CSE-170 Computer Graphics

Lecture 6

Projections, Picking, and Polygon Triangulation

Dr. Renato Farias
rfarias2@ucmerced.edu

Quick Review

Dot Product

The dot product between a and b = $\mathbf{a} \cdot \mathbf{b} = \mathbf{a}^T \mathbf{b}$

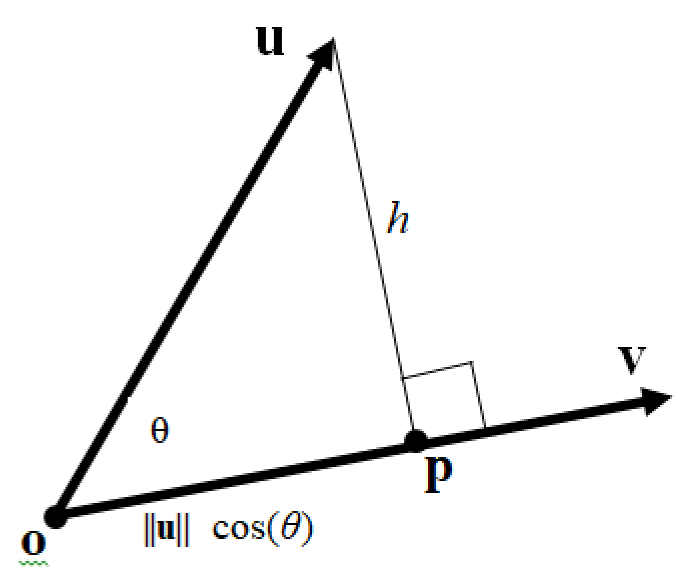
Therefore, the square of the norm has a dot product form: $\|\mathbf{v}\|^2 = \mathbf{v}^T \mathbf{v}$

If $\mathbf{u}$ and $\mathbf{v}$ are perpendicular $\Rightarrow \mathbf{u} \cdot \mathbf{v} = 0$
Acute angle: $\cos(\theta) > 0 \Rightarrow \mathbf{u} \cdot \mathbf{v} > 0$
Obtuse angle: $\cos(\theta) < 0 \Rightarrow \mathbf{u} \cdot \mathbf{v} < 0$

Dot product and the angle between two vectors:

$$\cos(\theta) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\|\|\mathbf{v}\|}$$

$$h = \|\mathbf{u}\|\sin(\theta), \quad \mathbf{p} = (\|\mathbf{u}\|\cos(\theta))\frac{\mathbf{v}}{\|\mathbf{v}\|}$$



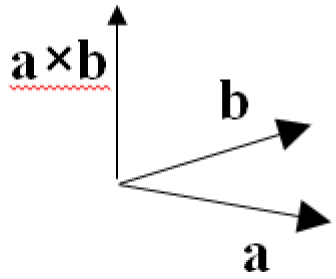
- ⇨ $\mathbf{p}$ is the projection of $\mathbf{u}$ on vector $\mathbf{v}$.
- ⇨ If $\mathbf{u}$ and $\mathbf{v}$ are unit vectors, the length of the projection is simply the dot product: $\|\mathbf{o} - \mathbf{p}\| = \mathbf{u} \cdot \mathbf{v}$ (Note that the norm of $\mathbf{o}$-$\mathbf{p}$ gives the distance between $\mathbf{o}$ and $\mathbf{p}$.)

Quick Review

Cross product

The cross product between $\mathbf{a}$ and $\mathbf{b}$ is

$$\mathbf{v} = \mathbf{a} \times \mathbf{b} = \begin{vmatrix} \hat{\mathbf{i}} & \hat{\mathbf{j}} & \hat{\mathbf{k}} \\ a_x & a_y & a_z \\ b_x & b_y & b_z \end{vmatrix} = \begin{pmatrix} a_y b_z - a_z b_y \\ a_z b_x - a_x b_z \\ a_x b_y - a_y b_x \end{pmatrix}$$



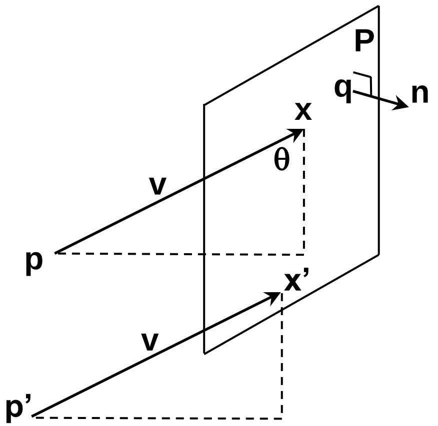
$\boxed{\mathbf{v} \text{ is perpendicular to } \mathbf{a} \text{ and } \mathbf{b},}$ and has length $\|\mathbf{v}\| = \|\mathbf{a} \times \mathbf{b}\| = \|\mathbf{a}\|\|\mathbf{b}\|\sin(\theta)$, theta being the angle between $\mathbf{a}$ and $\mathbf{b}$. The length is 2 times the area of the triangle with sides $\mathbf{a}$ and $\mathbf{b}$.

The direction of $\mathbf{v}$ follows the right hand rule, assuming we are working on a right-hand coordinate system.

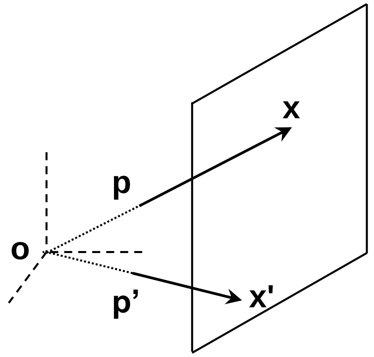
Projections

Projections

- Parallel Projection
  - Same direction, different origins
  - Point **p** is projected by direction **v** in a plane (**q**,**n**)
  - Projection can be of two types:
    - Orthographic (θ=90°), or Oblique

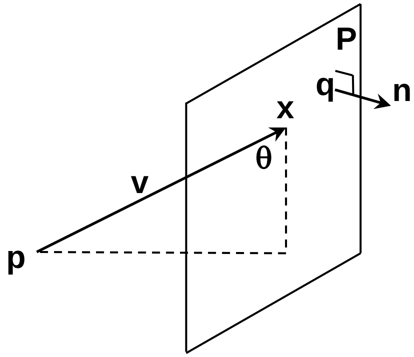


- Perspective Projection
  - Same origin, different directions
  - Assumes the camera “eye” is at the origin **o**
  - For each point **p**, the direction of projection is **p** (line from the origin to **p**)



Parallel Projection

- Parallel Projection
  - Point **p** is projected by direction **v** in a plane (**q**,**n**)



From the plane equation: $(\mathbf{x}-\mathbf{q})\cdot\mathbf{n}=0$ (1)

From the ray definition: $\exists t : \mathbf{x}=\mathbf{p}+t\mathbf{v}$ (2)

=> replacing (2) in (1):

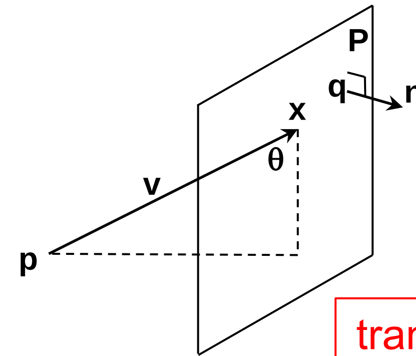
$$((\mathbf{p}+t\mathbf{v})-\mathbf{q})\cdot\mathbf{n}=0 \Rightarrow t=\frac{(\mathbf{q}-\mathbf{p})\cdot\mathbf{n}}{\mathbf{v}\cdot\mathbf{n}}$$

$$\Rightarrow \mathbf{x}=\mathbf{p}+\frac{(\mathbf{q}-\mathbf{p})\cdot\mathbf{n}}{\mathbf{v}\cdot\mathbf{n}}\mathbf{v} \qquad (3)$$

Parallel Projection

- Parallel Projection

$$\Rightarrow \mathbf{x} = \mathbf{p} + \frac{(\mathbf{q}-\mathbf{p})\cdot\mathbf{n}}{\mathbf{v}\cdot\mathbf{n}}\mathbf{v} \qquad (3)$$



Rewriting as an affine map:

translation

$$\mathbf{x} = \mathbf{p} + \frac{\mathbf{q}\cdot\mathbf{n}}{\mathbf{v}\cdot\mathbf{n}}\mathbf{v} - \frac{\mathbf{p}\cdot\mathbf{n}}{\mathbf{v}\cdot\mathbf{n}}\mathbf{v} = \mathbf{p} - \frac{\mathbf{n}^T\mathbf{p}}{\mathbf{v}\cdot\mathbf{n}}\mathbf{v} + \frac{\mathbf{q}\cdot\mathbf{n}}{\mathbf{v}\cdot\mathbf{n}}\mathbf{v} = \mathbf{p} - \frac{\mathbf{v}\mathbf{n}^T}{\mathbf{v}\cdot\mathbf{n}}\mathbf{p} + \frac{\mathbf{q}\cdot\mathbf{n}}{\mathbf{v}\cdot\mathbf{n}}\mathbf{v} = \left(\mathbf{I} - \frac{\mathbf{v}\mathbf{n}^T}{\mathbf{v}\cdot\mathbf{n}}\right)\mathbf{p} + \frac{\mathbf{q}\cdot\mathbf{n}}{\mathbf{v}\cdot\mathbf{n}}\mathbf{v}$$

Will be identity after division

translation

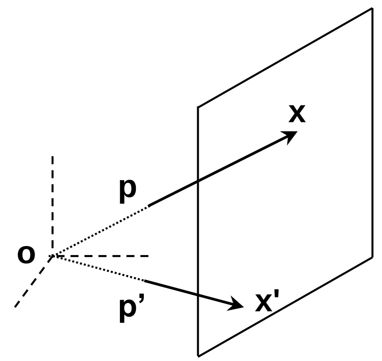
division

$$\mathbf{x} = \begin{pmatrix} \begin{pmatrix} \mathbf{v}\cdot\mathbf{n} & 0 & 0 \\ 0 & \mathbf{v}\cdot\mathbf{n} & 0 \\ 0 & 0 & \mathbf{v}\cdot\mathbf{n} \end{pmatrix} - \left(\mathbf{v}\mathbf{n}^T\right) & (\mathbf{q}\cdot\mathbf{n})\mathbf{v} \\ 0 \quad 0 \quad 0 & \mathbf{v}\cdot\mathbf{n} \end{pmatrix} \begin{pmatrix} \mathbf{p} \\ 1 \end{pmatrix}$$

division

Perspective Projection

- Perspective Projection
  - Assumes the camera "eye" is at the origin **o**
  - For each point **p**, the direction of projection is **p** (line from the origin to **p**)
    - Direction **v** from parallel projection becomes **p**

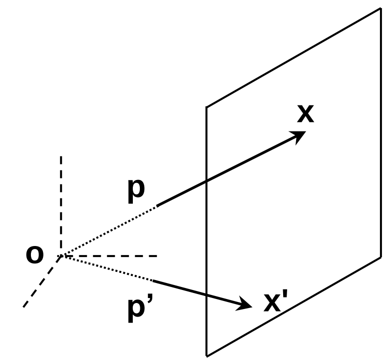


$$\text{From (3): } \mathbf{x} = \mathbf{p} + \frac{(\mathbf{q}-\mathbf{p})\cdot\mathbf{n}}{\mathbf{v}\cdot\mathbf{n}}\mathbf{p} = \mathbf{p} + \frac{\mathbf{q}\cdot\mathbf{n}}{\mathbf{p}\cdot\mathbf{n}}\mathbf{p} - \frac{\mathbf{p}\cdot\mathbf{n}}{\mathbf{p}\cdot\mathbf{n}}\mathbf{p} \Rightarrow \mathbf{x} = \frac{\mathbf{q}\cdot\mathbf{n}}{\mathbf{p}\cdot\mathbf{n}}\mathbf{p}$$

