COMP3421

Vector geometry, Clipping

Transformations

- Object in model co-ordinates
- Transform into world co-ordinates
- Represent points in object as 1D Matrices
- Multiply by matrices to transform them

Matrices

2D array of numbers

$$\begin{pmatrix} 1 & 0 & 3 \\ 2 & 3 & 4 \\ 0 & 0 & 1 \end{pmatrix}$$

Vectors are just matrices with a single column

$$\begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

Matrix multiplication

$$\begin{pmatrix} 1 & 0 & 3 \\ 2 & 3 & 4 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 2 \end{pmatrix} = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix}$$

Matrix multiplication

$$\begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{3} \\ 2 & 3 & 4 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \mathbf{2} & 1 & 1 \\ \mathbf{0} & 0 & 1 \\ \mathbf{1} & 1 & 2 \end{pmatrix} = \begin{pmatrix} ? & & \\ & & \\ & & \end{pmatrix}$$

$$1 \times 2 + 0 \times 0 + 3 \times 1 = 5$$

Matrix multiplication

$$\begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{3} \\ 2 & 3 & 4 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \mathbf{2} & 1 & 1 \\ \mathbf{0} & 0 & 1 \\ \mathbf{1} & 1 & 2 \end{pmatrix} = \begin{pmatrix} 5 & & \\ & & \\ & & \end{pmatrix}$$

Matrix multiplication

$$\begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{3} \\ 2 & 3 & 4 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & \mathbf{1} & 1 \\ 0 & \mathbf{0} & 1 \\ 1 & \mathbf{1} & 2 \end{pmatrix} = \begin{pmatrix} 5 & ? & \\ & & \\ & & \end{pmatrix}$$

Matrix multiplication

$$\begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{3} \\ 2 & 3 & 4 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & \mathbf{1} & 1 \\ 0 & \mathbf{0} & 1 \\ 1 & \mathbf{1} & 2 \end{pmatrix} = \begin{pmatrix} 5 & 4 & \\ & & \\ & & \end{pmatrix}$$

Matrix multiplication

$$\begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{3} \\ 2 & 3 & 4 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 1 & \mathbf{1} \\ 0 & 0 & \mathbf{1} \\ 1 & 1 & \mathbf{2} \end{pmatrix} = \begin{pmatrix} 5 & 4 & ? \\ & & \\ & & \end{pmatrix}$$

Matrix multiplication

$$\begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{3} \\ 2 & 3 & 4 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 1 & \mathbf{1} \\ 0 & 0 & \mathbf{1} \\ 1 & 1 & \mathbf{2} \end{pmatrix} = \begin{pmatrix} 5 & 4 & 7 \\ & & \\ & & \end{pmatrix}$$

Matrix multiplication

$$\begin{pmatrix} 1 & 0 & 3 \\ \mathbf{2} & \mathbf{3} & \mathbf{4} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \mathbf{2} & 1 & 1 \\ \mathbf{0} & 0 & 1 \\ \mathbf{1} & 1 & 2 \end{pmatrix} = \begin{pmatrix} 5 & 4 & 7 \\ ? & & \\ & & \end{pmatrix}$$

Matrix multiplication

$$\begin{pmatrix} 1 & 0 & 3 \\ \mathbf{2} & \mathbf{3} & \mathbf{4} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \mathbf{2} & 1 & 1 \\ \mathbf{0} & 0 & 1 \\ \mathbf{1} & 1 & 2 \end{pmatrix} = \begin{pmatrix} 5 & 4 & 7 \\ 8 & & \\ & & \end{pmatrix}$$

Matrix multiplication

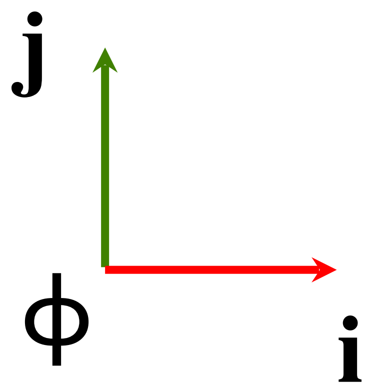
$$\begin{pmatrix} 1 & 0 & 3 \\ 2 & 3 & 4 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 2 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 2 \end{pmatrix} = \begin{pmatrix} 5 & 4 & 7 \\ 8 & 6 & 13 \\ 1 & 1 & 2 \end{pmatrix}$$

Coordinate frames

We can now think of a coordinate frame in terms of vectors.

