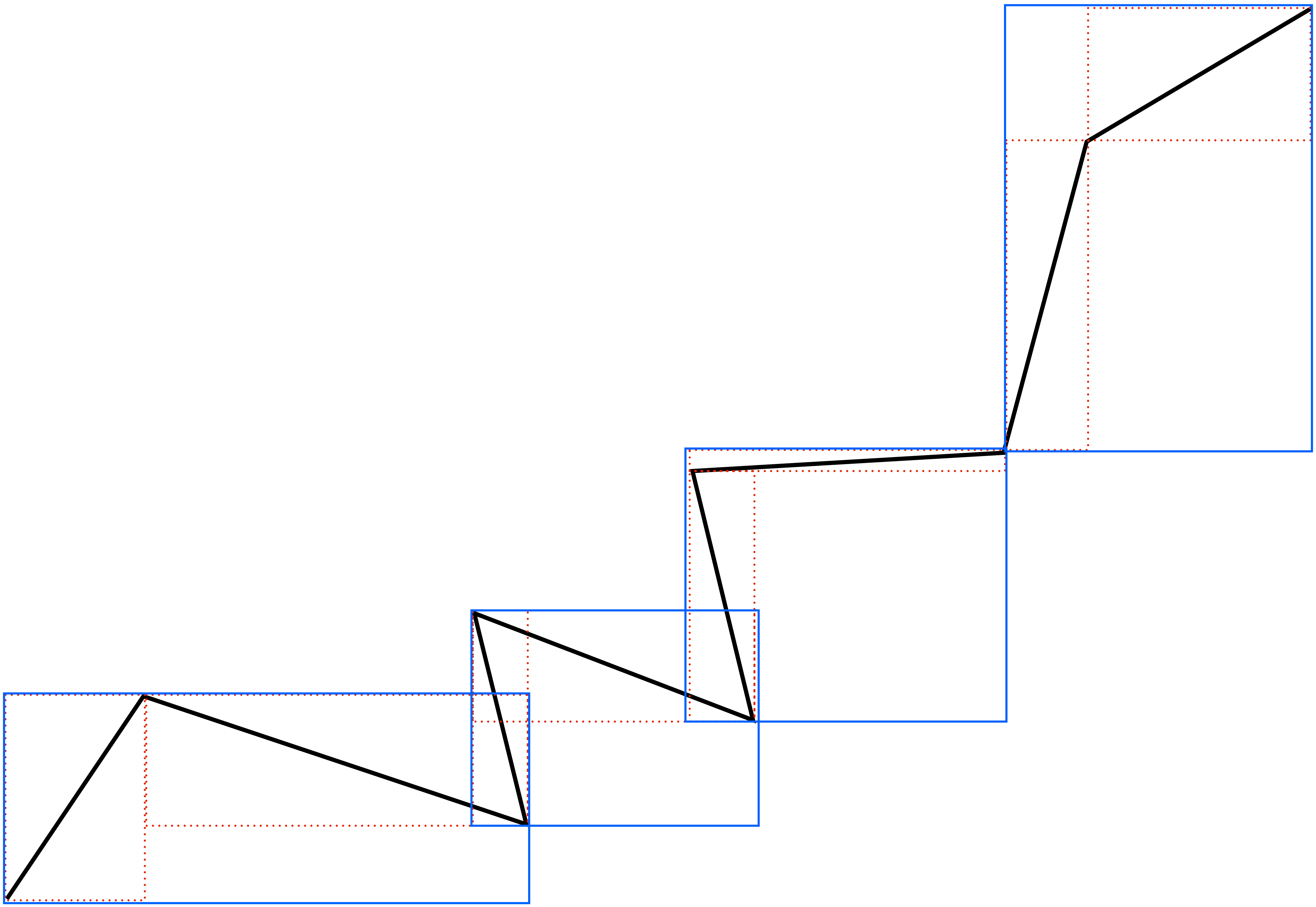


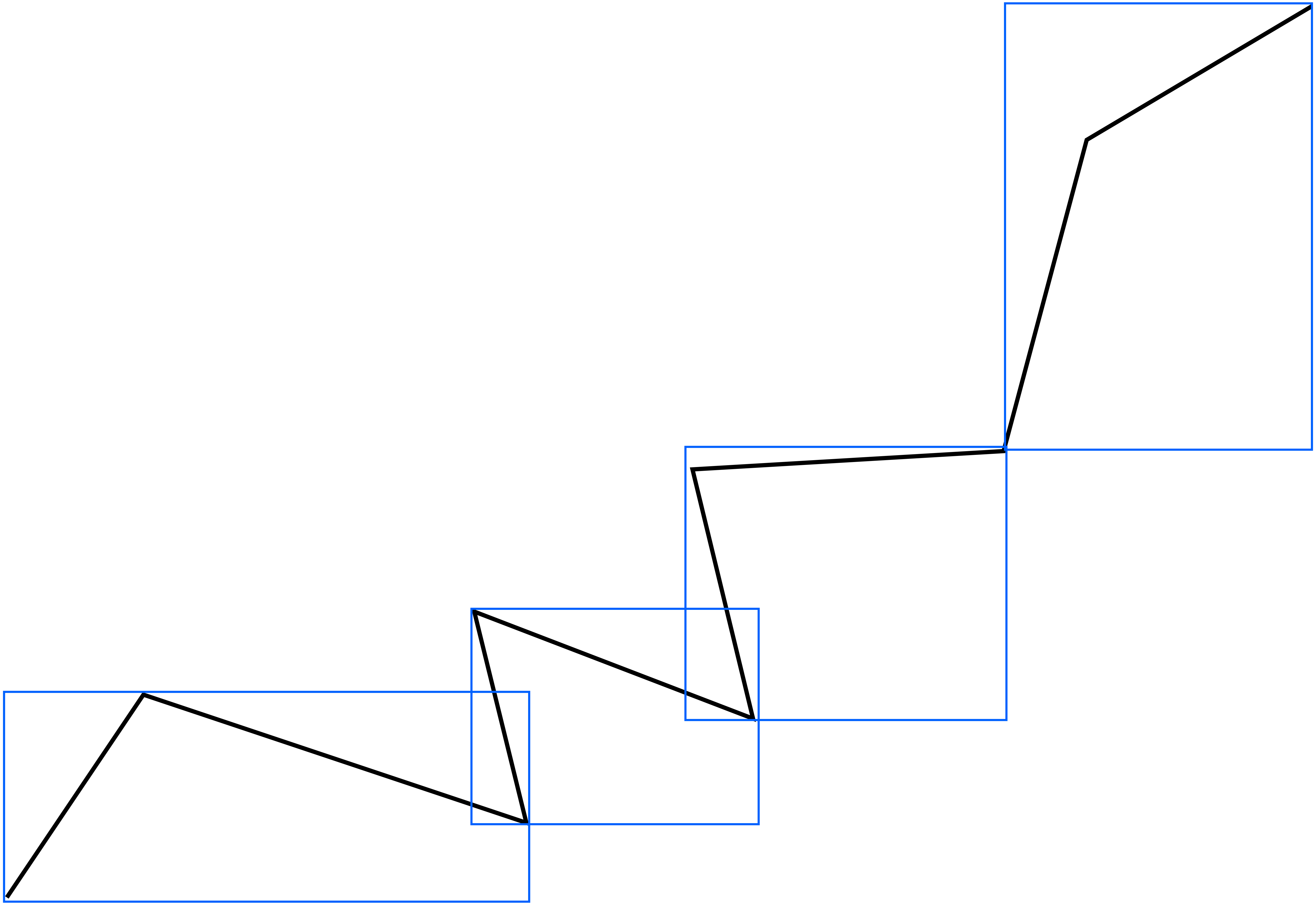


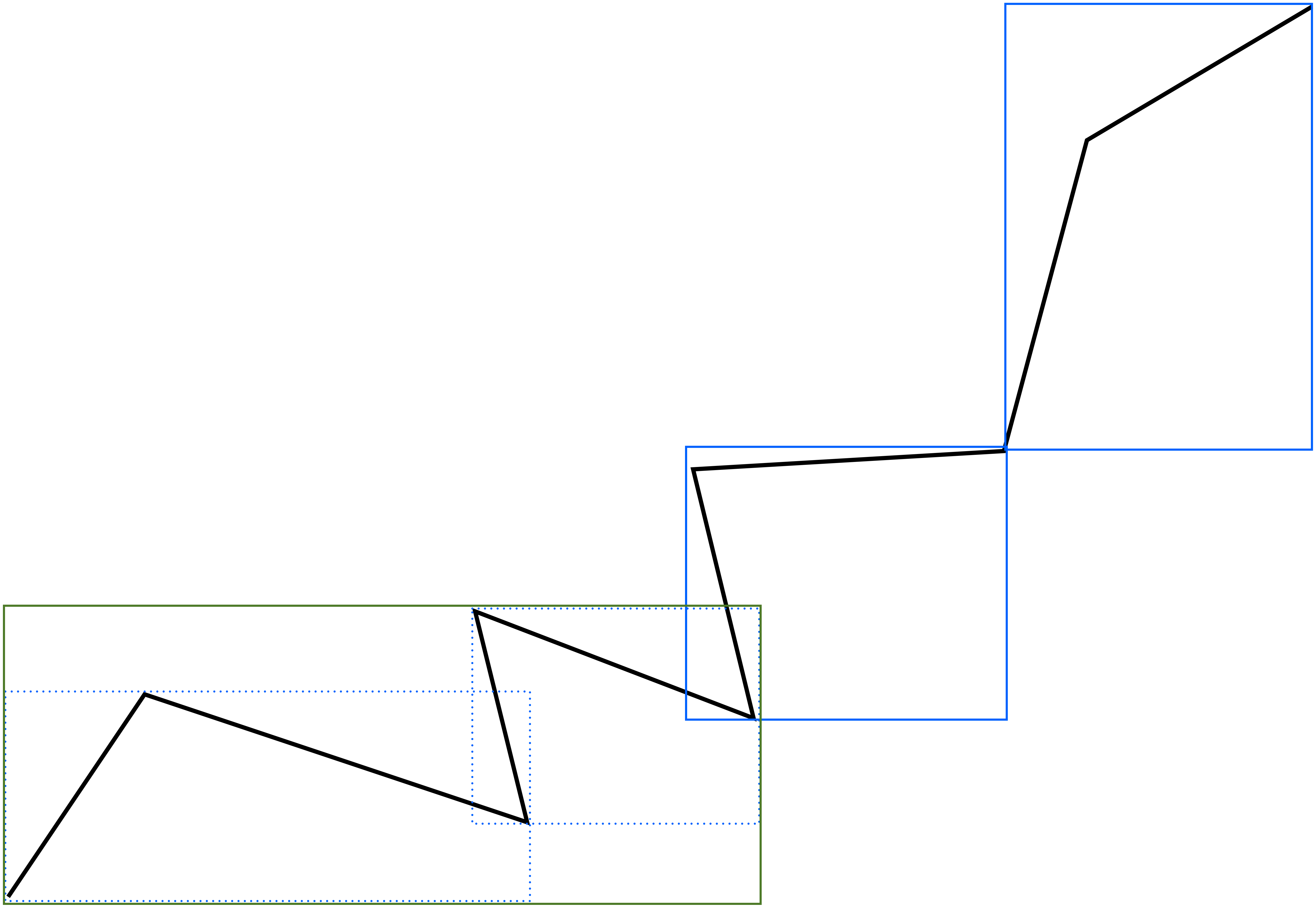


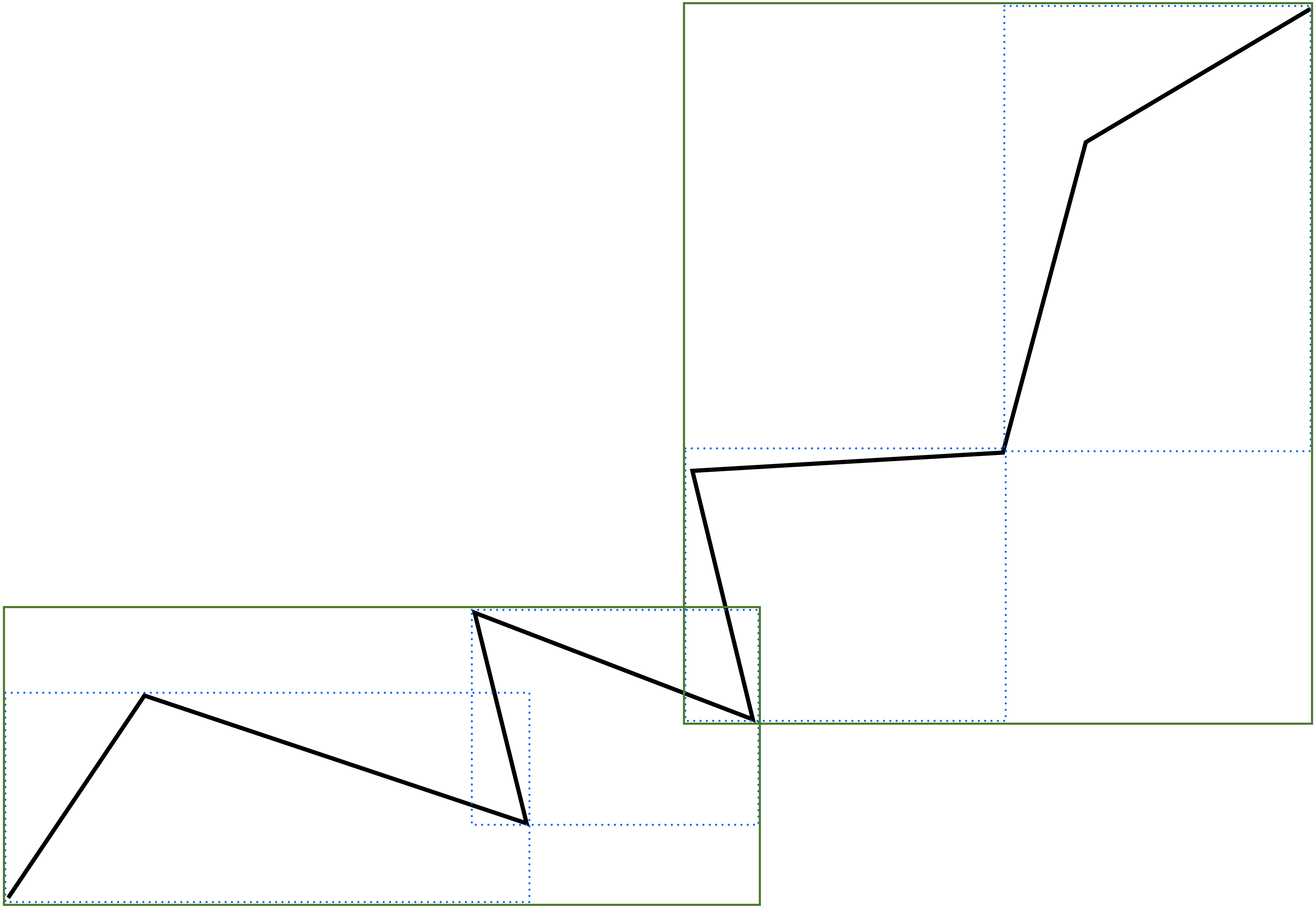


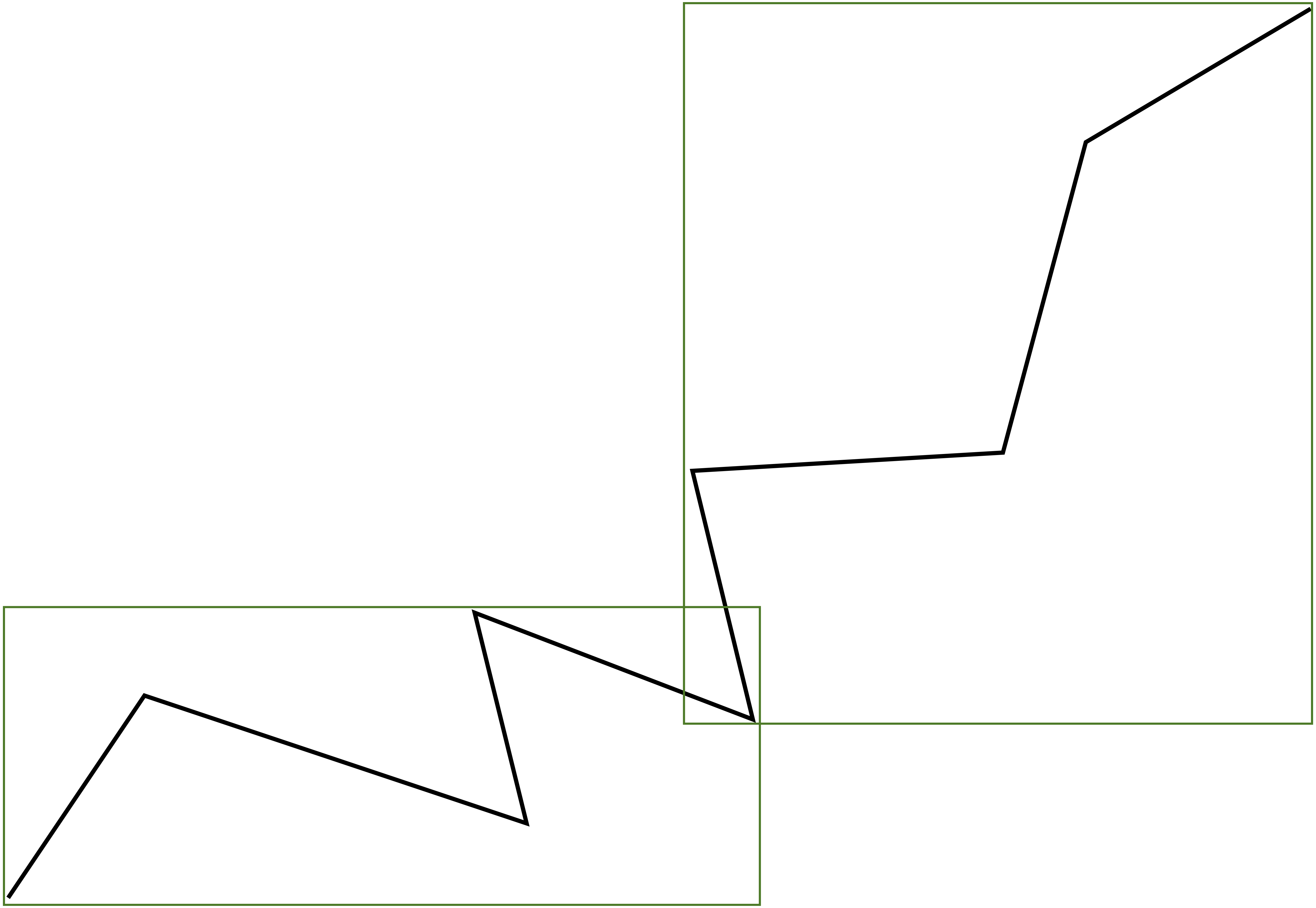