Perspective Projection

- Perspective Projection

$\mathbf{x} = \dfrac{\mathbf{q}\cdot\mathbf{n}}{\mathbf{p}\cdot\mathbf{n}}\mathbf{p}$, writing as a matrix multiplication, we have:



$$\mathbf{x} = \begin{pmatrix} \begin{pmatrix} \mathbf{q}\cdot\mathbf{n} & 0 & 0 \\ 0 & \mathbf{q}\cdot\mathbf{n} & 0 \\ 0 & 0 & \mathbf{q}\cdot\mathbf{n} \end{pmatrix} & 0 \\ 0 \quad 0 \quad 0 & \mathbf{p}\cdot\mathbf{n} \end{pmatrix} \begin{pmatrix} \mathbf{p} \\ 1 \end{pmatrix}$$

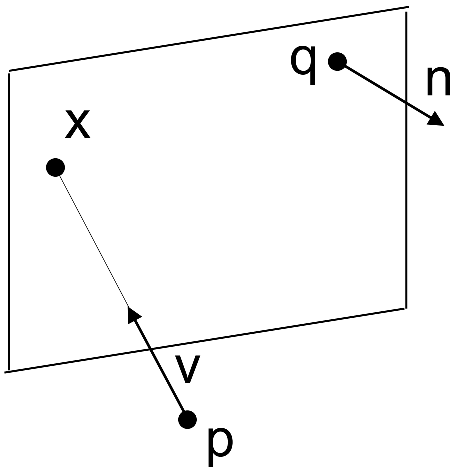
Note that the previous matrix obtained still depends on $\mathbf{p}$. Fortunately, we can further manipulate its elements to achieve the following equivalent result:

$$\mathbf{x} = \begin{pmatrix} \begin{pmatrix} \mathbf{q}\cdot\mathbf{n} & 0 & 0 \\ 0 & \mathbf{q}\cdot\mathbf{n} & 0 \\ 0 & 0 & \mathbf{q}\cdot\mathbf{n} \end{pmatrix} & 0 \\ \mathbf{n}^{T} & 0 \end{pmatrix} \begin{pmatrix} \mathbf{p} \\ 1 \end{pmatrix}$$

Summary

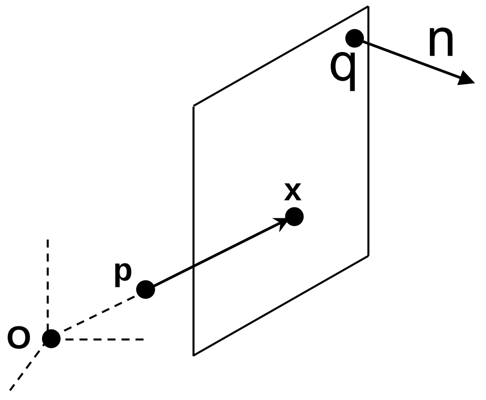
- Parallel projection by direction **v** into a plane(**q**,**n**):

$$\left(\begin{array}{c|c} \begin{pmatrix} \mathbf{v}\cdot\mathbf{n} & 0 & 0 \\ 0 & \mathbf{v}\cdot\mathbf{n} & 0 \\ 0 & 0 & \mathbf{v}\cdot\mathbf{n} \end{pmatrix} - (\mathbf{v}\mathbf{n}^T) & (\mathbf{q}\cdot\mathbf{n})\mathbf{v} \\ \hline 0 \quad 0 \quad 0 & \mathbf{v}\cdot\mathbf{n} \end{array}\right)$$



- Perspective projection into a plane(**q**,**n**):

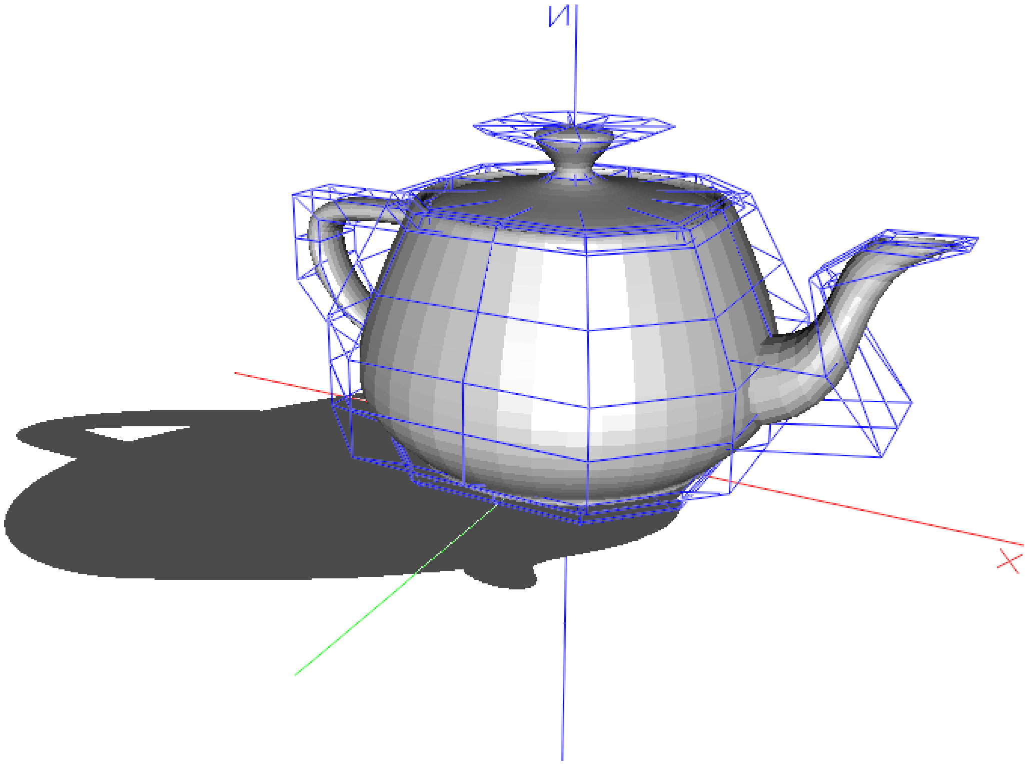
$$\left(\begin{array}{c|c} \mathbf{I}(\mathbf{q}\cdot\mathbf{n}) & 0 \\ \hline \mathbf{n}^{\mathrm{T}} & 0 \end{array}\right)$$



Example

– Shadow generation by projecting triangles to the floor:

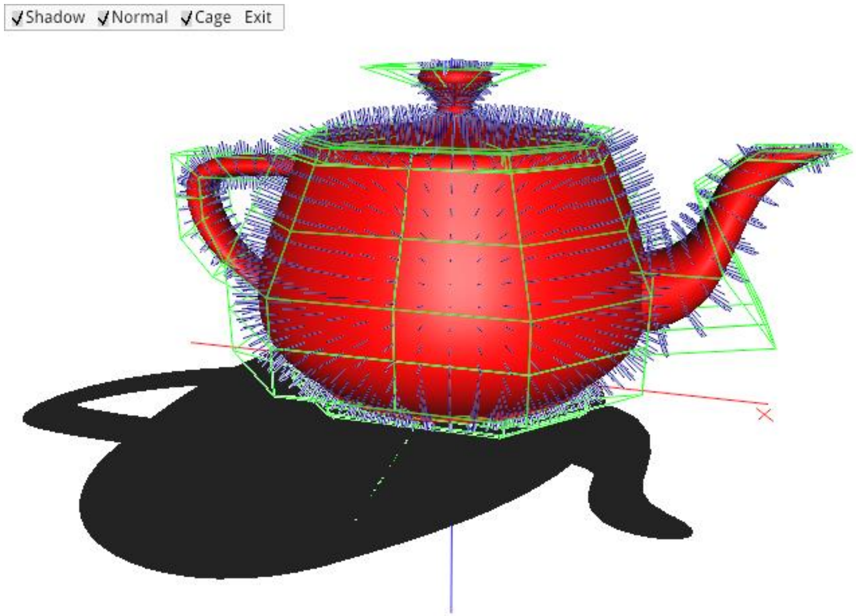
(http://graphics.ucmerced.edu/~mkallmann/courses/eecs267-17f/proj2.html)

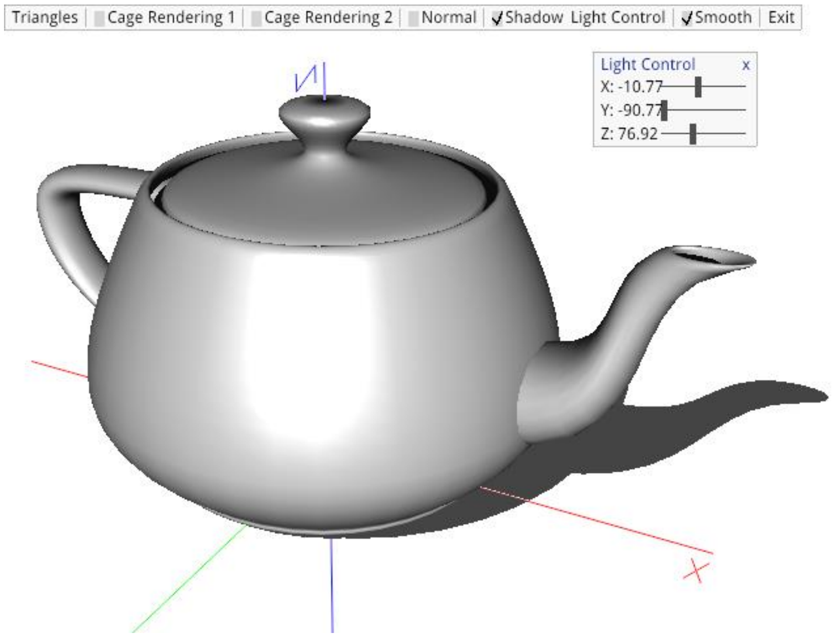


Example

– Examples with smooth shading (upcoming topic):

(http://graphics.ucmerced.edu/~mkallmann/courses/eecs267-17f/proj2.html)