A 2D frame is defined by:

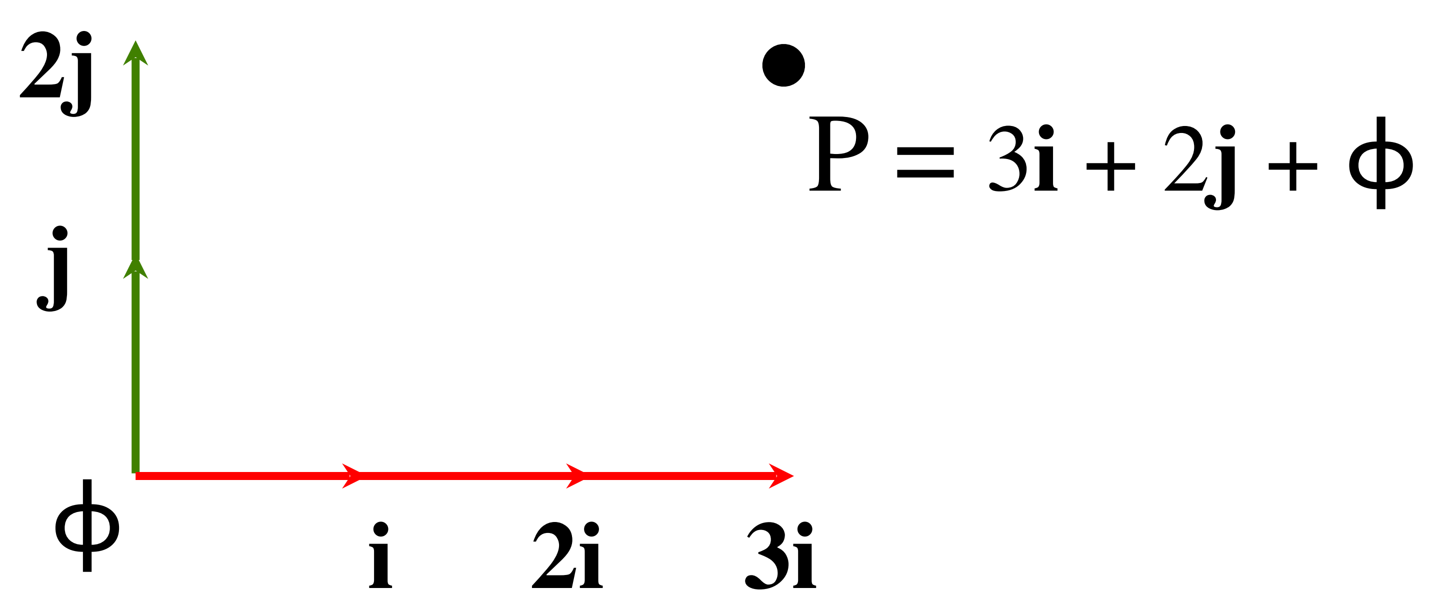
- an origin: ϕ
- 2 axis vectors: $\mathbf{i}, \mathbf{j}$



Points

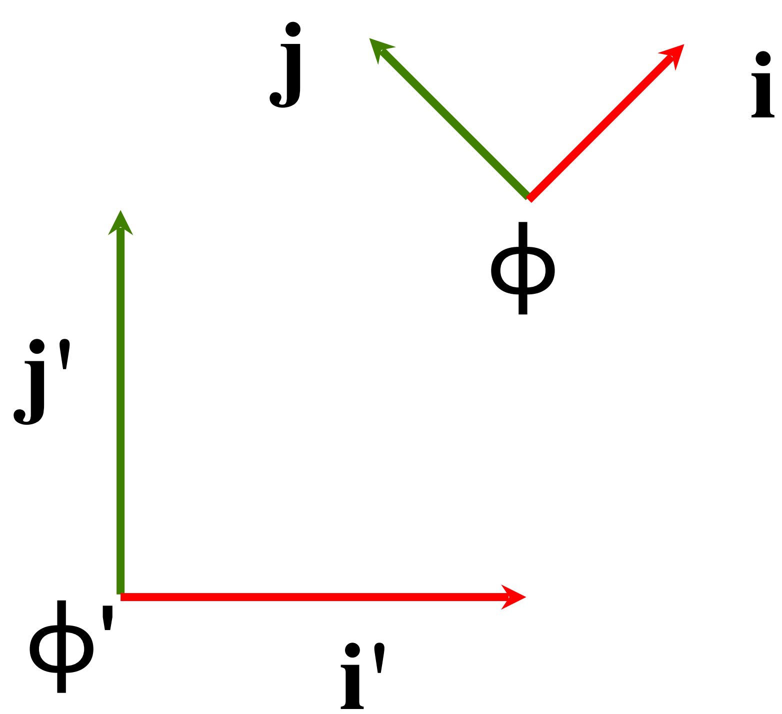
A point in a coordinate frame can be described as a displacement from the origin:

$$P = p_1\mathbf{i} + p_2\mathbf{j} + \phi$$



Transformation

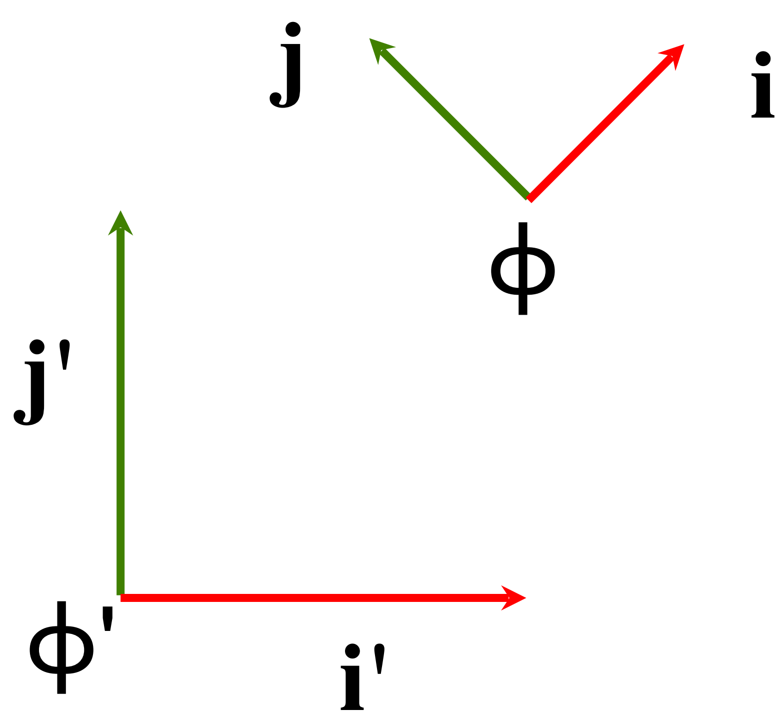
To convert P to a different coordinate frame, we just need to know how to convert $\mathbf{i}$, $\mathbf{j}$ and ϕ.



$$\begin{aligned} \phi &= 1\mathbf{i}' + 1\mathbf{j}' + \phi' \\ \mathbf{i} &= 0.4\mathbf{i}' + 0.4\mathbf{j}' \\ \mathbf{j} &= -0.4i' + 0.4\mathbf{j}' \end{aligned}$$

Transformation

To convert P to a different coordinate frame, we just need to know how to convert $\mathbf{i}$, $\mathbf{j}$ and ϕ.



$$
\begin{aligned}
P &= 3\mathbf{i} + 2\mathbf{j} + \phi \\
&= 3(0.4\mathbf{i}' + 0.4\mathbf{j}') + \\
&\quad 2(-0.4\mathbf{i}' + 0.4\mathbf{j}') + \\
&\quad 1\mathbf{i}' + 1\mathbf{j}' + \phi' \\
&= 1.4\mathbf{i}' + 3\mathbf{j}' + \phi'
\end{aligned}
$$

Transformation

This transformation is much easier to represent as a matrix:

$$
\begin{aligned}
P &= \begin{array}{c} \begin{array}{ccc} \mathbf{i} & \mathbf{j} & \phi \end{array} \\ \begin{pmatrix} 0.4 & -0.4 & 1 \\ 0.4 & 0.4 & 1 \\ 0 & 0 & 1 \end{pmatrix} \end{array} \begin{pmatrix} 3 \\ 2 \\ 1 \end{pmatrix} \\
&= \begin{pmatrix} 1.4 \\ 3 \\ 1 \end{pmatrix}
\end{aligned}
$$

Homogenous coordinates

We can use a single notation to describe both points and vectors.