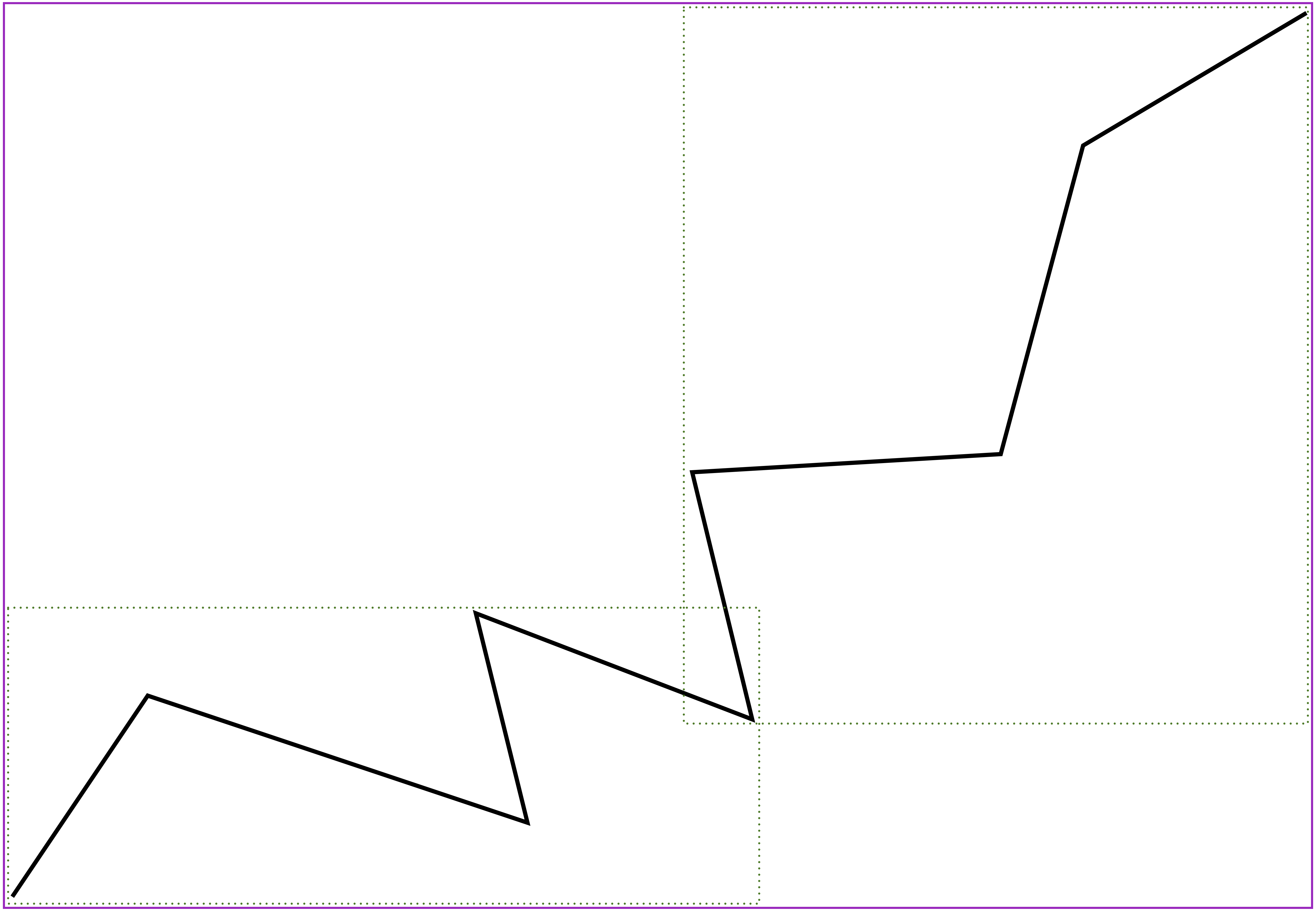


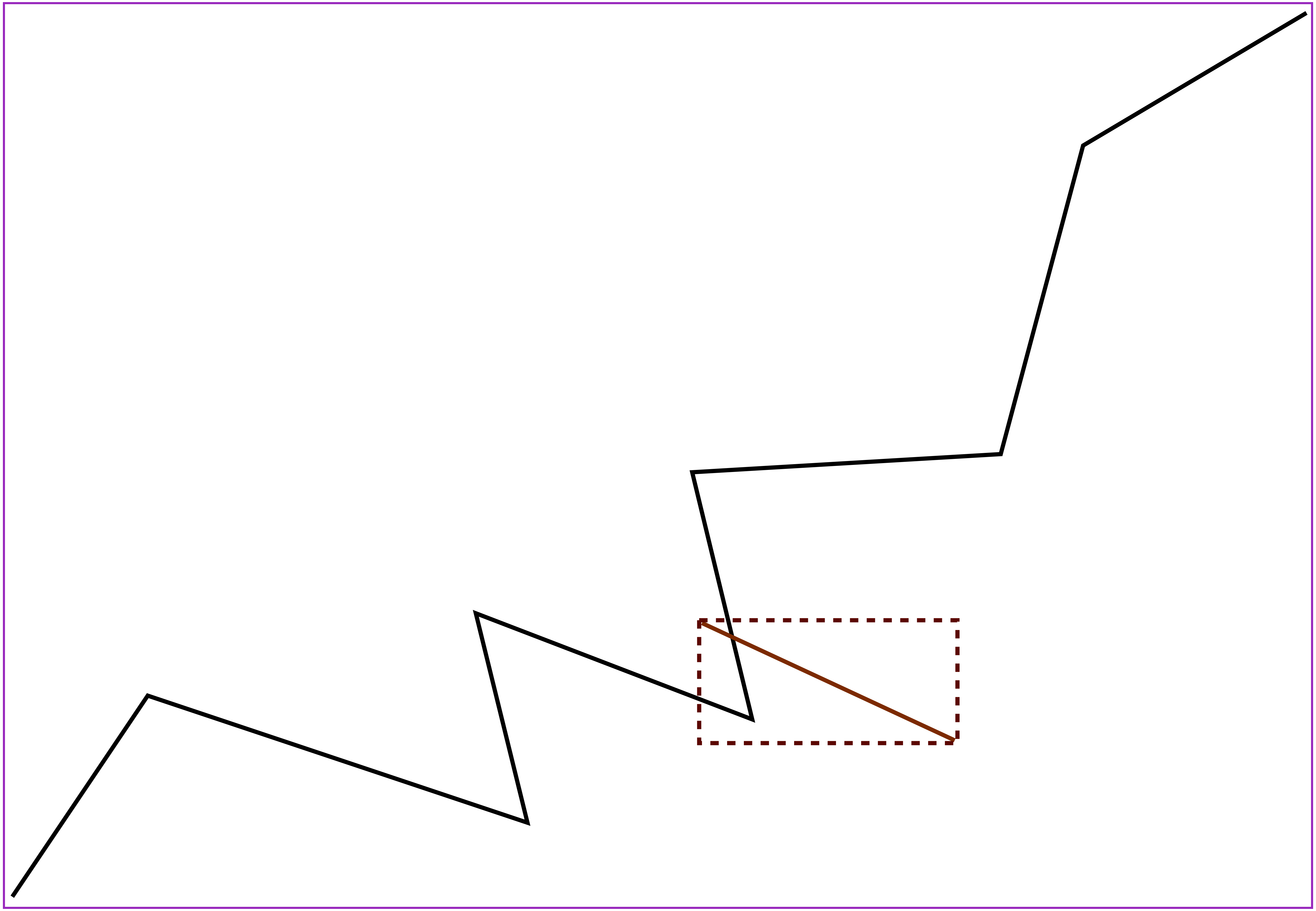


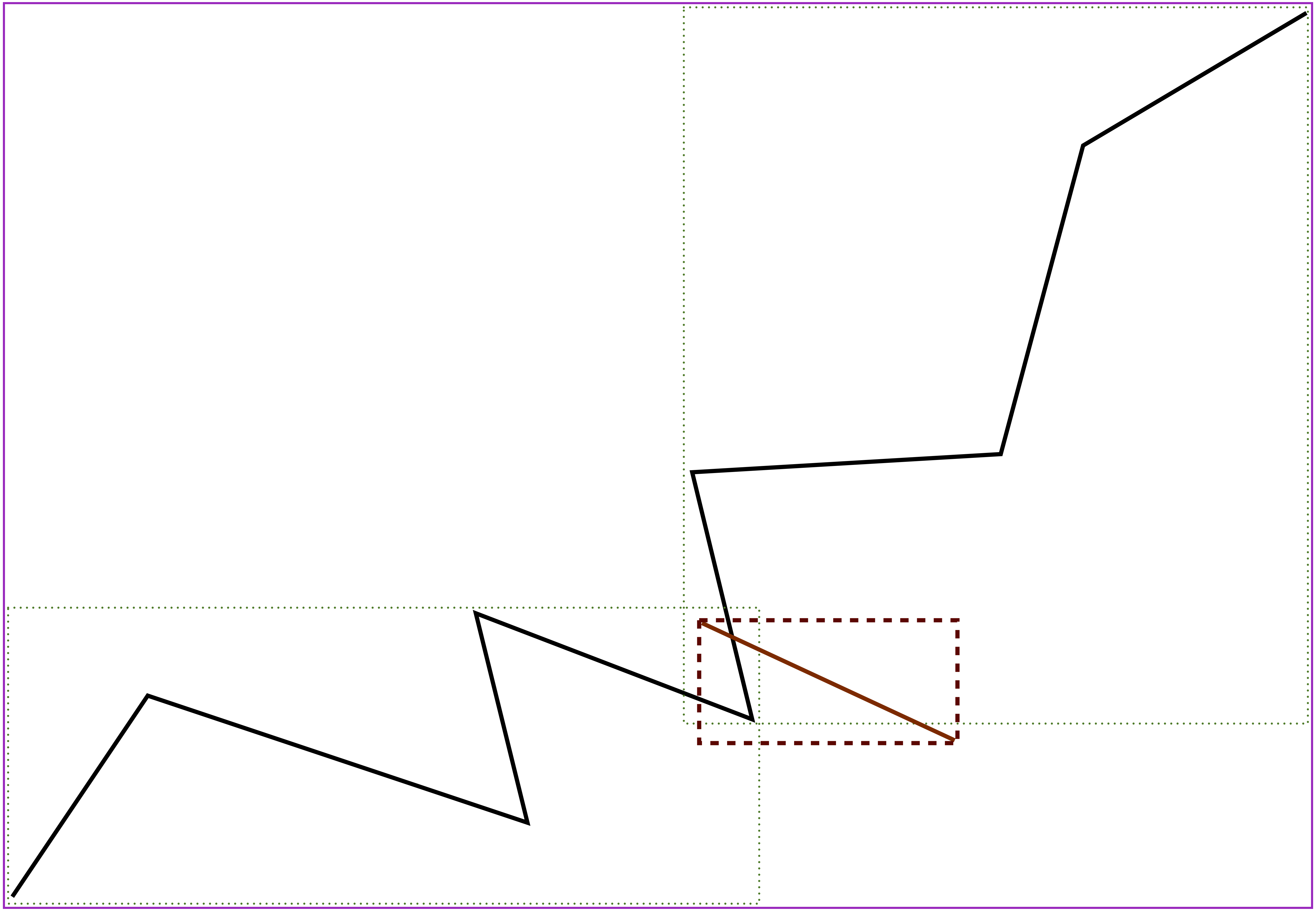


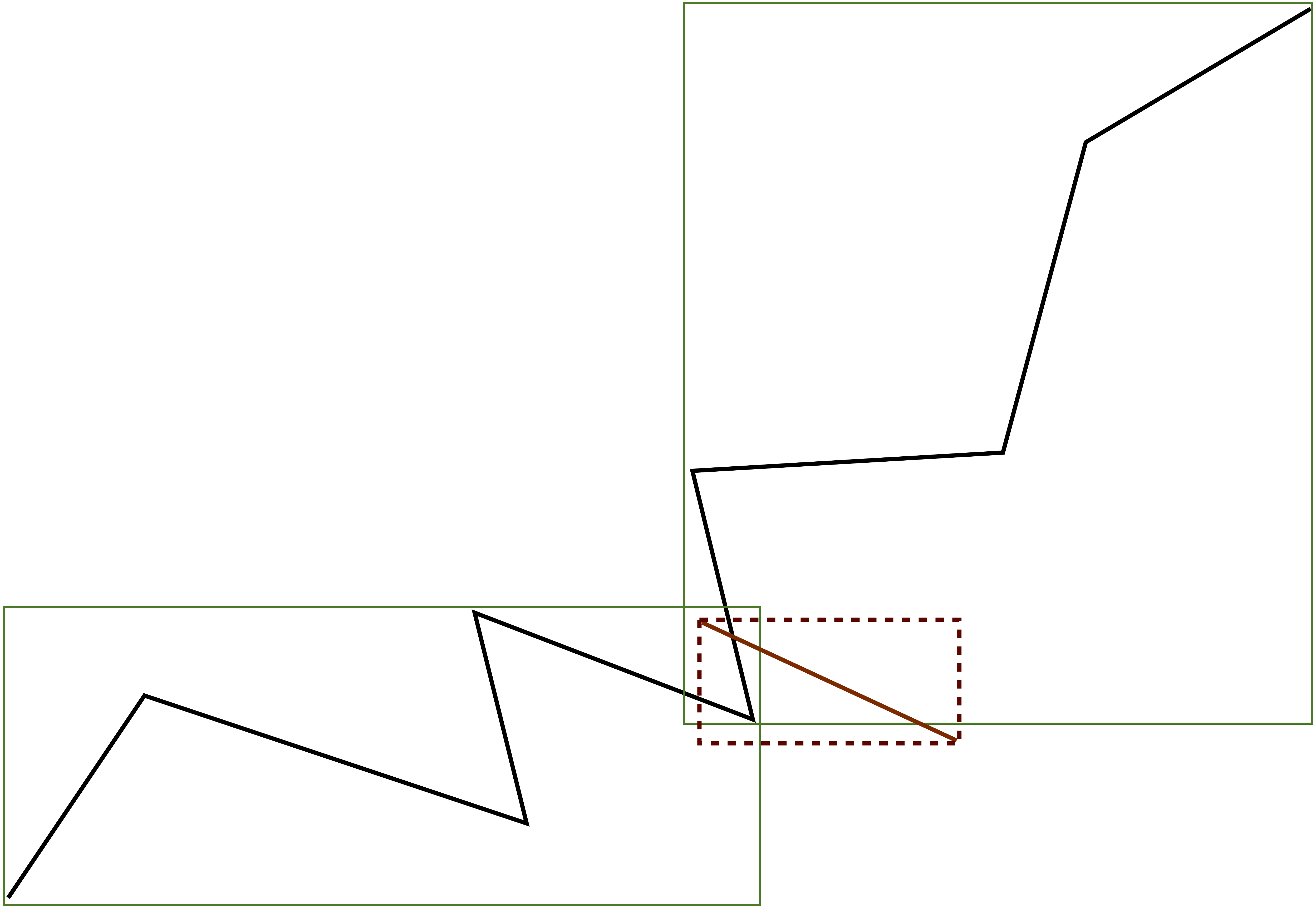


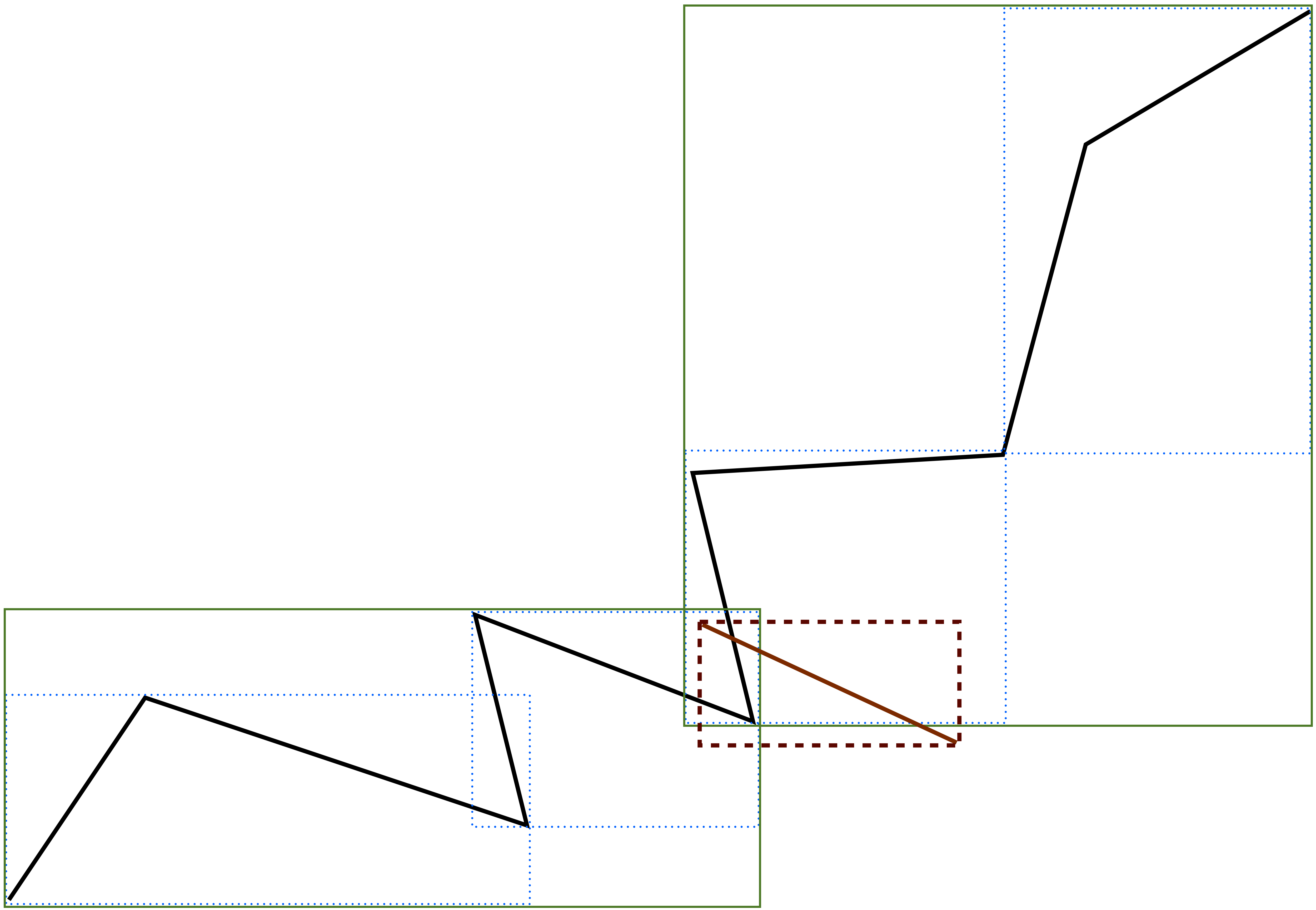