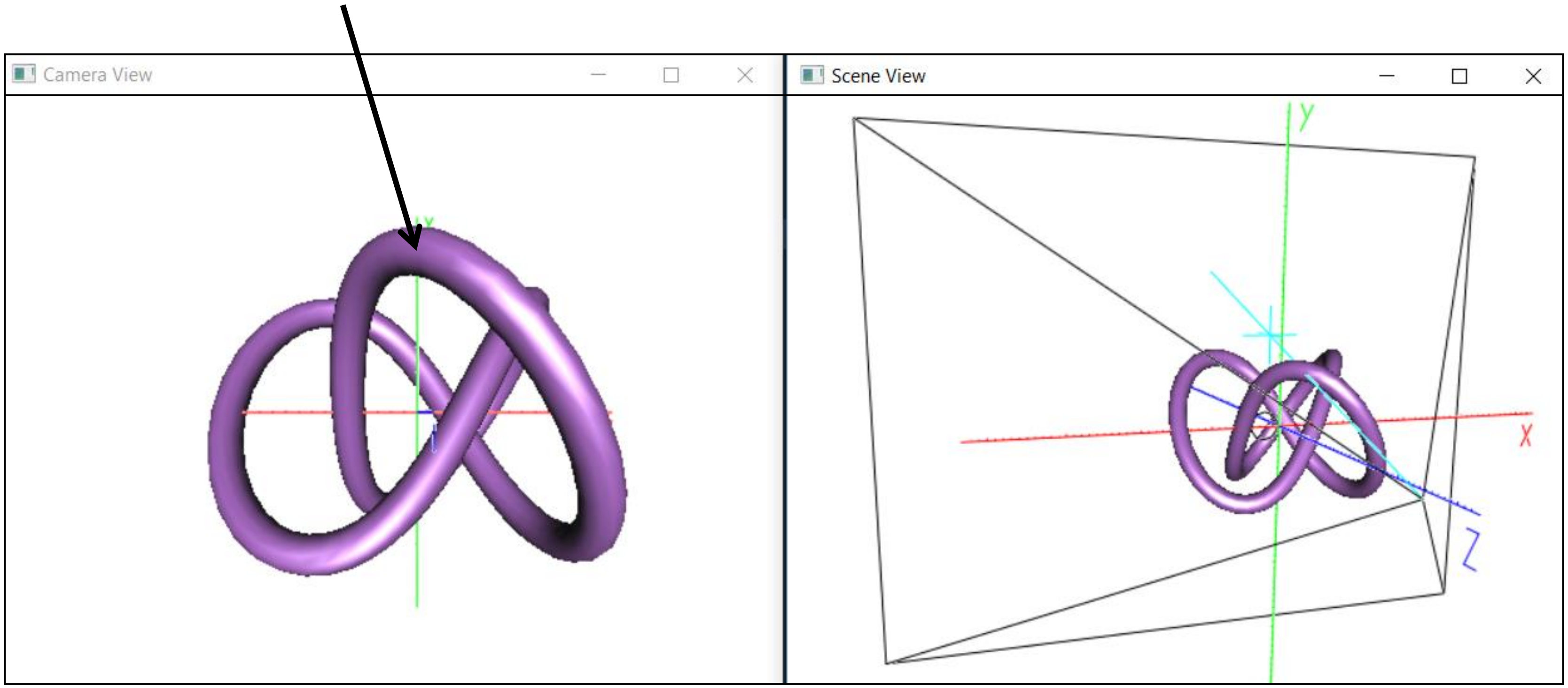
Picking

Picking

- How to select 3D objects in the scene when we do a mouse click?

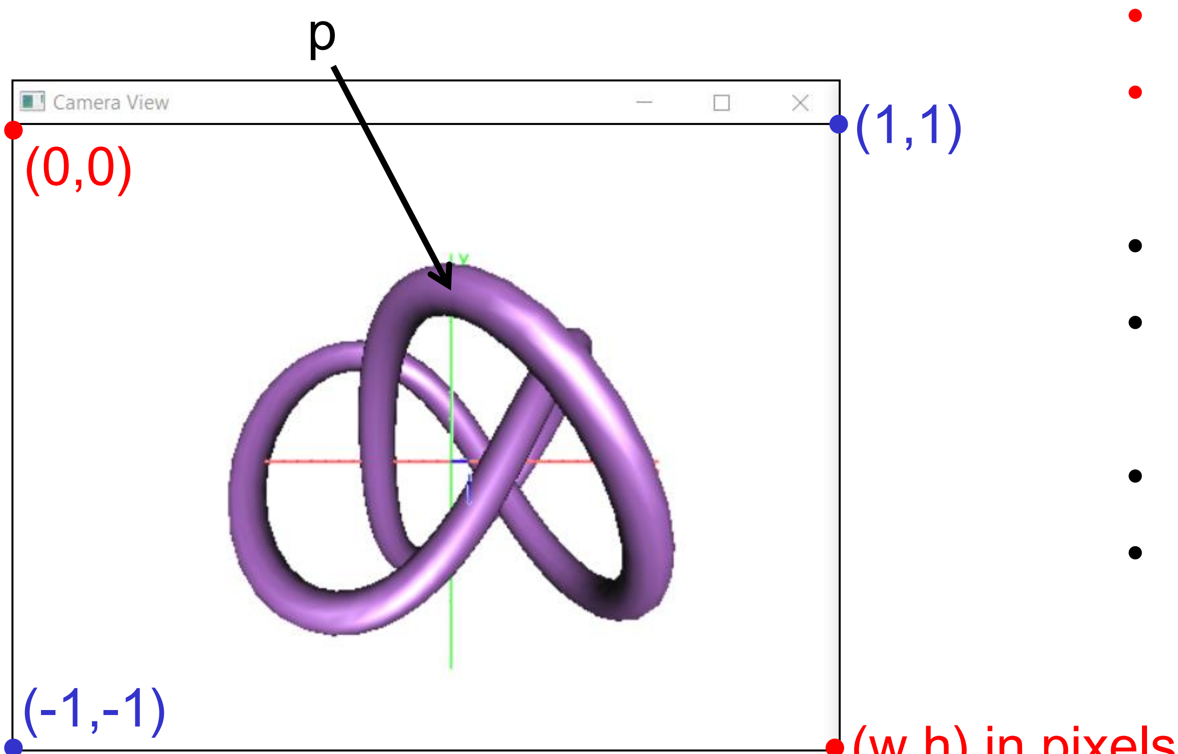
mouse click on
the screen plane,
for ex. here:

→ corresponding ray
going into the scene:



Picking

- p = (500,250) → the pixel coordinates you would receive in the callback function
- assuming a window of 1000x1000:
  - normalized **p = (0.0,0.5)**



- 500/1000 = 0.5
- 250/1000 = 0.25

- 0.5 – 0.5 = 0.0
- (1 - 0.25) – 0.5 = 0.25

- 0.0 * 2 = 0.0
- 0.25 * 2 = 0.5

Picking

- First, normalize p as per the previous slide

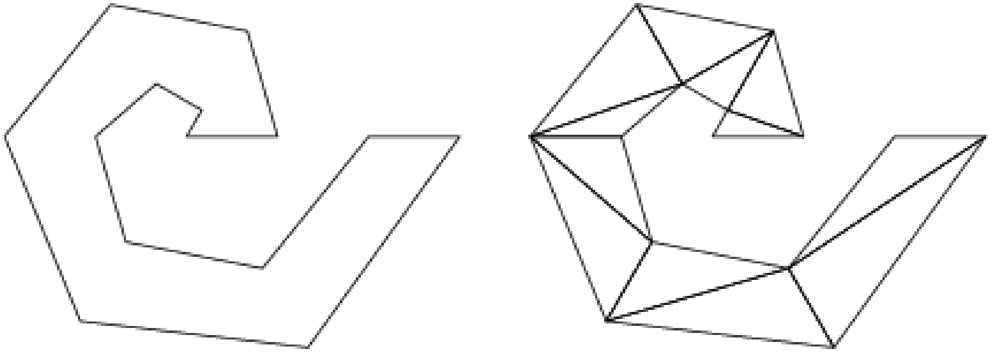
- Second, compute the ray endpoints:
  – Recall our viewing transformation: $M = P\ M_{cam}$
  – Given p in normalized 2D window coordinates:
    – Ray $p1 = M^{-1}$ (p.x, p.y, near plane z)
    – Ray p2 = p1-eye

- Then:
  – Intersect ray with all objects (triangles) in the scene
  – Return the object with closest intersection to the eye position of the camera

Triangulation

Triangulation

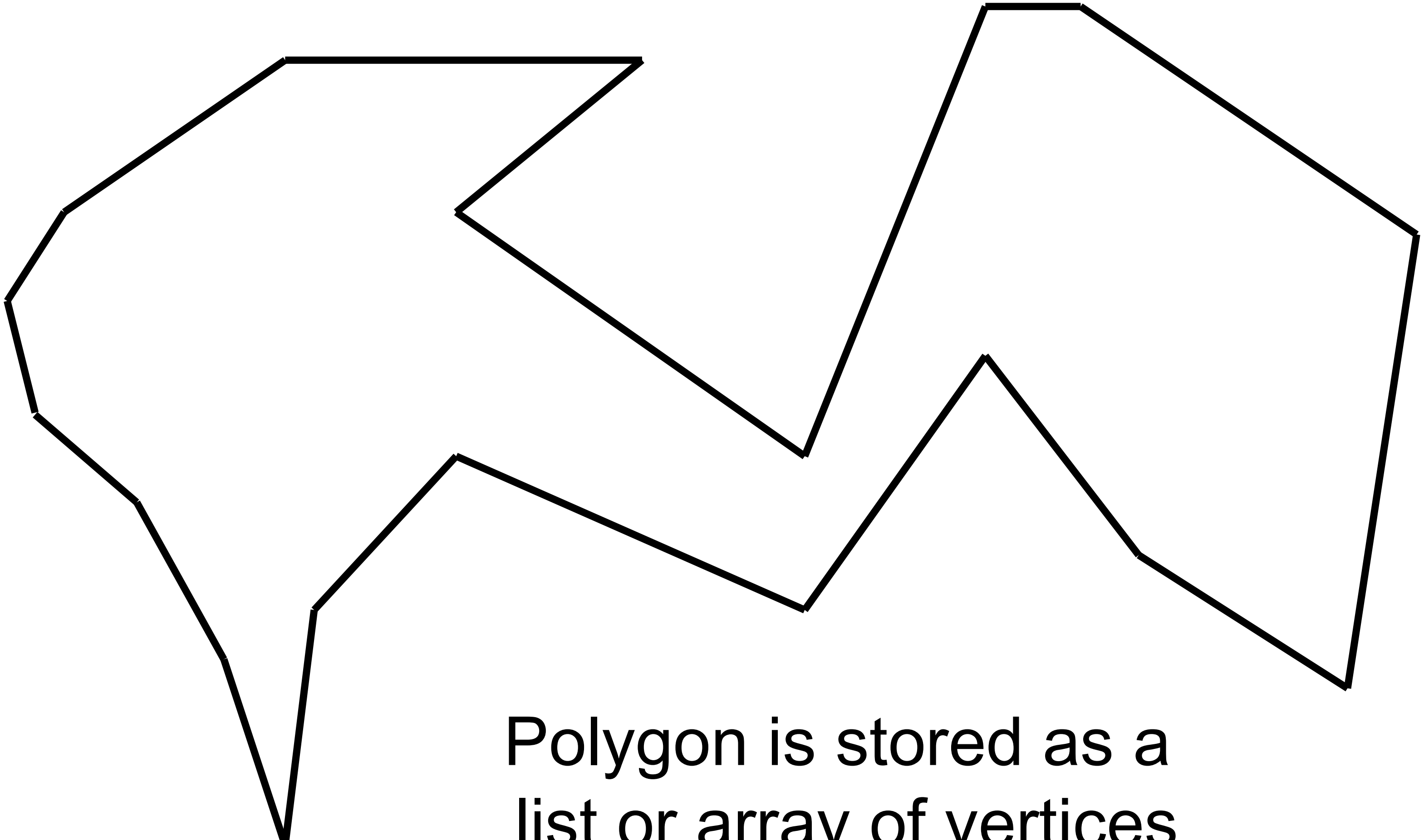
- It is much easier to work with triangles than with entire surfaces or planes
- Most of the algorithms in graphics pre-suppose we are working with triangles
- The graphics pipeline and hardware is also built for working with triangles
- Therefore, triangulation – the division of a surface into triangles – is extremely important