Homogenous coordinates have an extra dimension representing the origin:

$$P = \begin{pmatrix} p_1 \\ p_2 \\ 1 \end{pmatrix} \qquad v = \begin{pmatrix} v_1 \\ v_2 \\ 0 \end{pmatrix}$$

Includes Origin

Does not include origin

Points and vectors

We can add two vectors to get a vector:

$$(u_1, u_2, 0)^\top + (v_1, v_2, 0)^\top = (u_1 + v_1, u_2 + v_2, 0)^\top$$

We can add a vector to a point to get a new point:

$$(p_1, p_2, 1)^\top + (v_1, v_2, 0)^\top = (p_1 + v_1, p_2 + v_2, 1)^\top$$

We cannot add two points.

$$(p_1, p_2, 1)^\top + (q_1, q_2, 1)^\top = (p_1 + q_1, p_2 + q_2, \mathbf{2})^\top$$

Affine transformations

Transformations between coordinate frames can be represented as matrices:

$$Q = \mathbf{M}P$$

$$\begin{pmatrix} q_1 \\ q_2 \\ 1 \end{pmatrix} = \begin{pmatrix} i_1 & j_1 & \phi_1 \\ i_2 & j_2 & \phi_2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ 1 \end{pmatrix}$$

Matrices in this form (note the 0s with the 1 at the end of the bottom row) are called affine transformations .

Affine transformations

Similarly for vectors:

$$\mathbf{v} = \mathbf{M}\mathbf{u}$$

$$\begin{pmatrix} v_1 \\ v_2 \\ 0 \end{pmatrix} = \begin{pmatrix} i_1 & j_1 & \phi_1 \\ i_2 & j_2 & \phi_2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ 0 \end{pmatrix}$$

Basic transformations

All affine transformations can be expressed as combinations of four basic types:

- Translation
- Rotation
- Scale
- Shear

Affine transformations

Affine transformations preserve straight lines:

$$\underbrace{\mathbf{M}(A + t\mathbf{v}) = \mathbf{M}A}_{\text{point}} + \underbrace{t\mathbf{M}\mathbf{v}}_{\text{vector}}$$

They maintain parallel lines

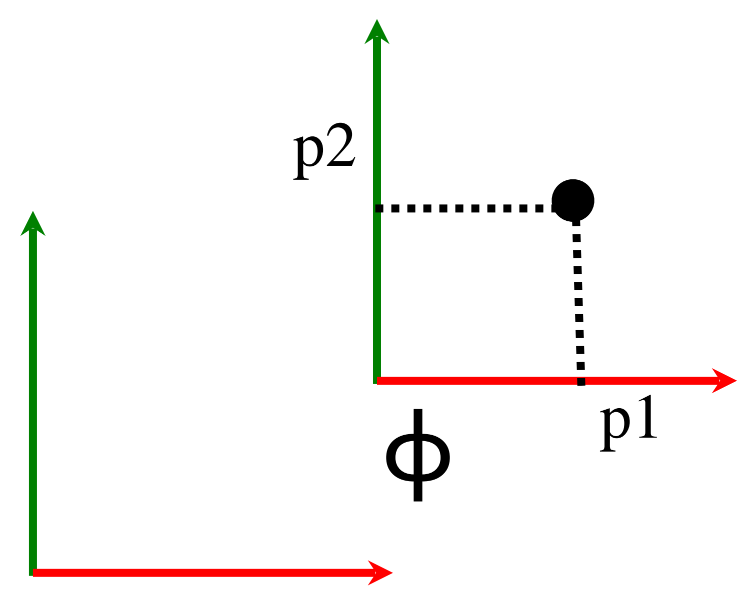
They maintain relative distances on lines (ie midpoints are still midpoints etc)

They don't always preserve angles or area

2D Translation

To translate the origin to a new point ϕ.