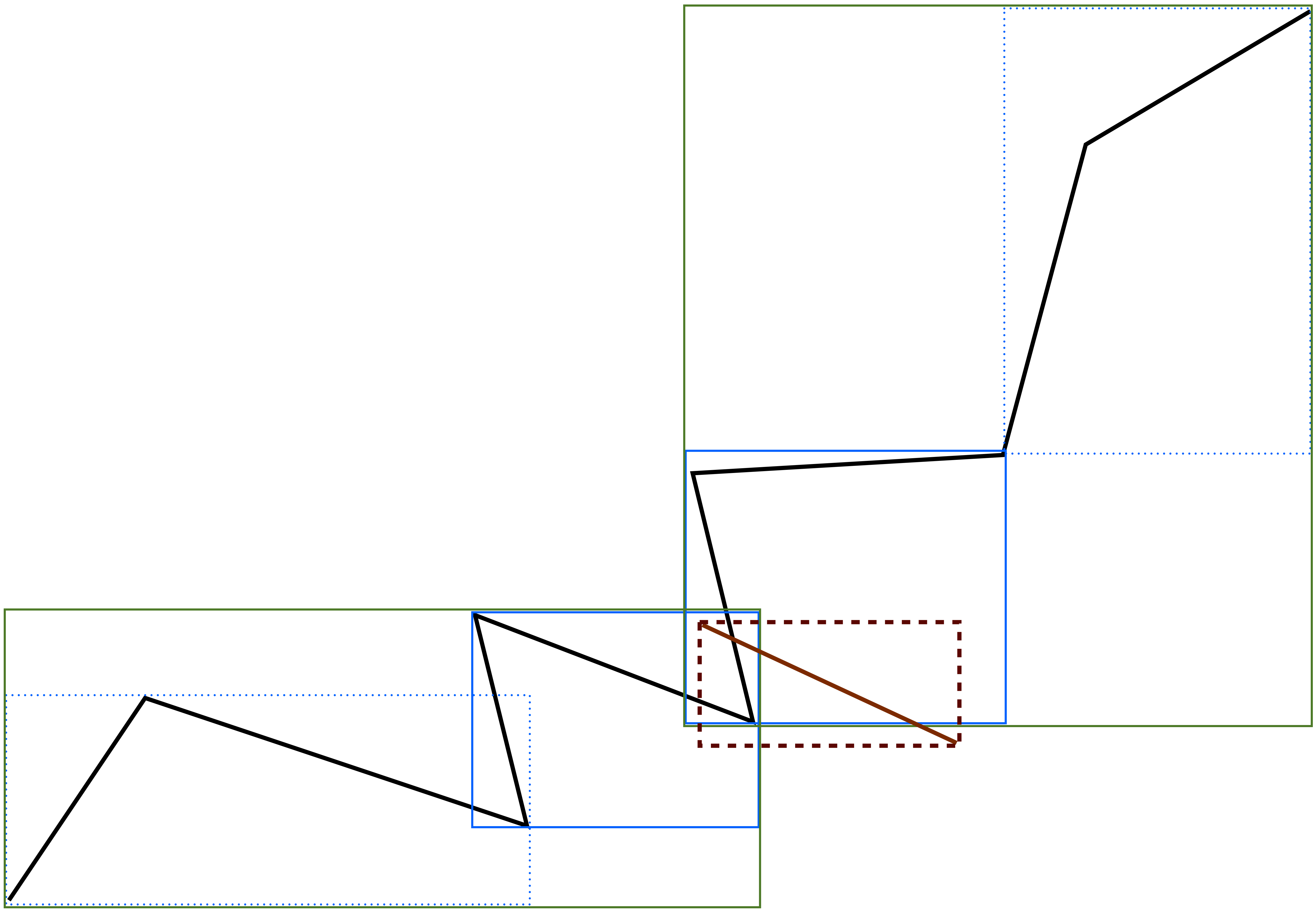


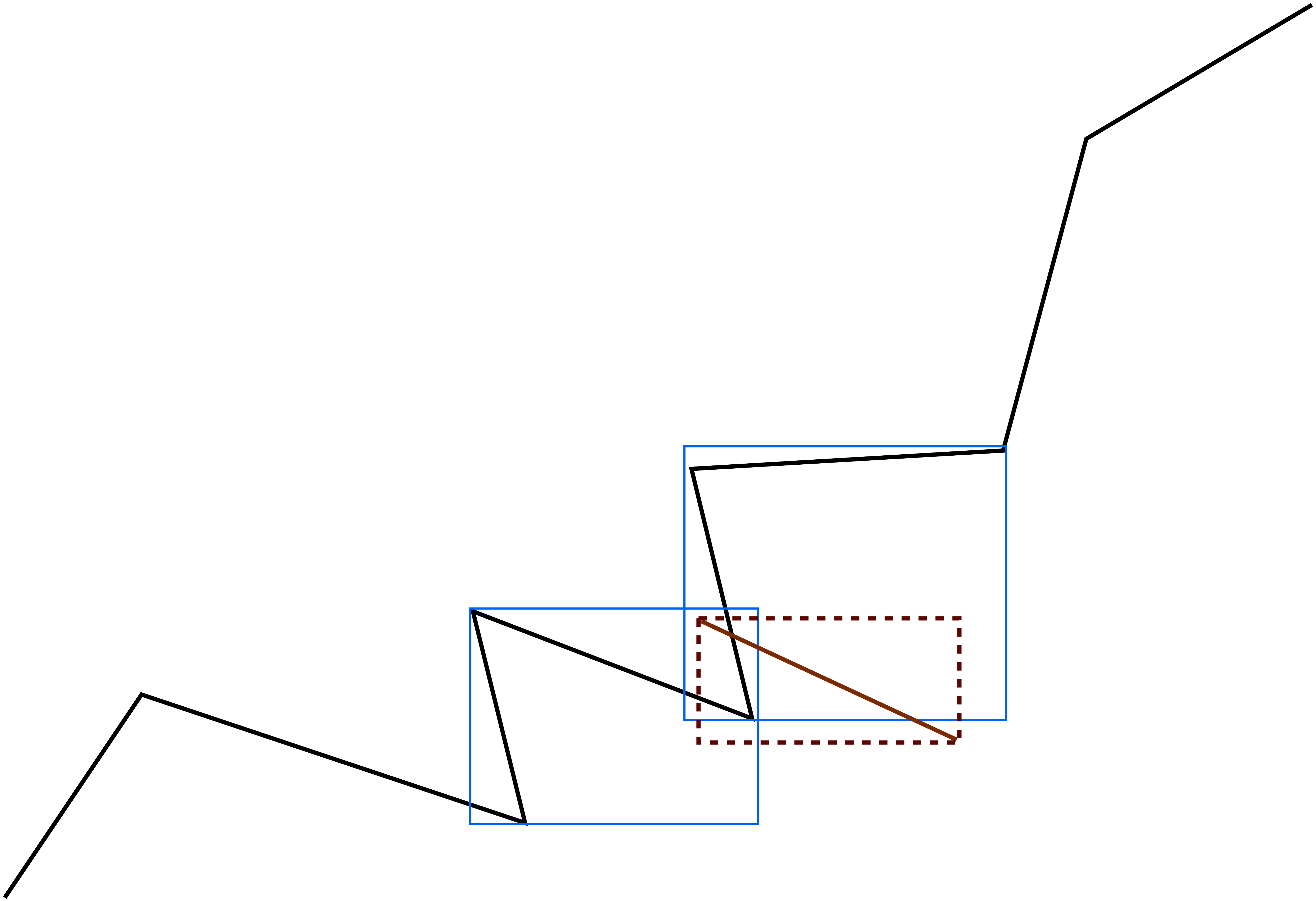


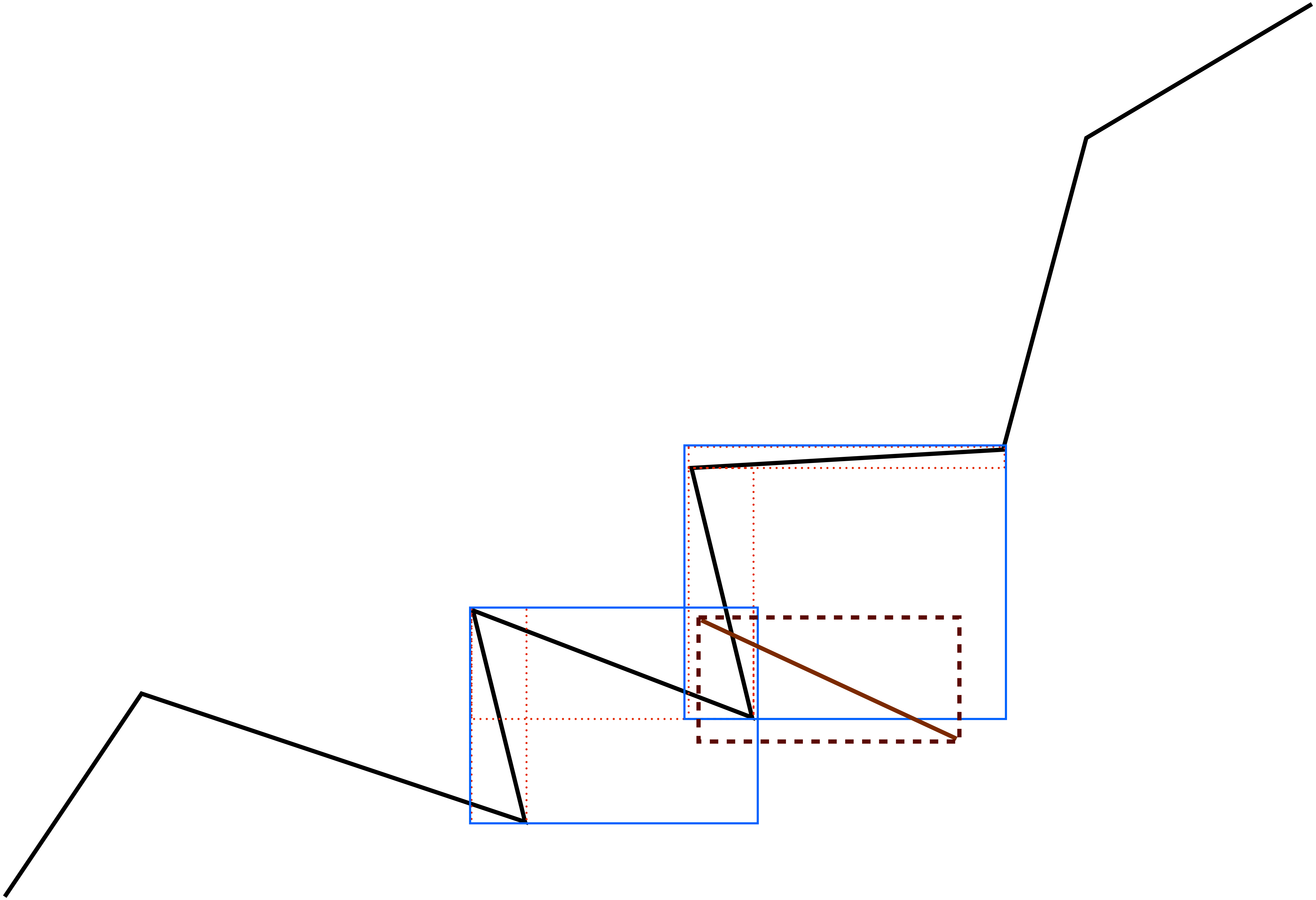


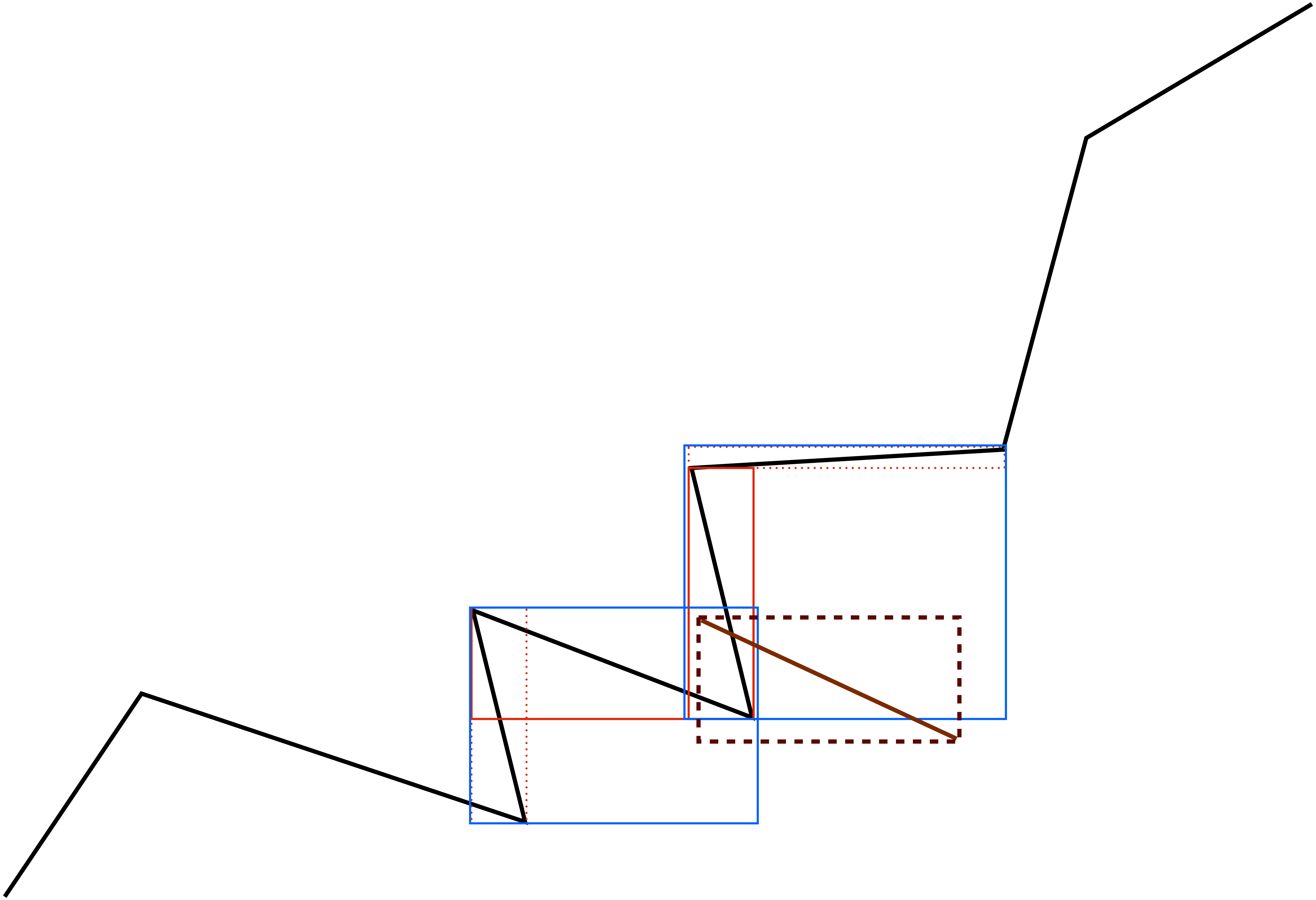


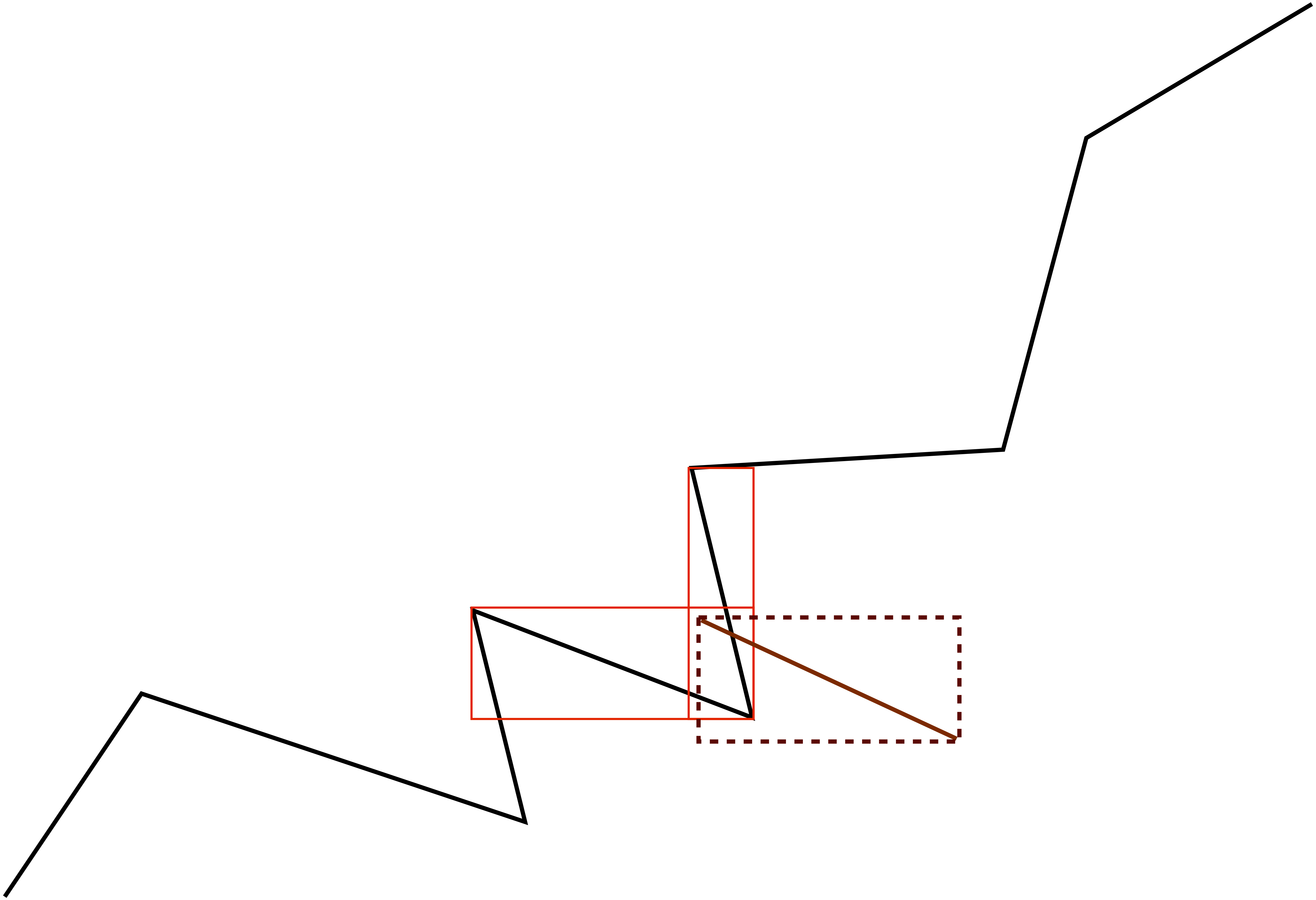