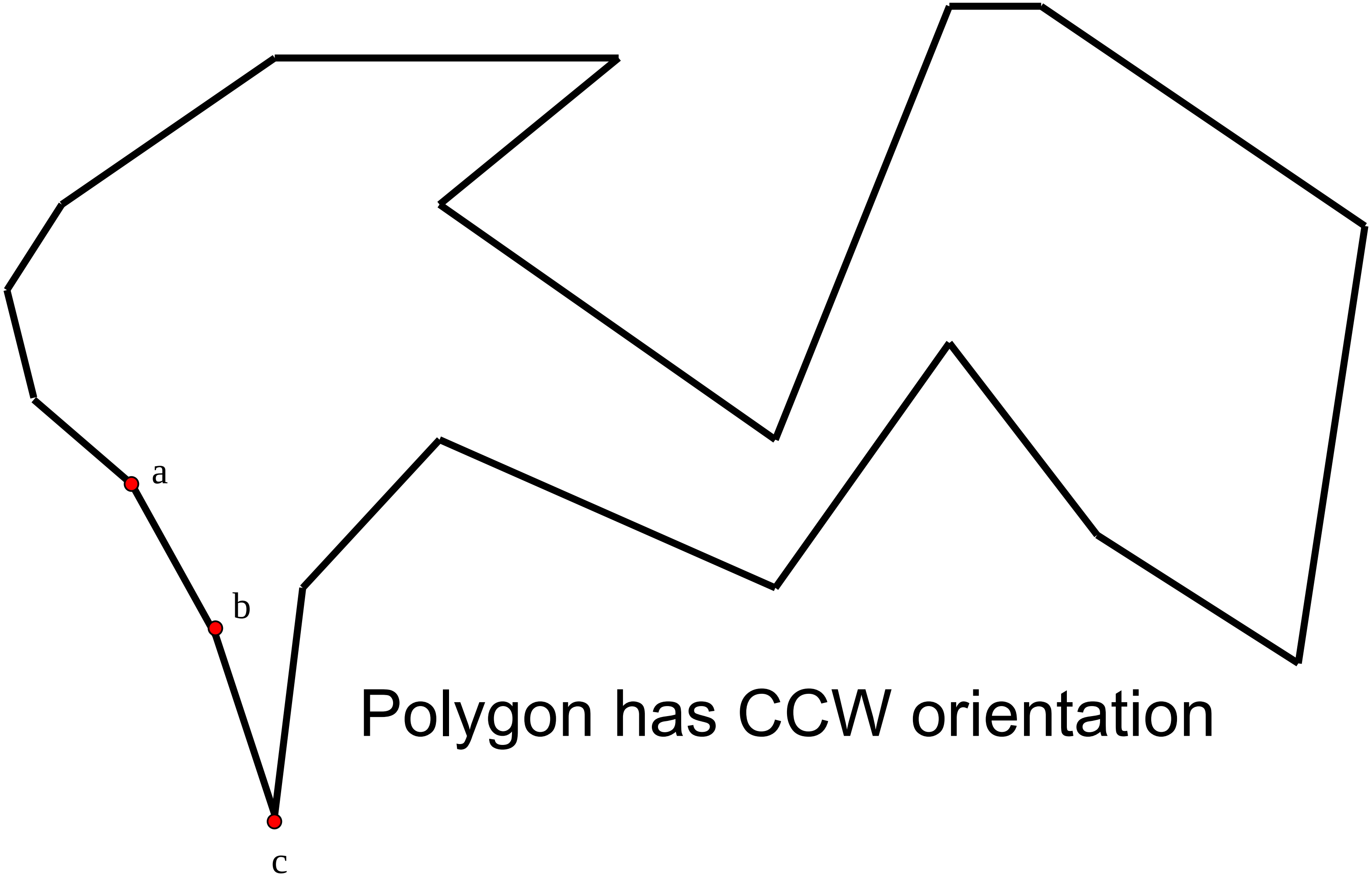
Simple Polygon Triangulation

- Many methods exist...
- Fastest
  - Linear O(n) time [Chazelle 1991], however the algorithm is complex
- Quality
  - We prefer triangles of good shape, not too "thin"
    - Constrained Delaunay Triangulation very popular, it optimizes the minimum internal angle of all triangles
    - Can be implemented in $O(n \log n)$
- Easiest
  - "Ear triangulation"
    - Simple implementation: $O(n^3)$
    - Keeping track of convex and concave lists: $O(n^2)$

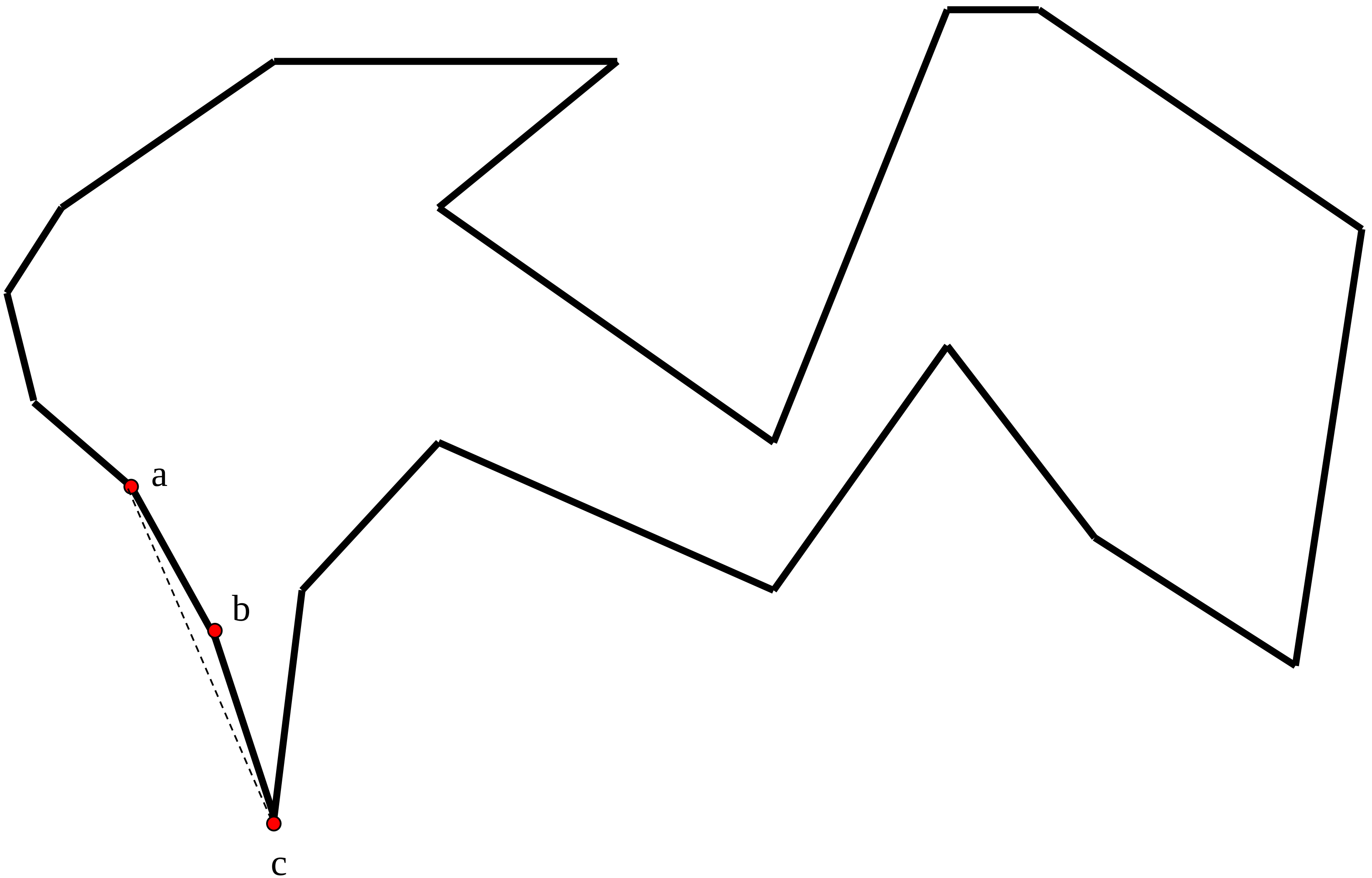
Ear Triangulation:

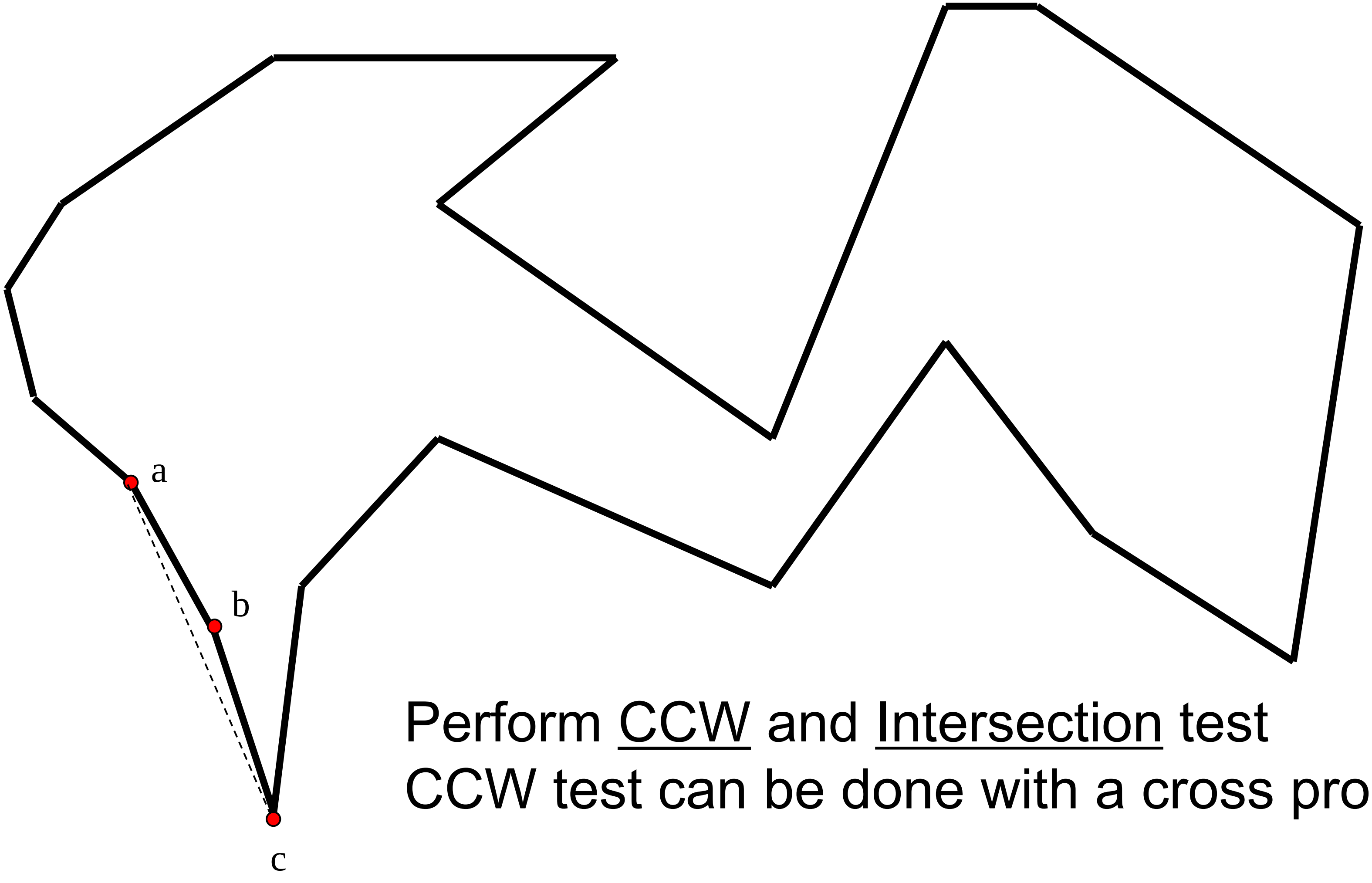


Polygon is stored as a list or array of vertices

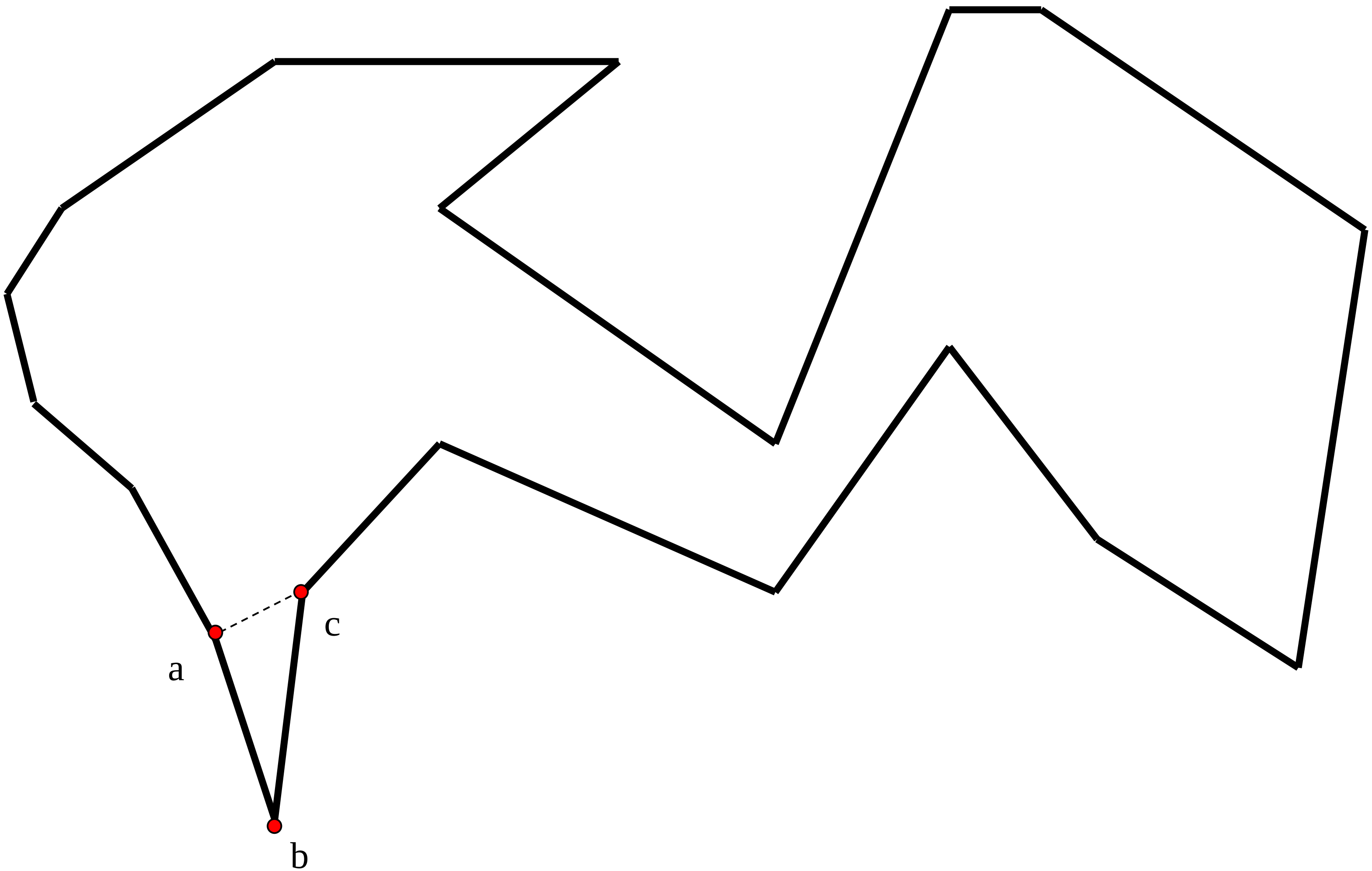


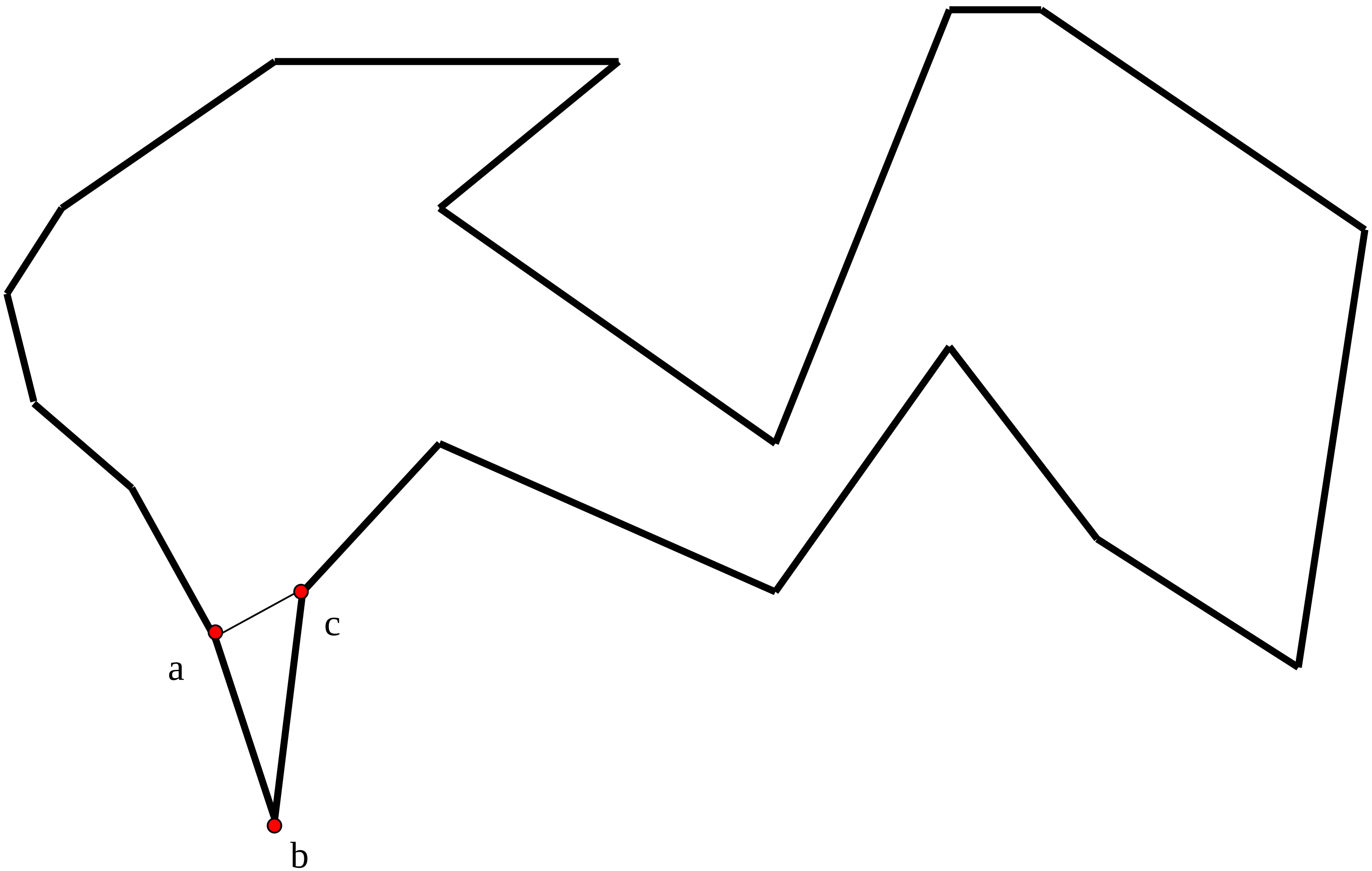
Polygon has CCW orientation