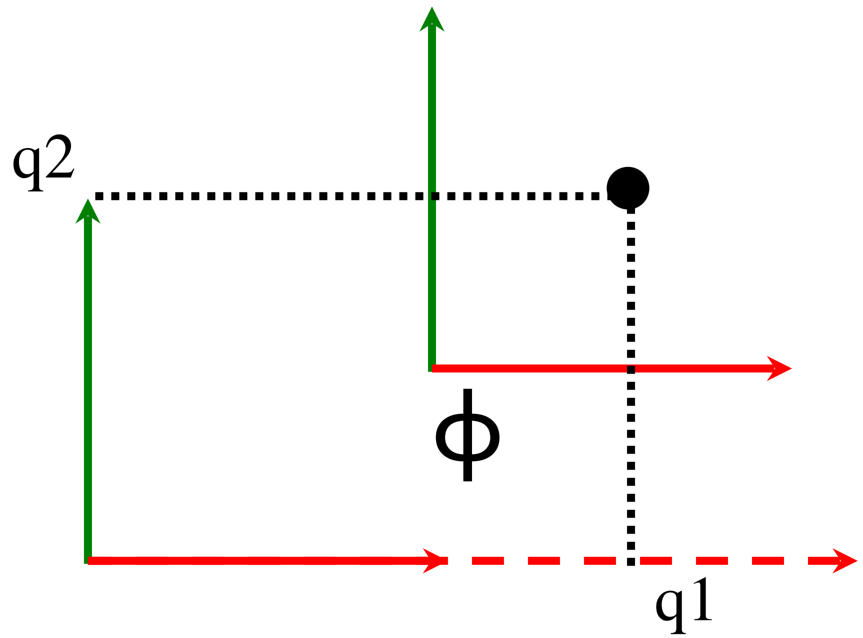
$$\begin{pmatrix} q_1 \\ q_2 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & \phi_1 \\ 0 & 1 & \phi_2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ 1 \end{pmatrix}$$



2D Translation

To translate the origin to a new point ϕ.

$$\begin{pmatrix} q_1 \\ q_2 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & \phi_1 \\ 0 & 1 & \phi_2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ 1 \end{pmatrix}$$



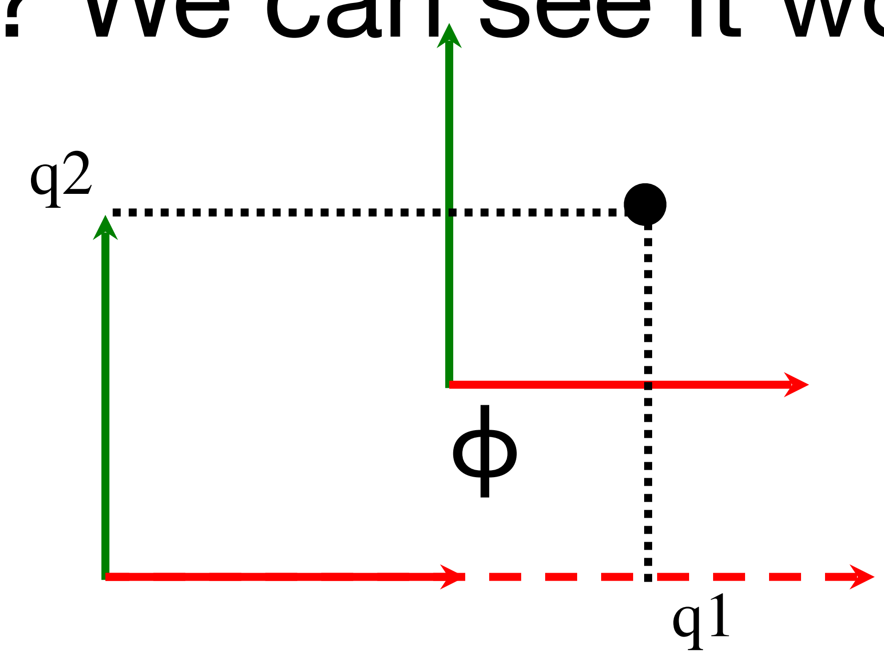
2D Translation

Translate by (1,0.5) then plot point

P = (0.5,0.5) in local frame.