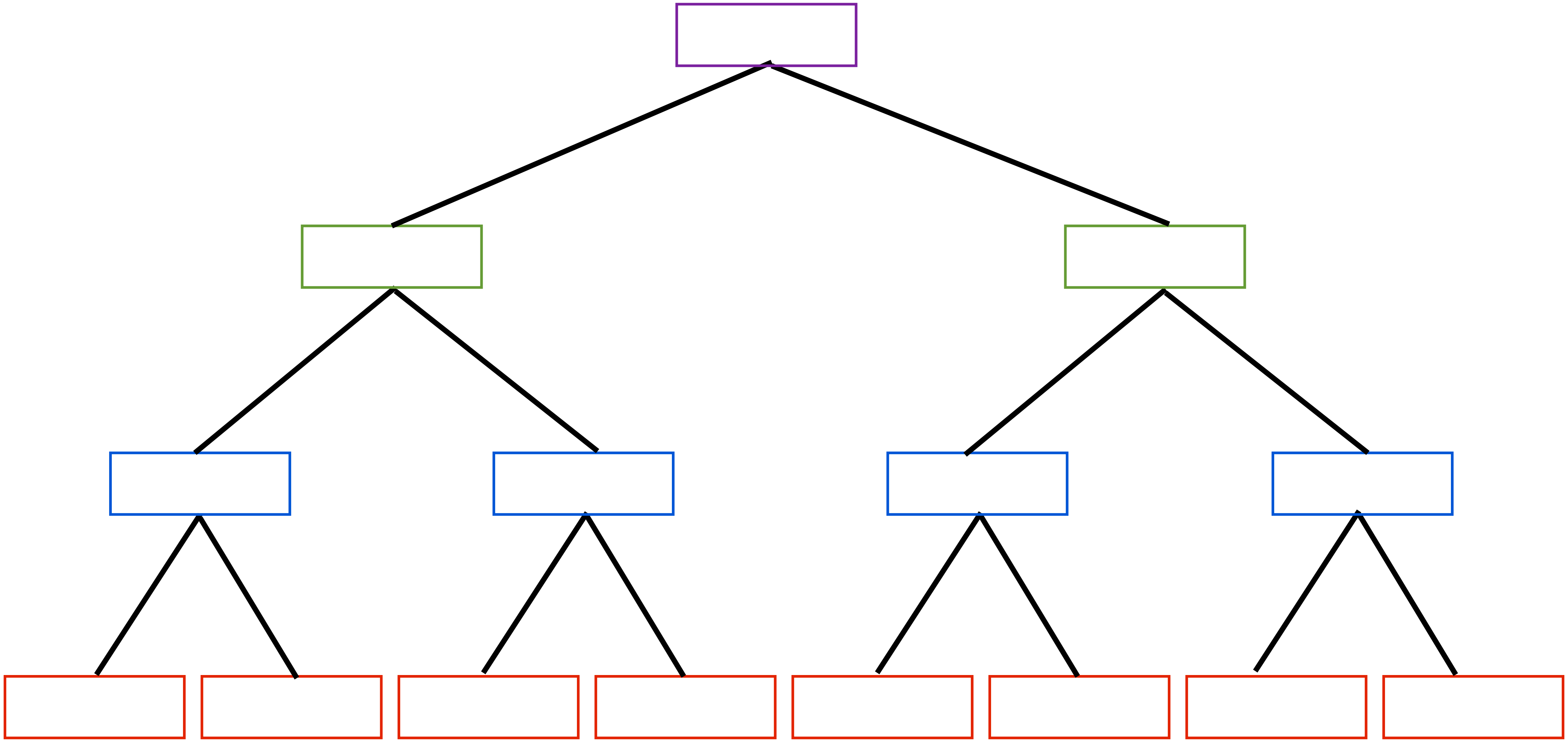


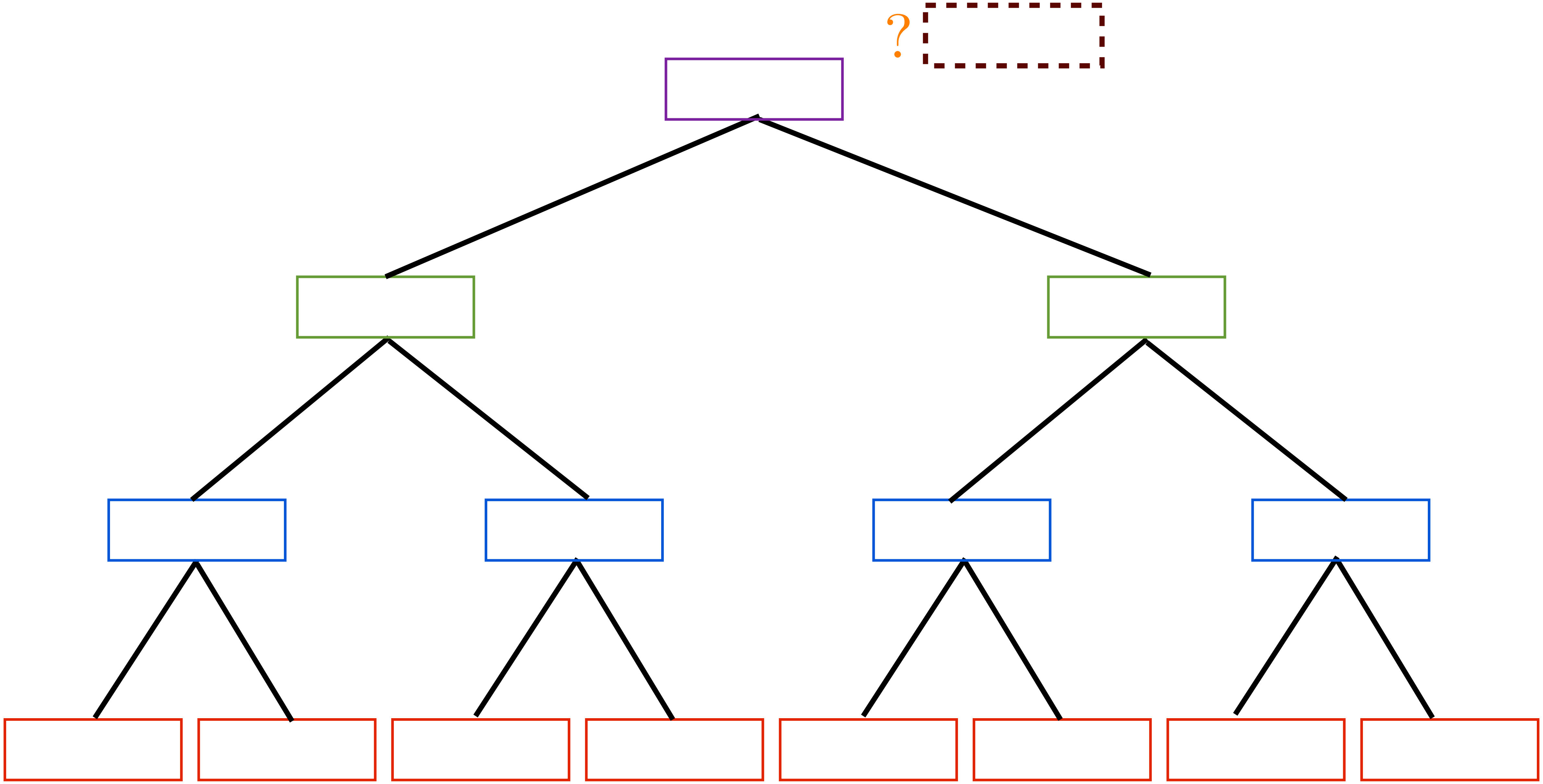


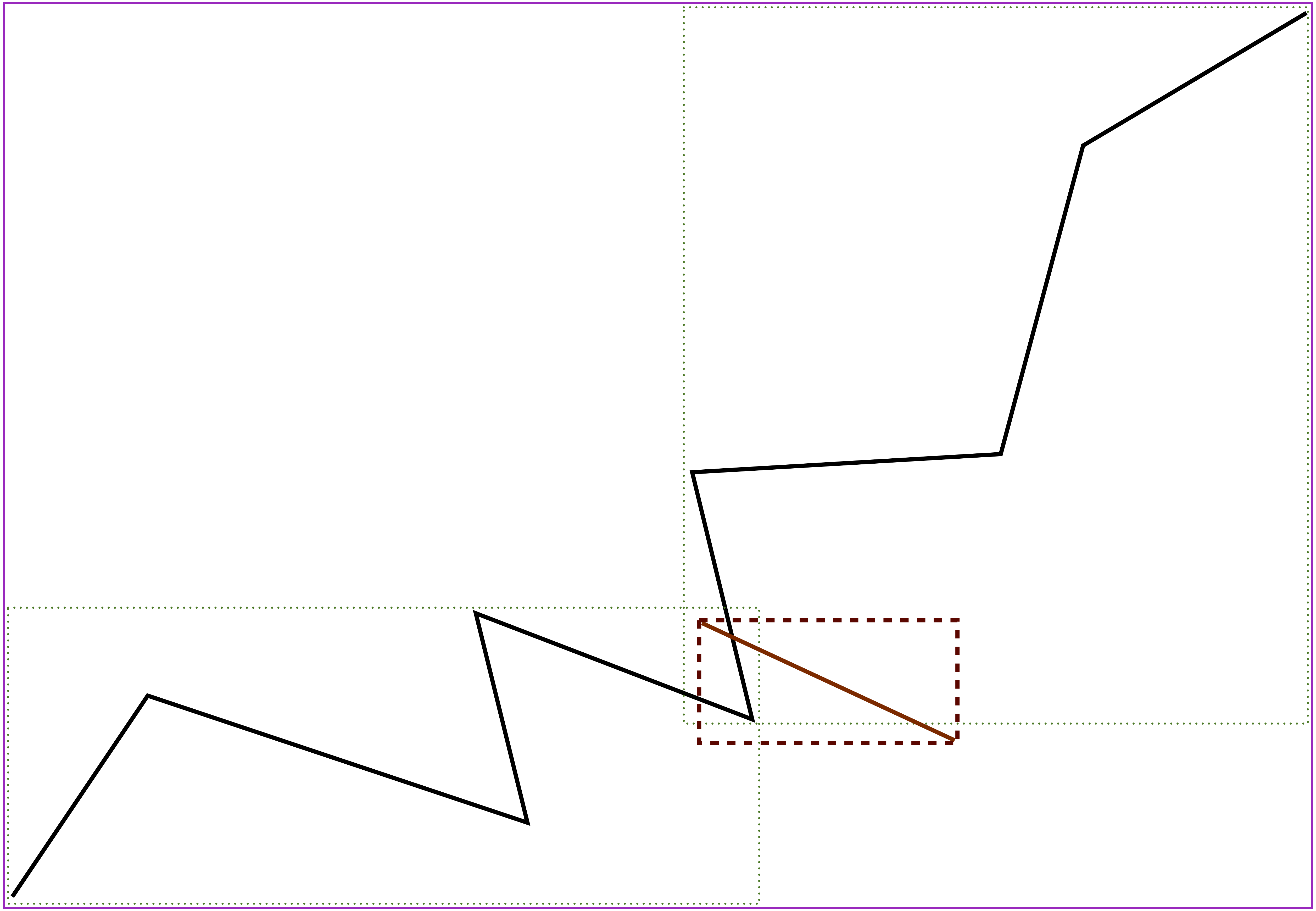


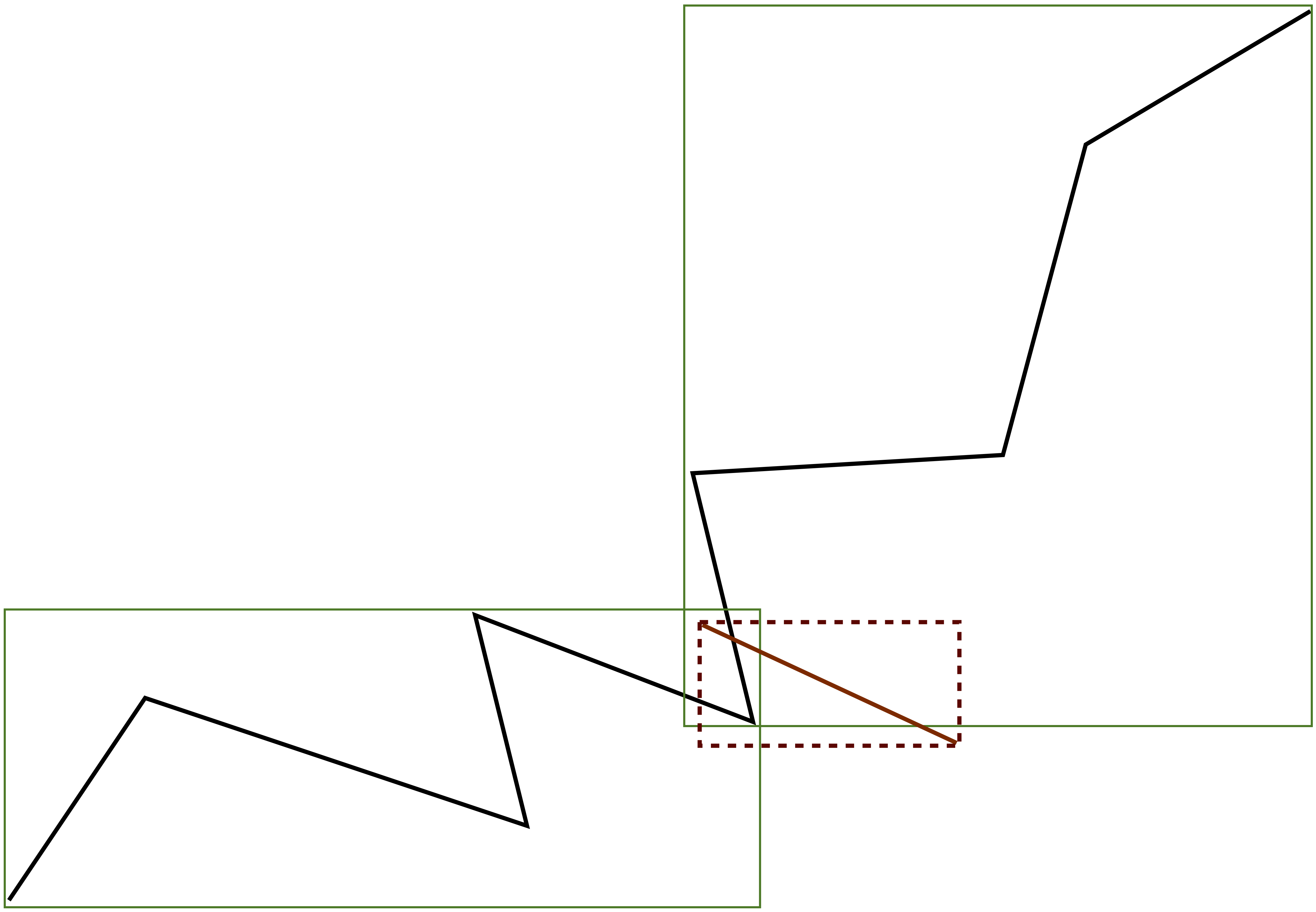


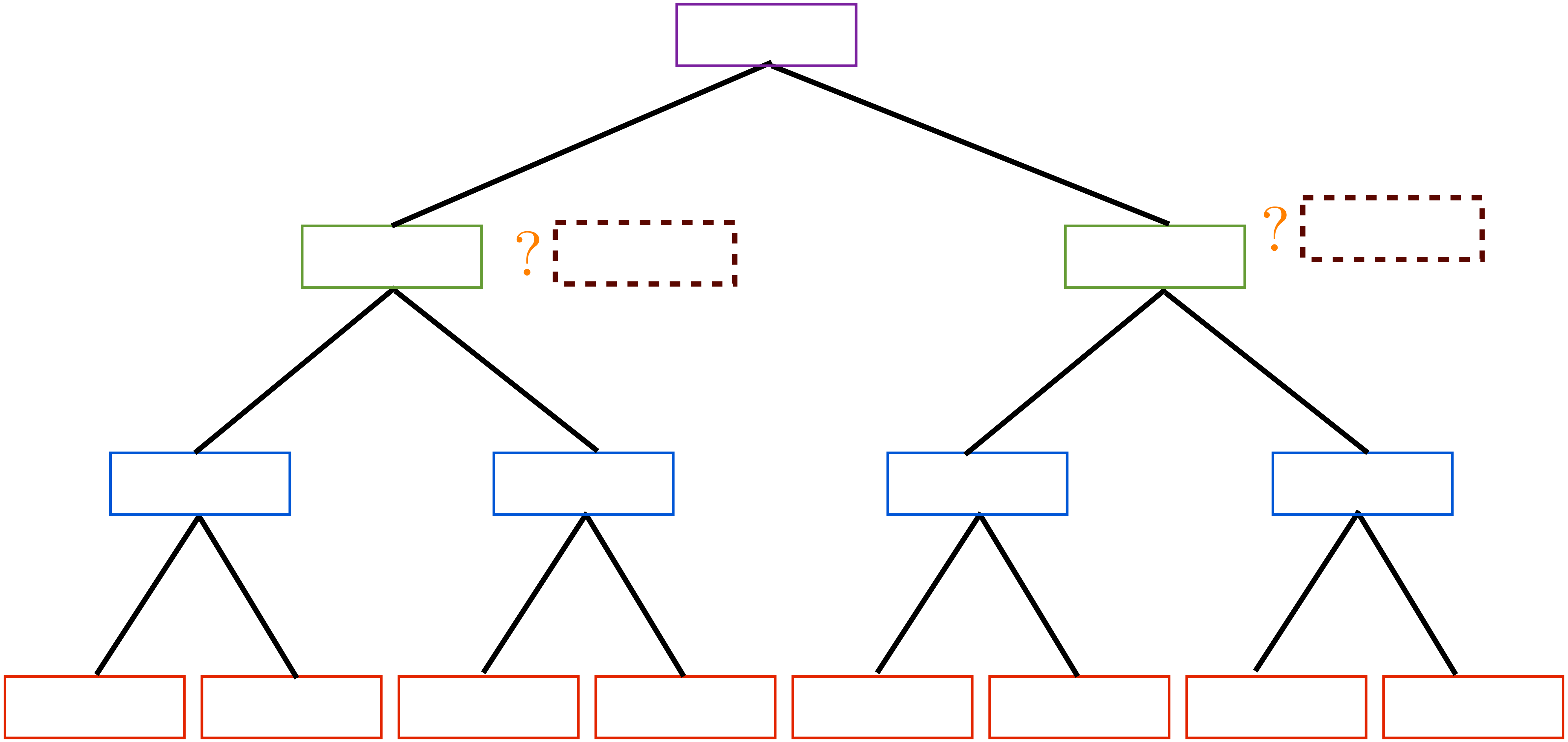