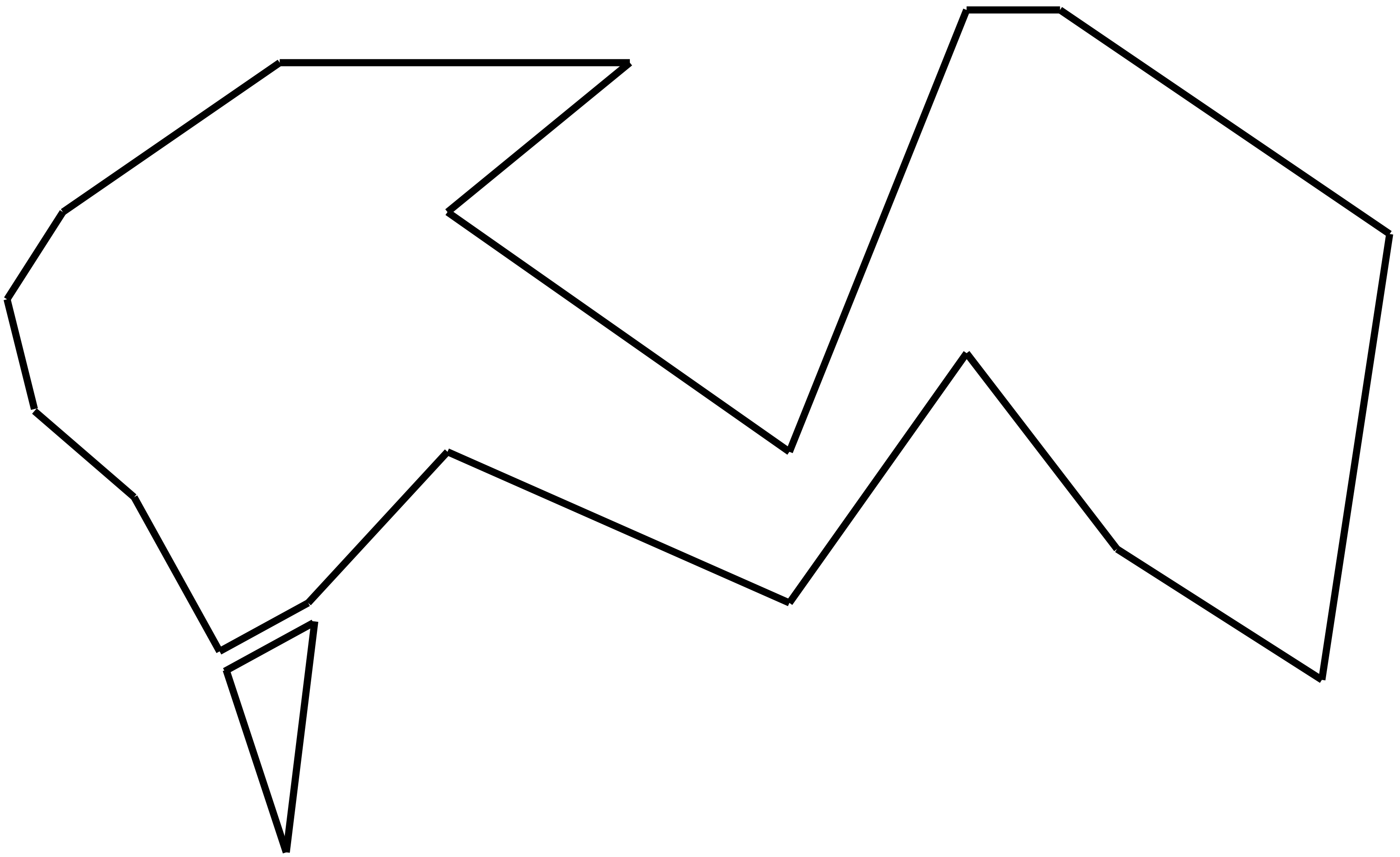

a
b
c

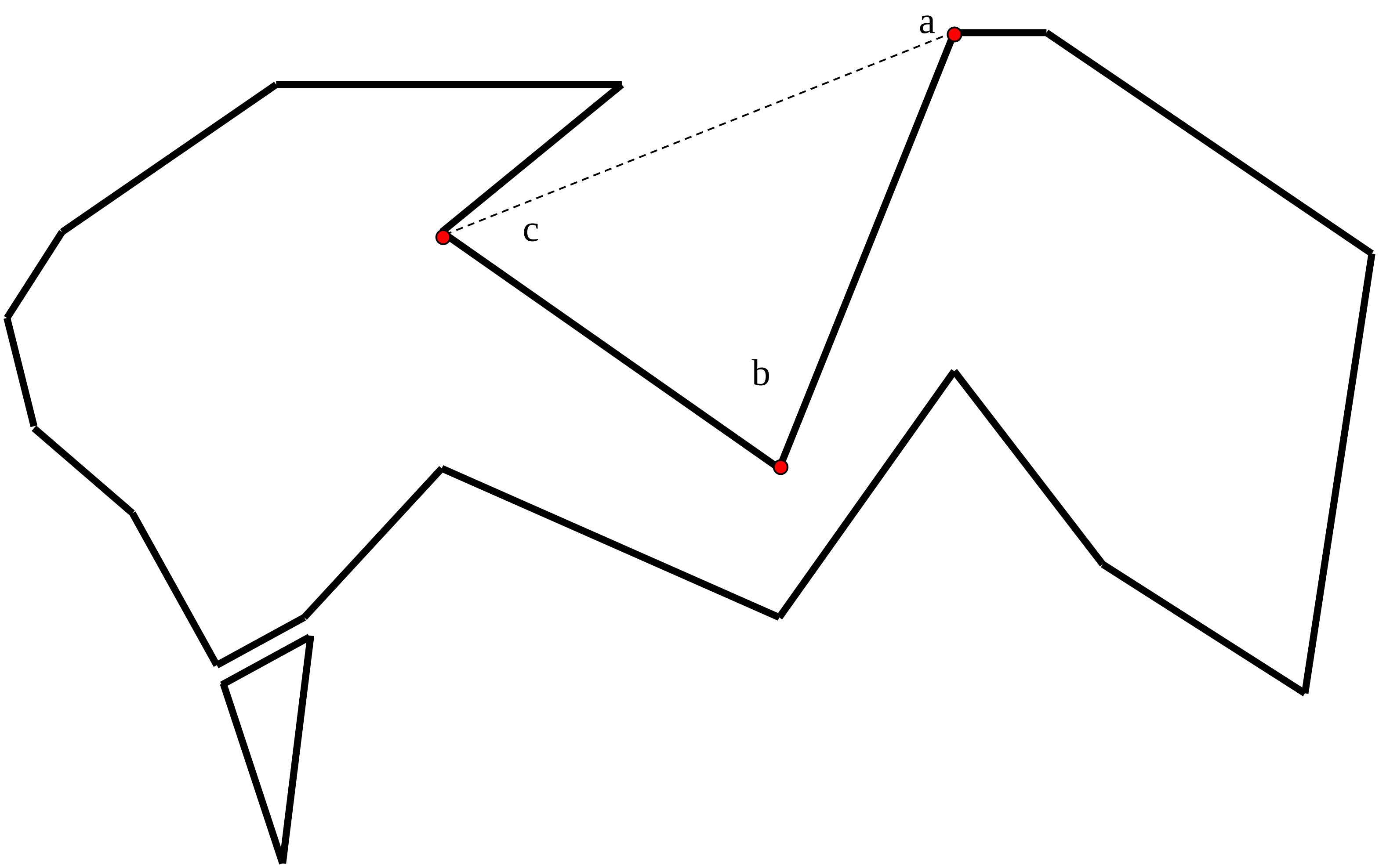


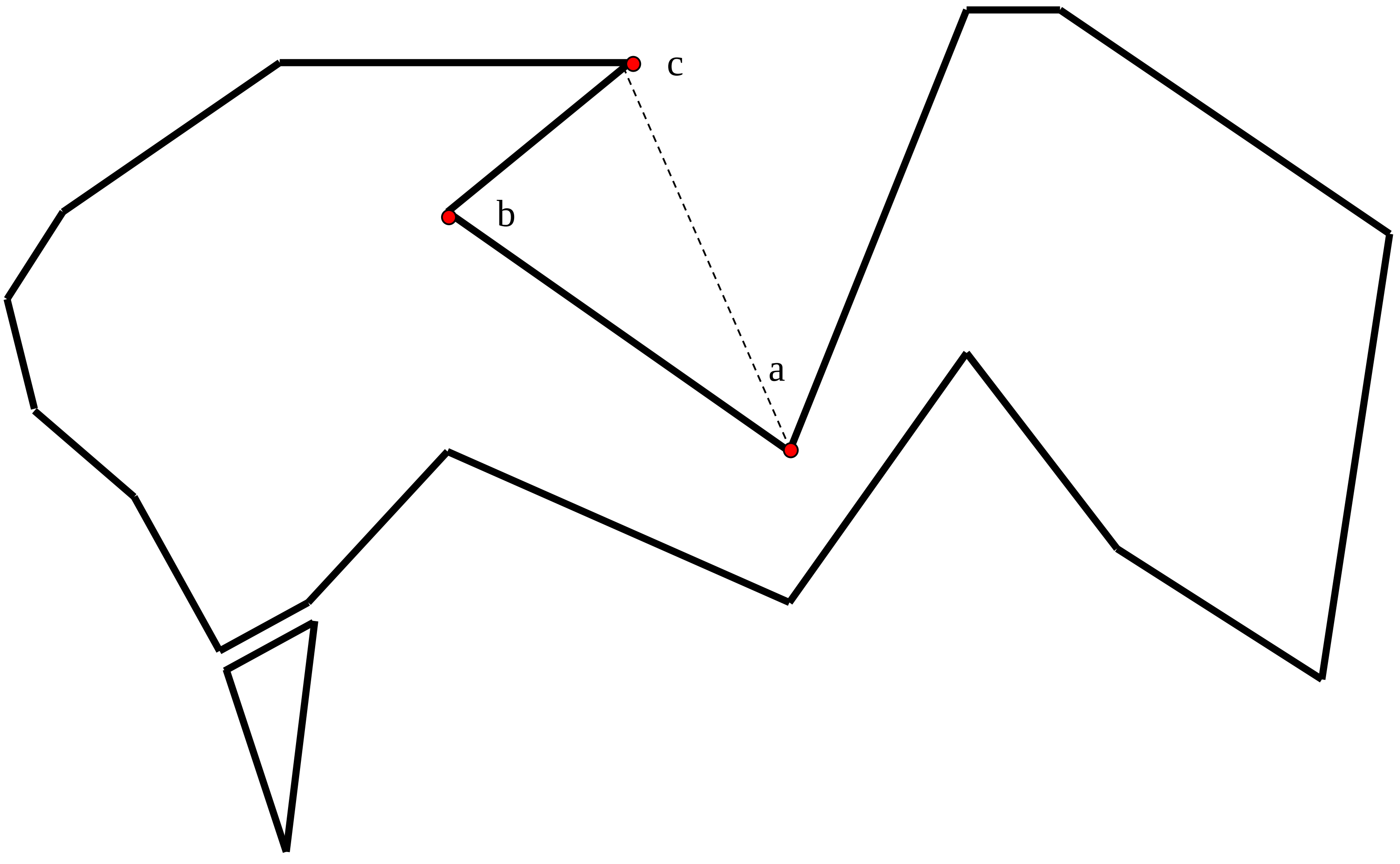
Perform <u>CCW</u> and <u>Intersection</u> test
CCW test can be done with a cross product