What is the point in world

co-ordinates? We can see it would be (1.5,1)



Example:Converting from Local to Global

Q(Global) = M P(local)

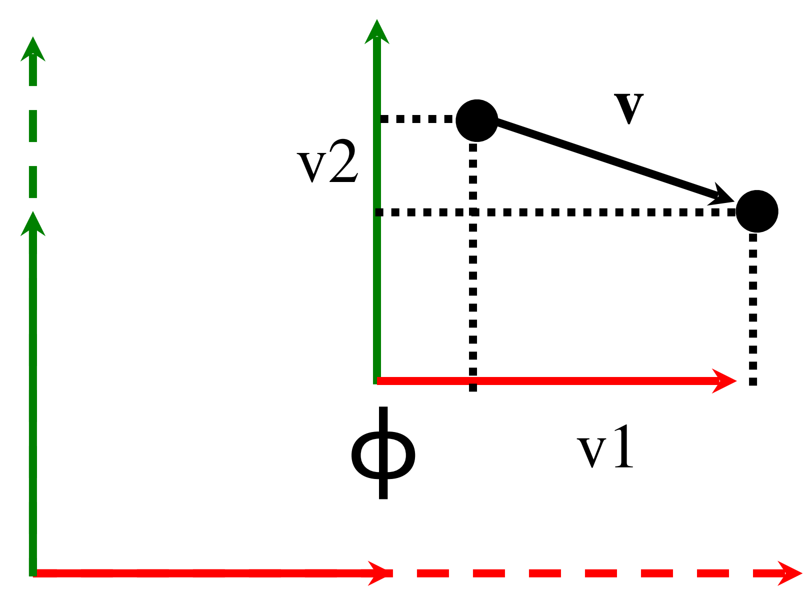
$$\begin{array}{ccccc} M & & P & & \\ \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0.5 \\ 0 & 0 & 1 \end{pmatrix} & & \begin{pmatrix} 0.5 \\ 0.5 \\ 1 \end{pmatrix} & = & \begin{pmatrix} 1.5 \\ 1 \\ 1 \end{pmatrix} \end{array}$$

So Q is (1.5,1)

2D Translation

Note: translating a vector has no effect.

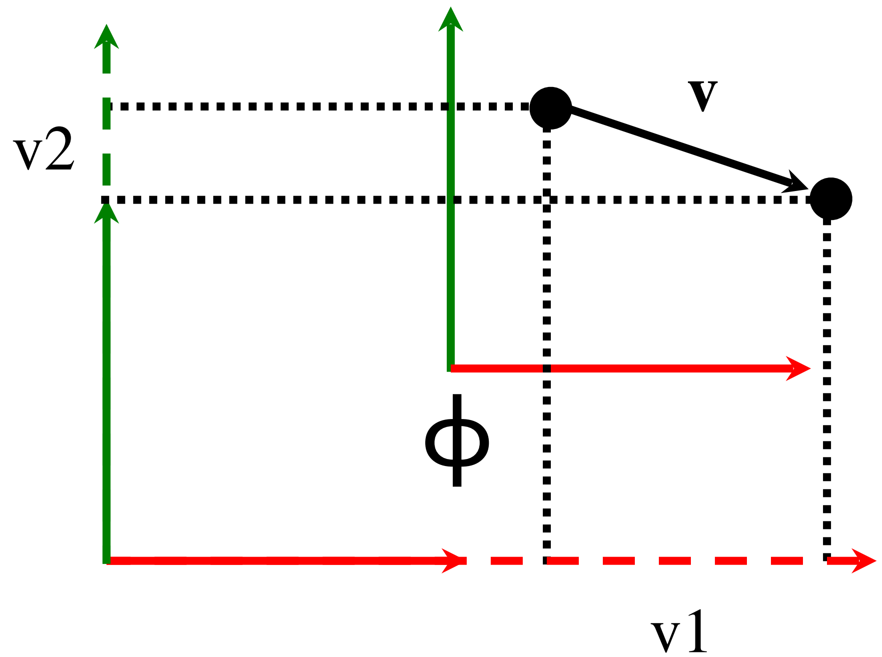
$$\begin{pmatrix} v_1 \\ v_2 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & \phi_1 \\ 0 & 1 & \phi_2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ 0 \end{pmatrix}$$



2D Translation

Note: translating a vector has no effect.