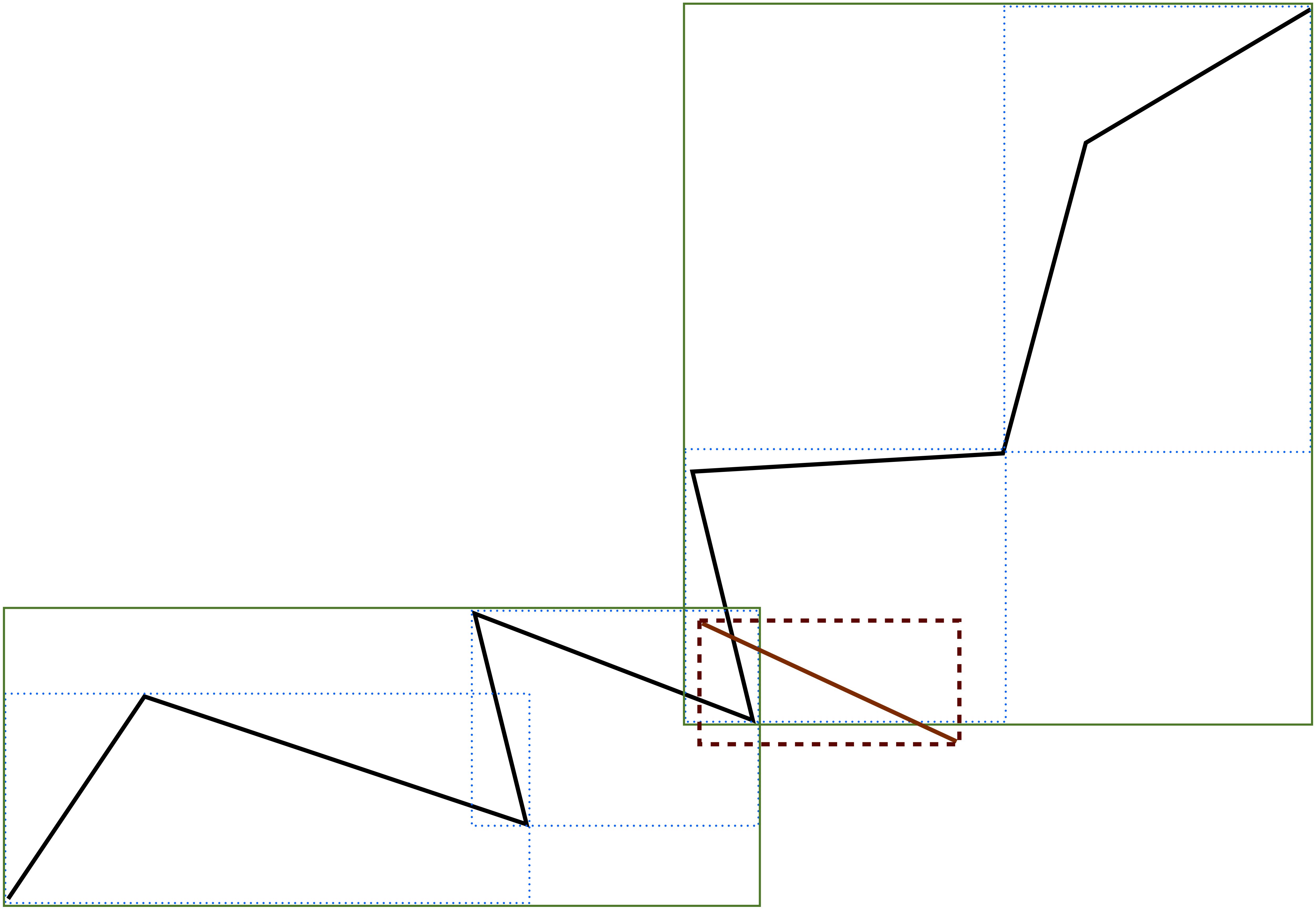


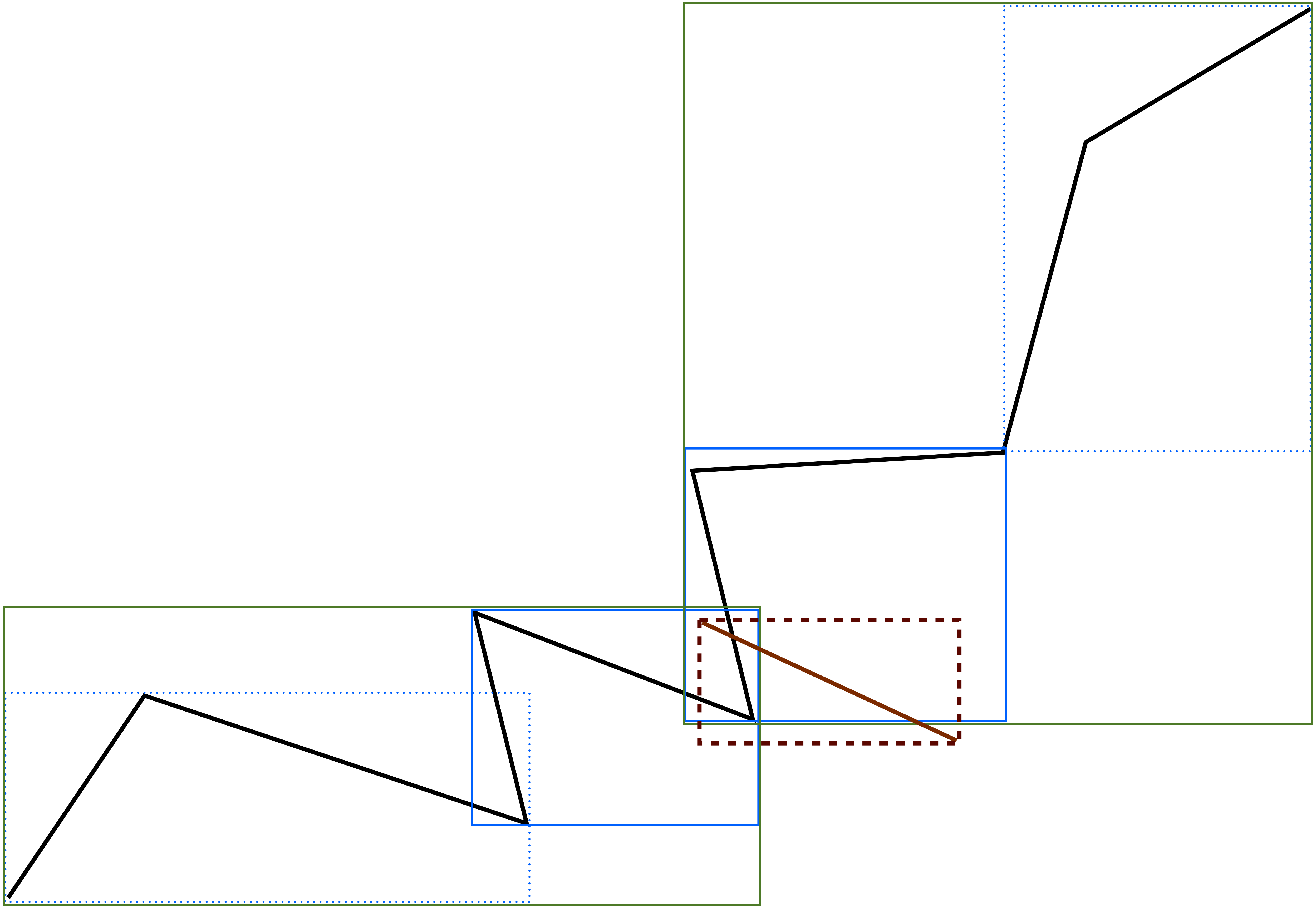


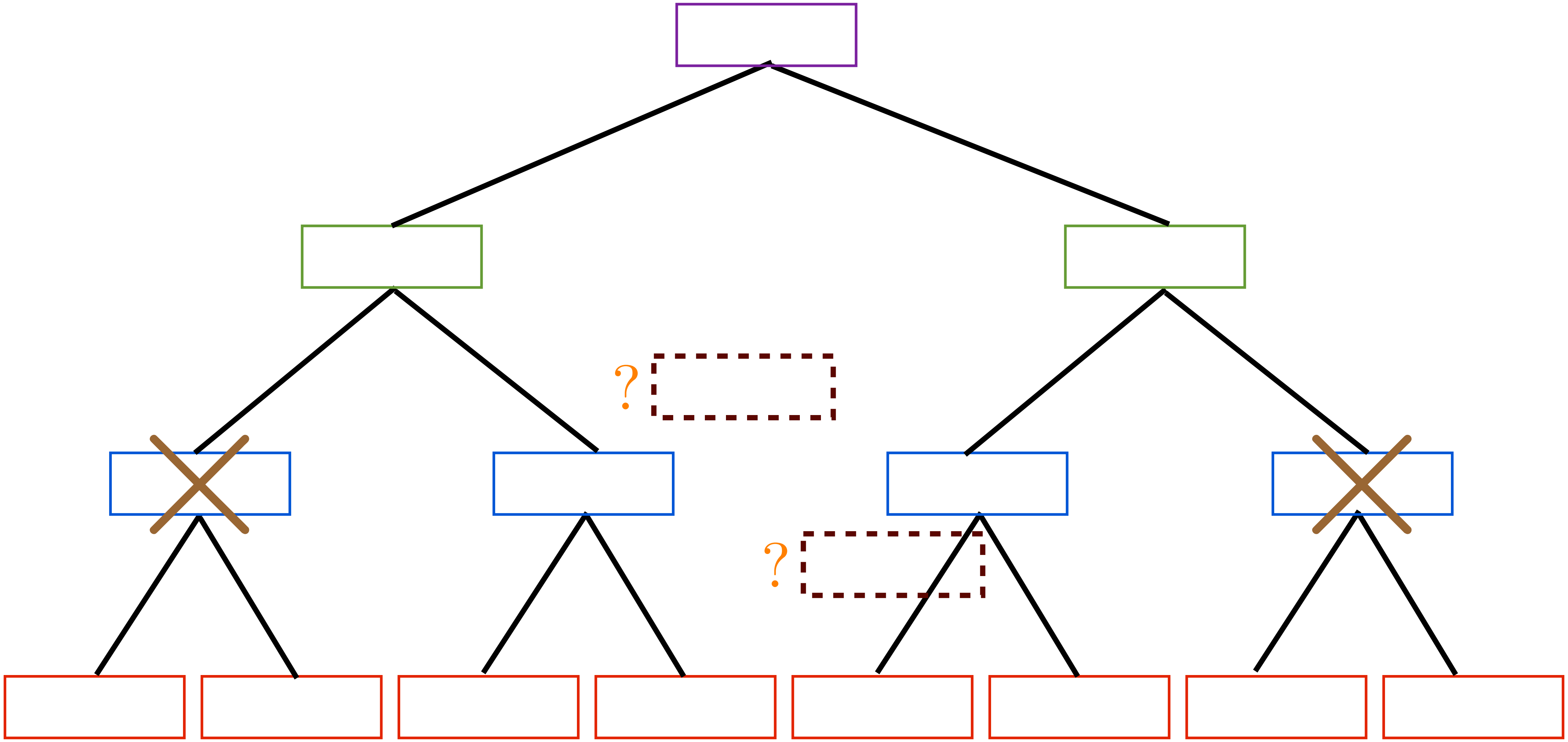


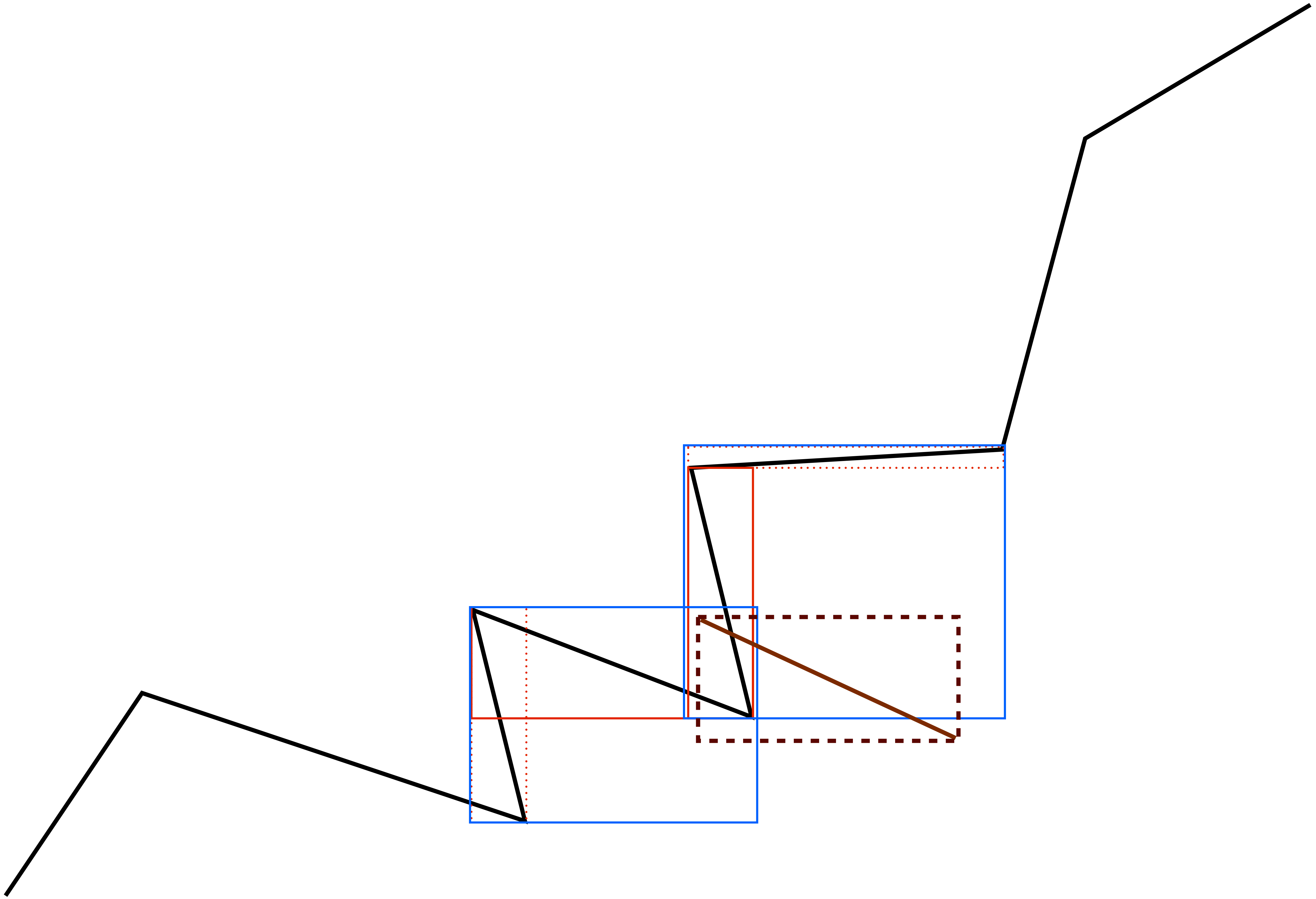


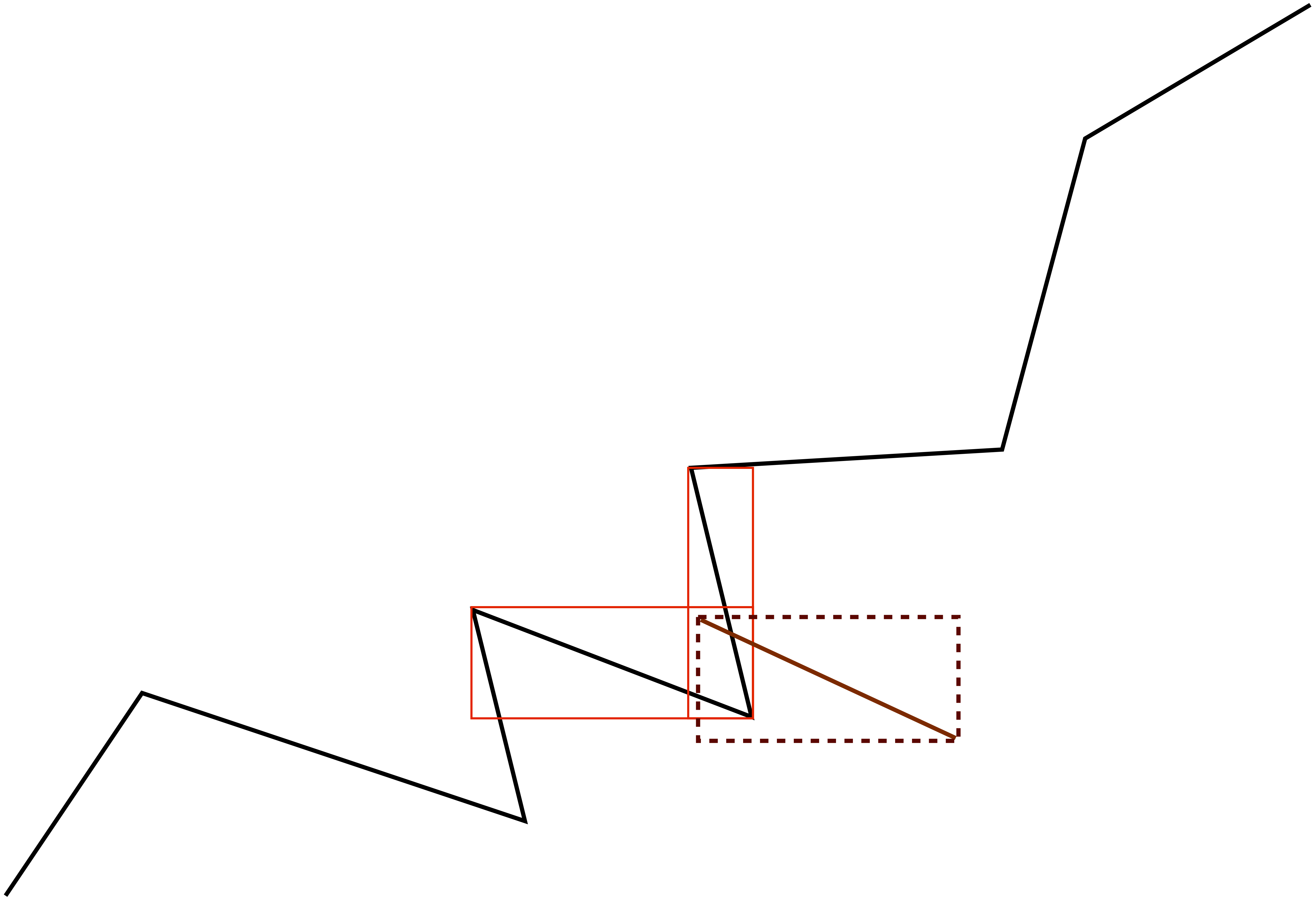