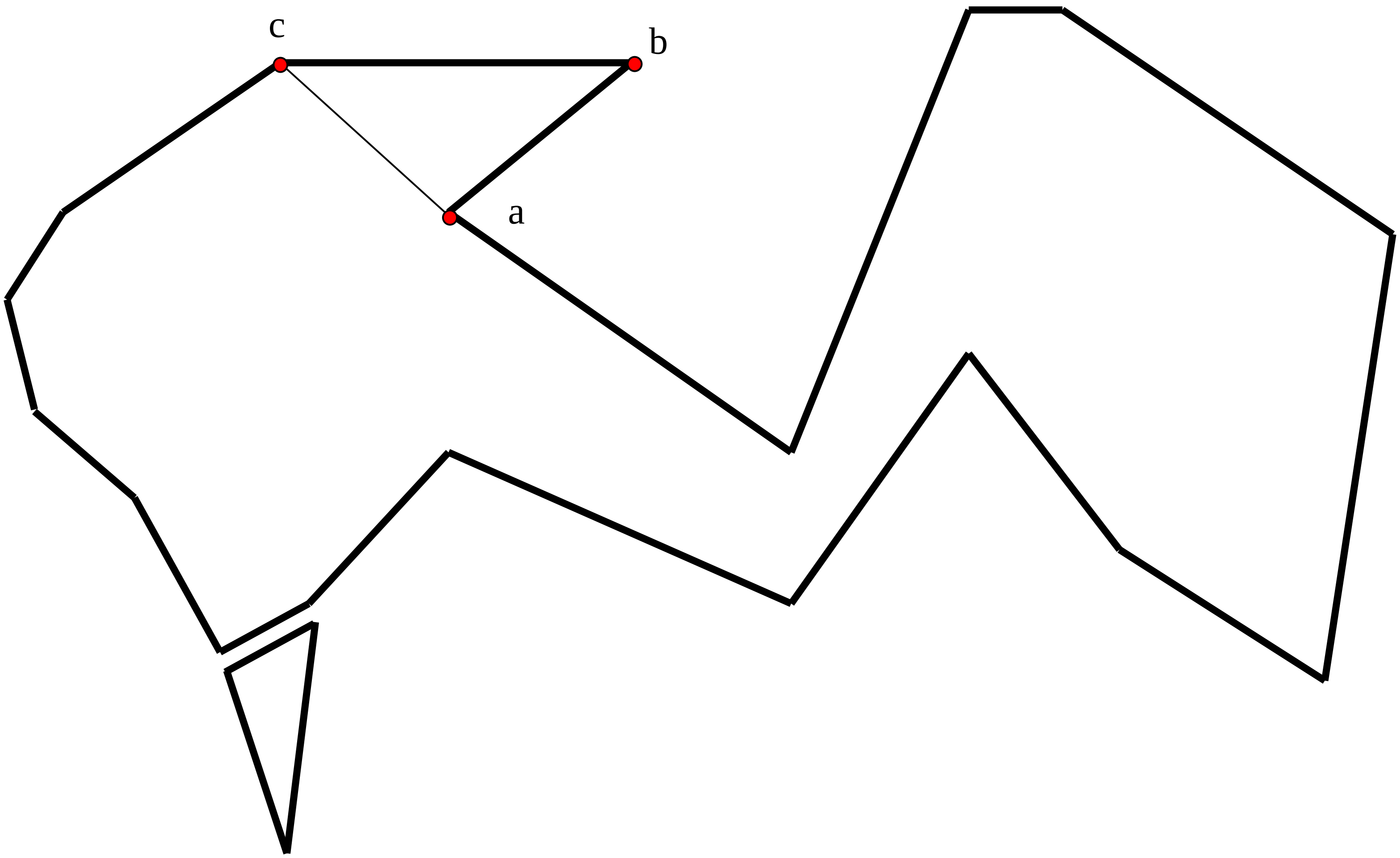

c
a
b

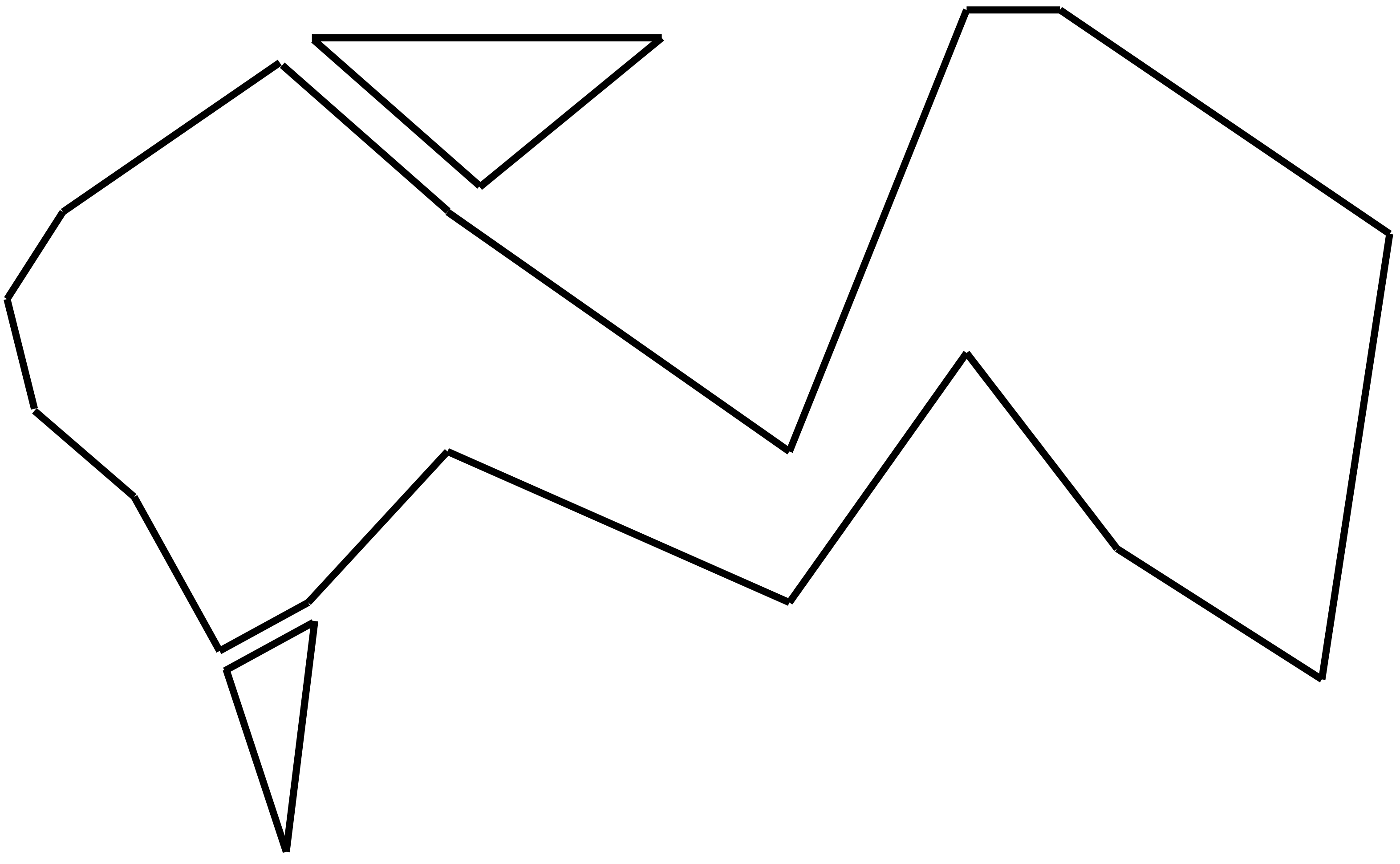

c
a
b

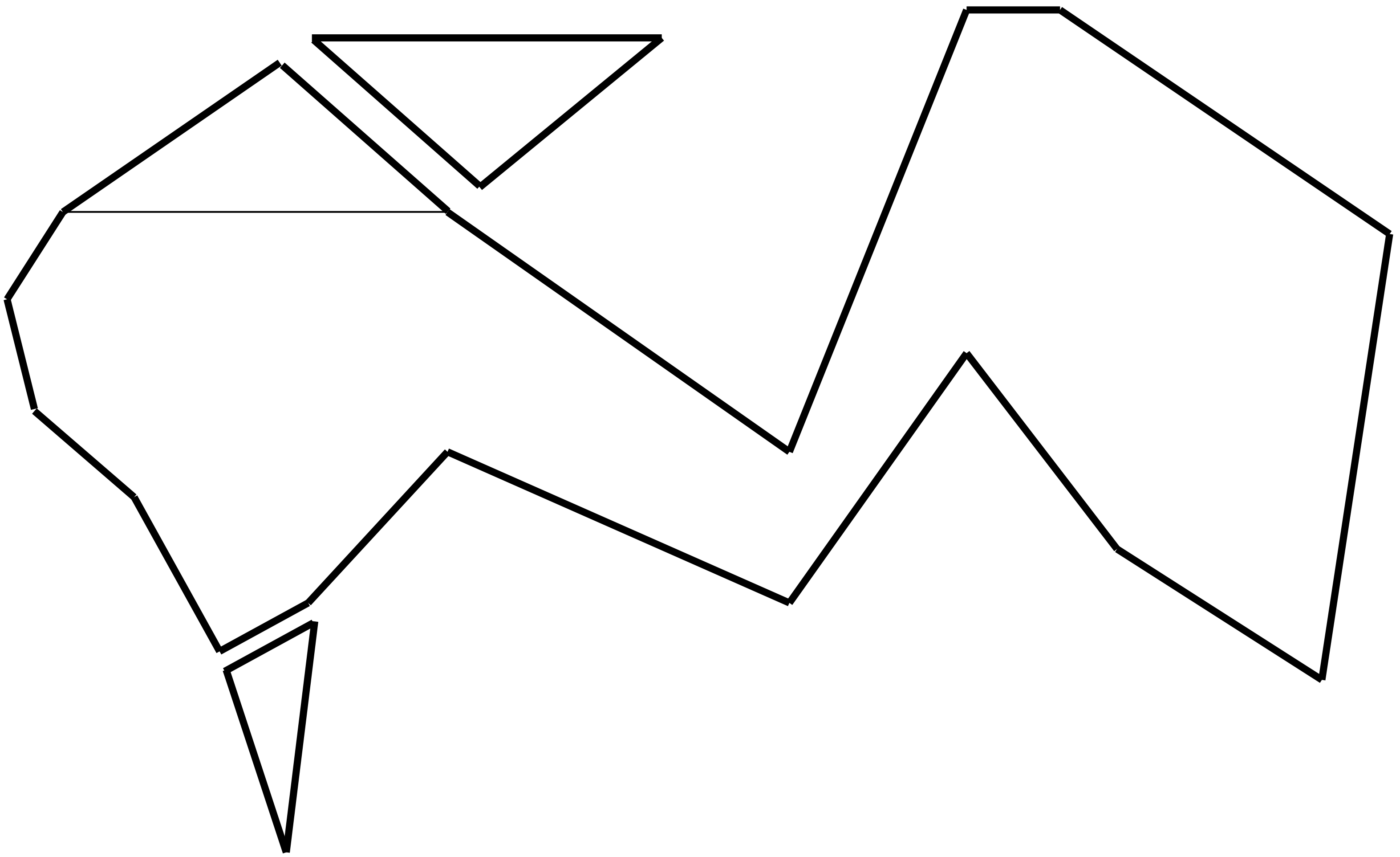


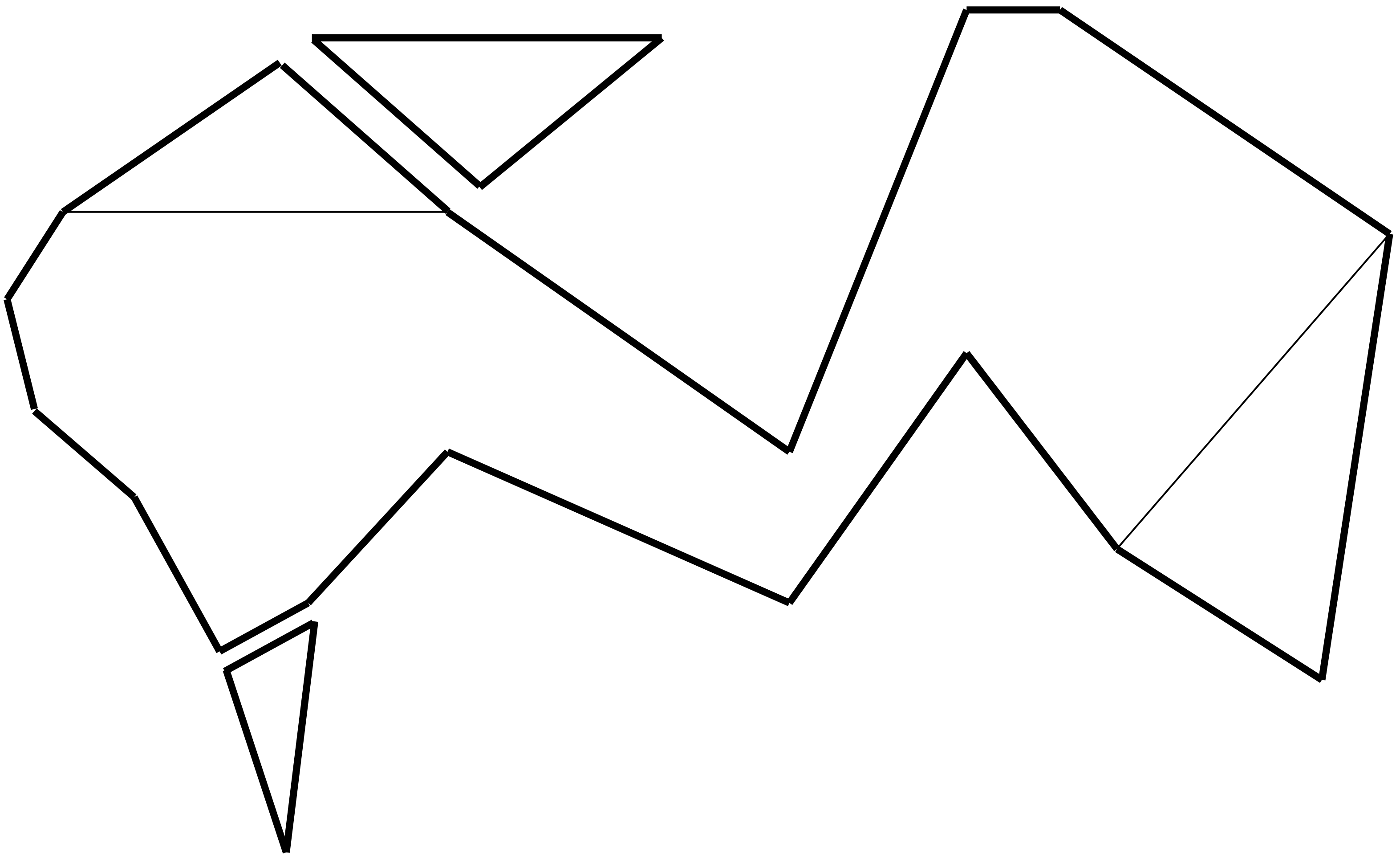

a
c
b

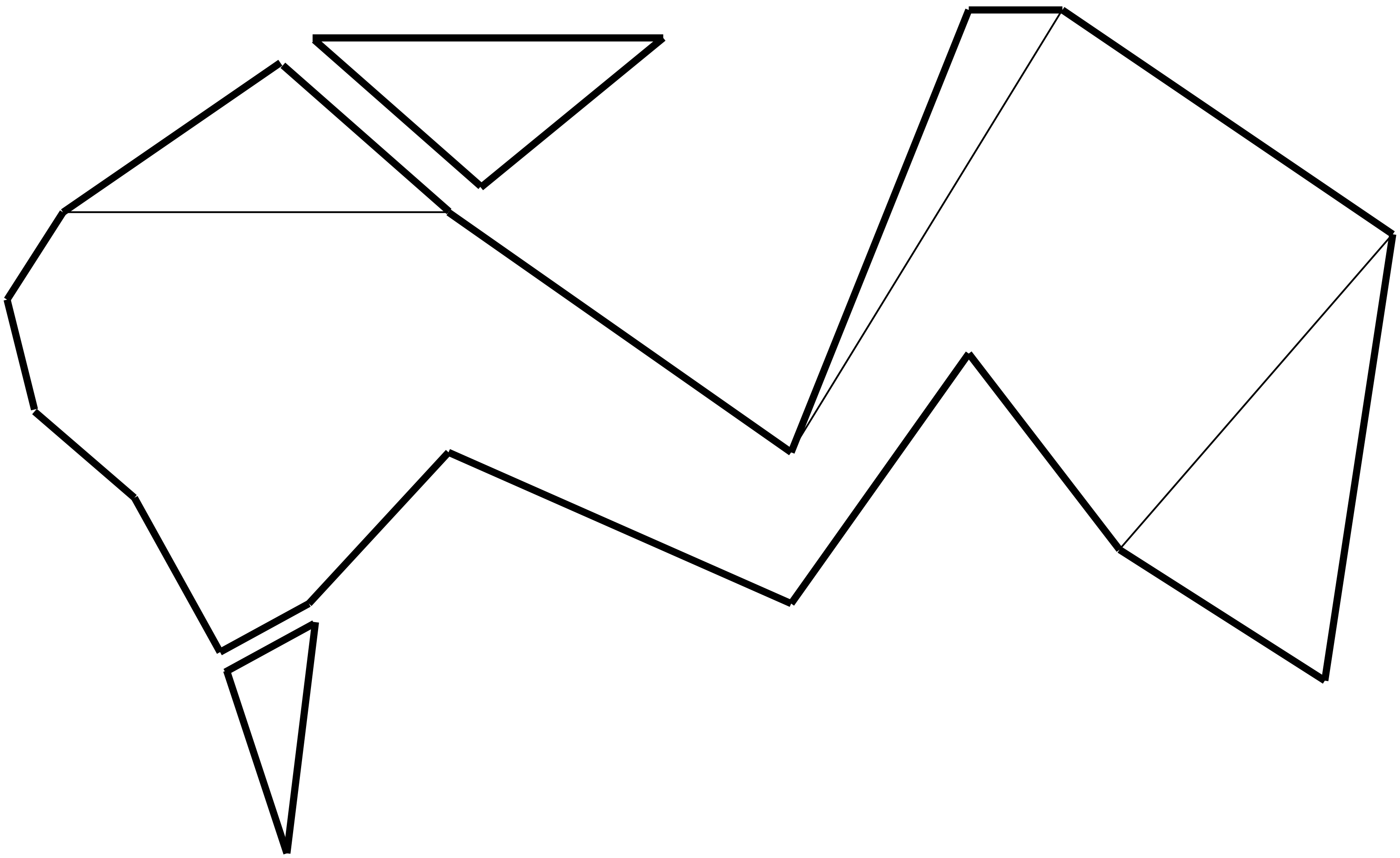