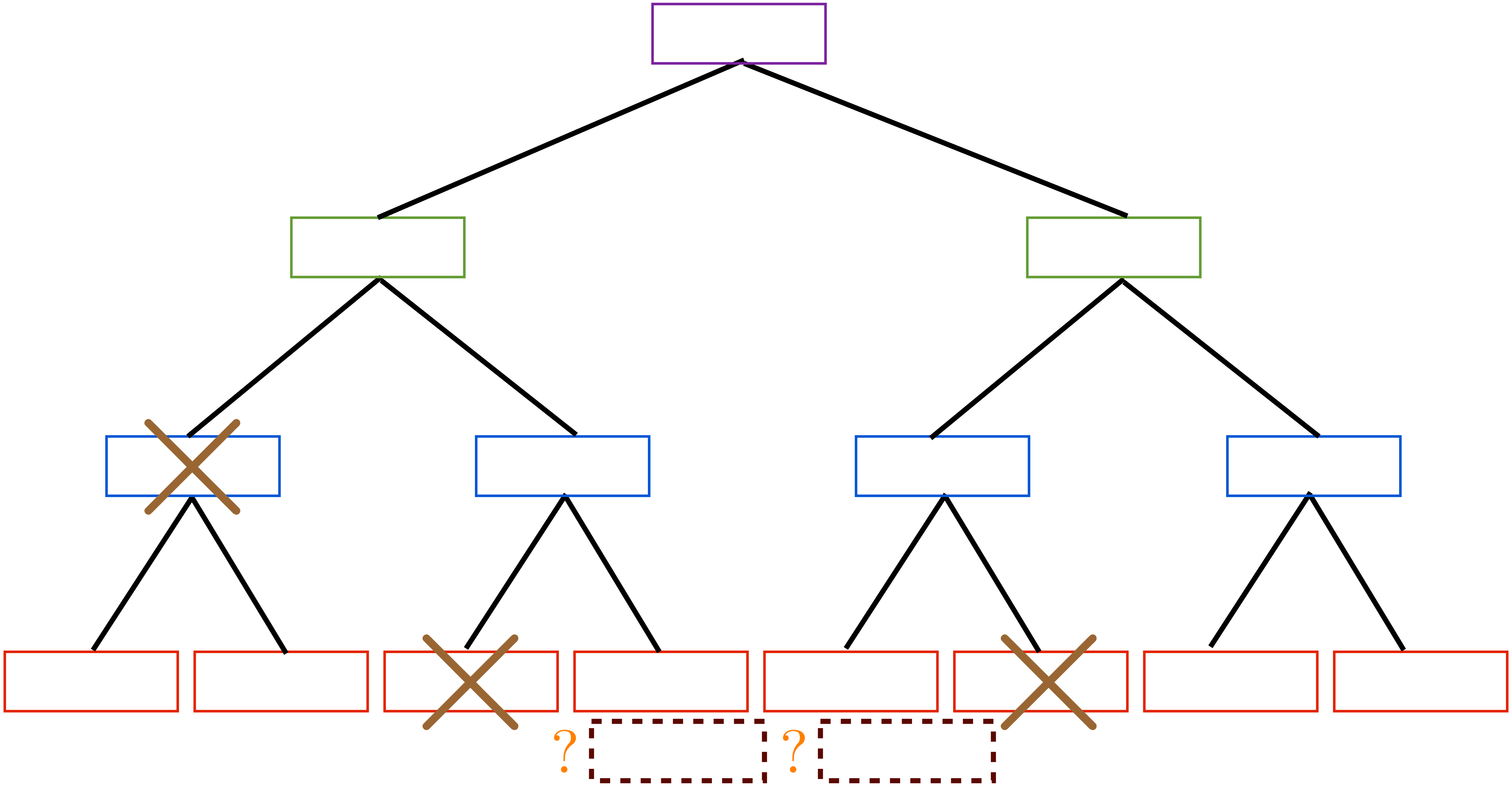












Collision detection (for simulated objects)

- Popular approach : Using axis-aligned bounding box (AABB) queries to accelerate collision detection
  - Prunes away most of the “*faraway*” collisions
  - Cost to check one primitive, against a box B-tree hierarchy with k leaves : O(logk) in the best case
    - Cost will increase if the box hierarchy is not optimally constructed (i.e. if we chose to merge faraway boxes)
    - Quality of hierarchy will degrade as object moves : May choose to re-build the hierarchy from scratch every few time steps
    - KD-Tree or Quad-/Oct-trees can be used to generate box hierarchies