c
b
a

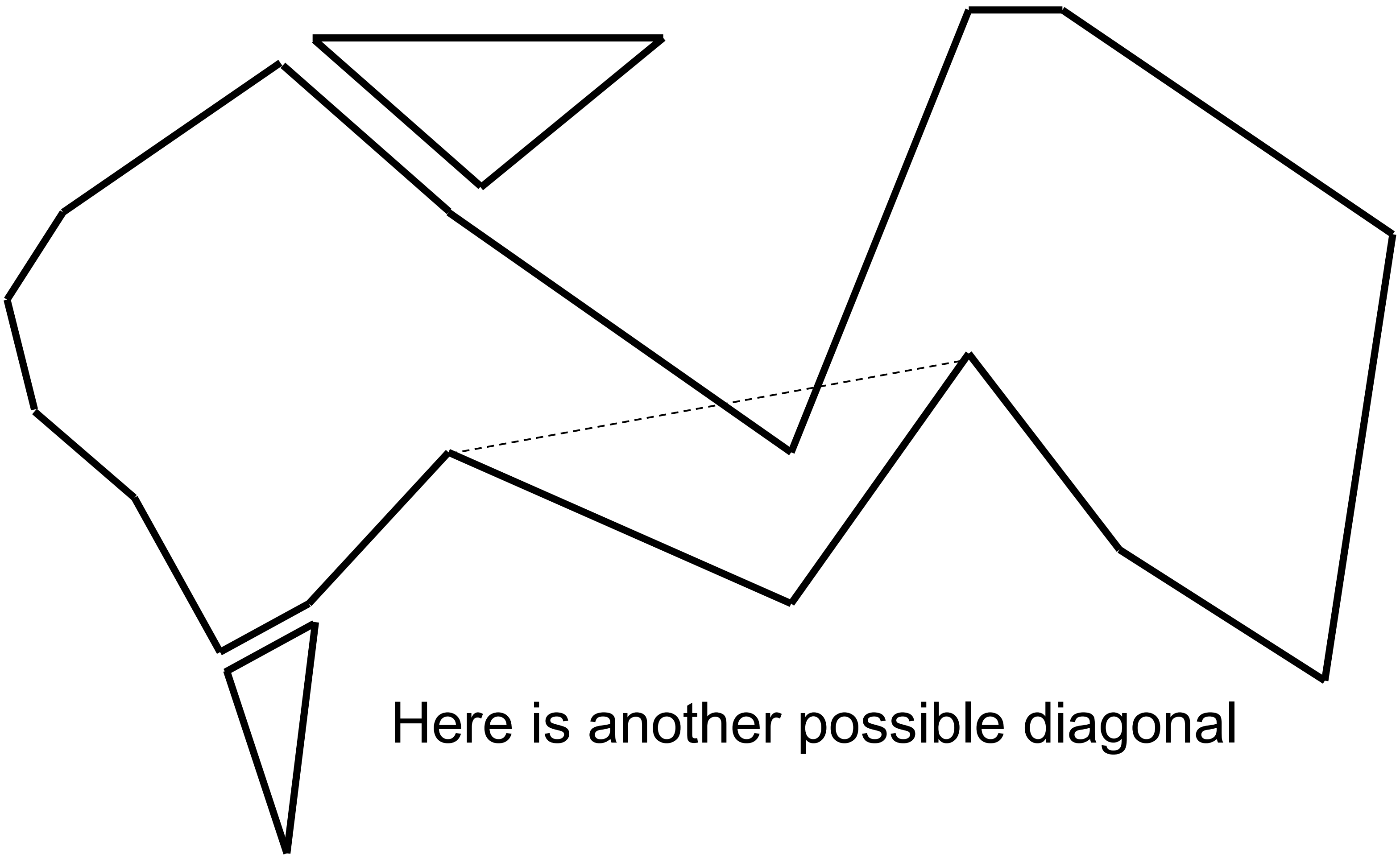

c
b
a



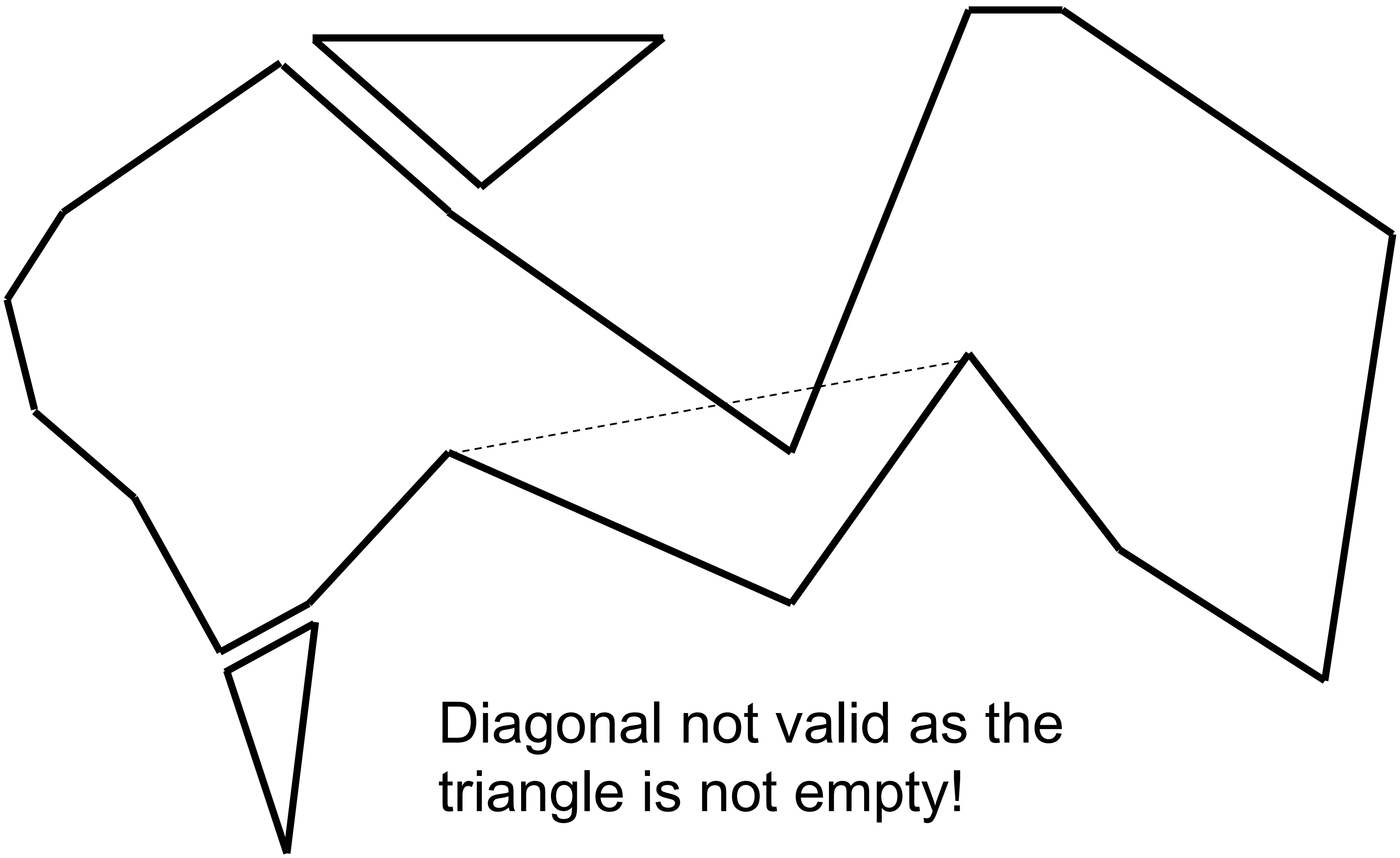








Here is another possible diagonal



Diagonal not valid as the triangle is not empty!