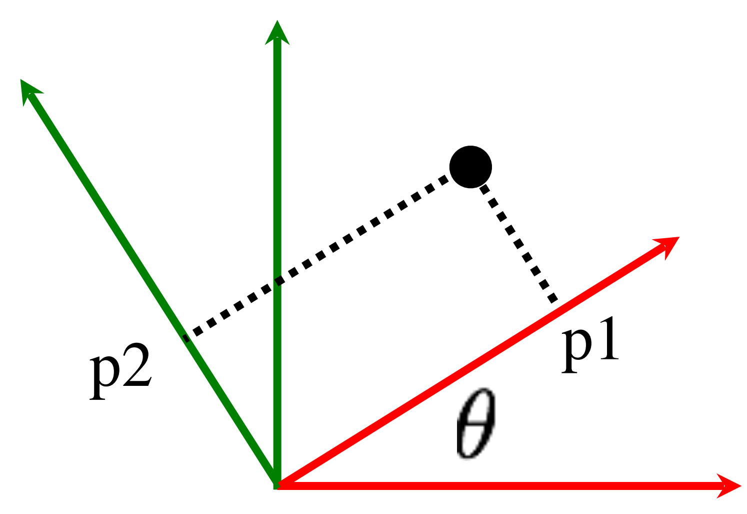
$$\begin{pmatrix} v_1 \\ v_2 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & \phi_1 \\ 0 & 1 & \phi_2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ 0 \end{pmatrix}$$



2D Rotation

To rotate a point about the origin:

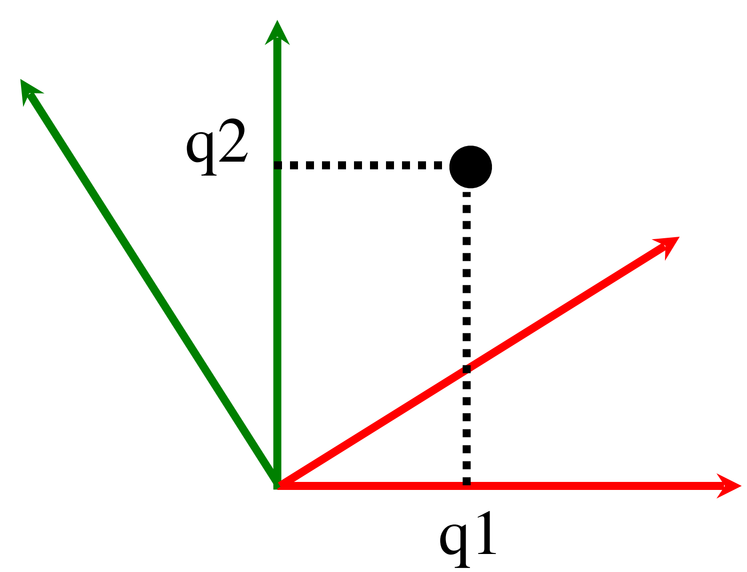
$$\begin{pmatrix} q_1 \\ q_2 \\ 1 \end{pmatrix} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ 1 \end{pmatrix}$$



2D Rotation

To rotate a point about the origin:

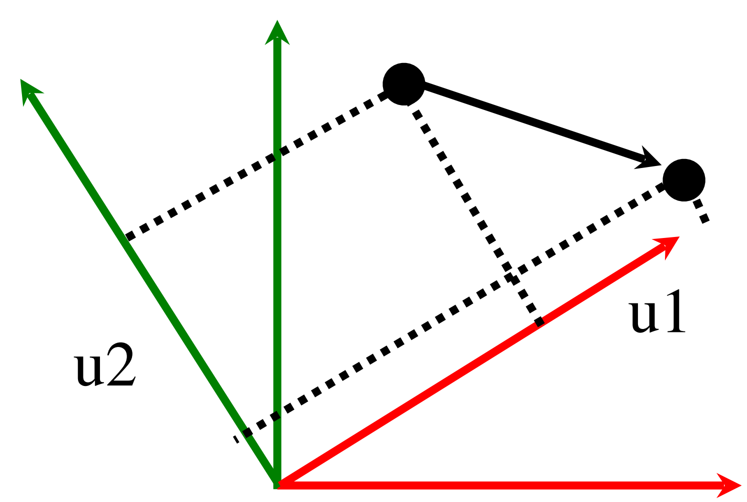
$$\begin{pmatrix} q_1 \\ q_2 \\ 1 \end{pmatrix} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ 1 \end{pmatrix}$$



2D Rotation

Likewise to rotate a vector:

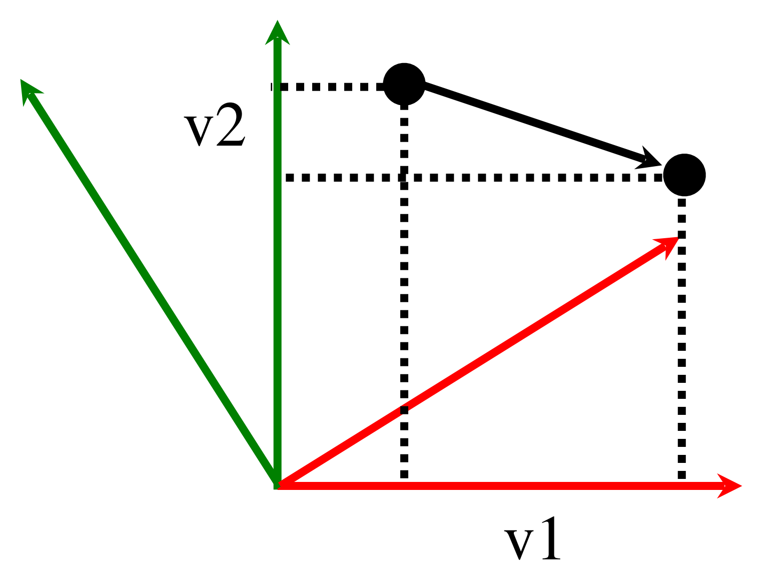
$$\begin{pmatrix} v_1 \\ v_2 \\ 0 \end{pmatrix} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ 0 \end{pmatrix}$$



2D Rotation

Likewise to rotate a vector:

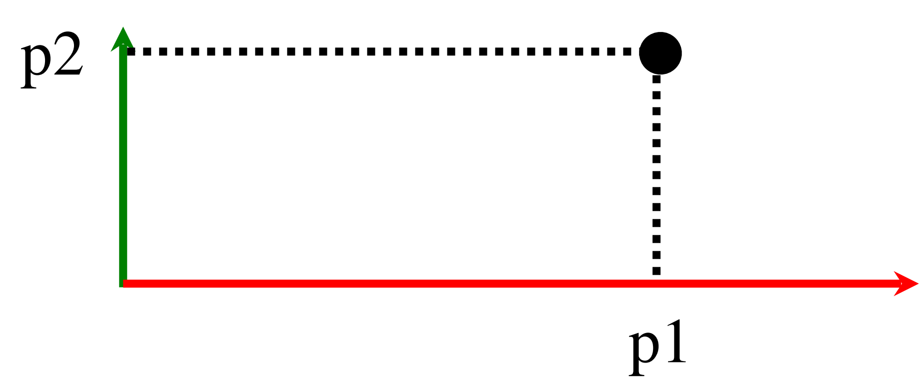
$$\begin{pmatrix} v_1 \\ v_2 \\ 0 \end{pmatrix} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ 0 \end{pmatrix}$$



2D Scale

To scale a point by factors (sx, sy) about the origin:

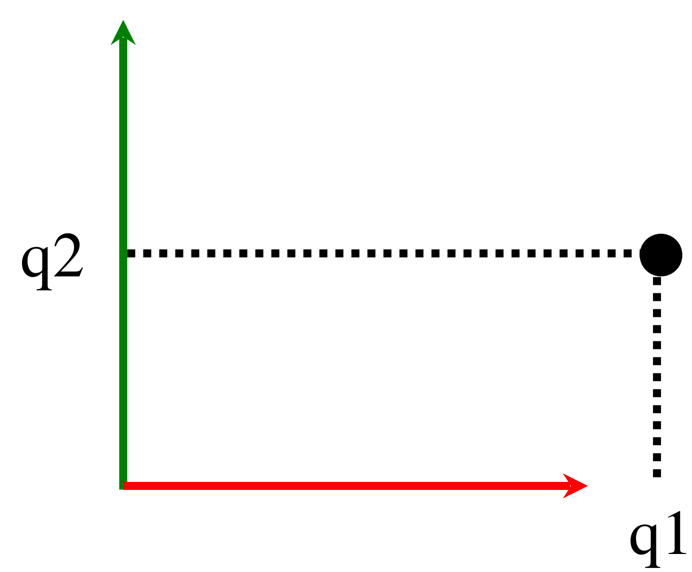
$$\begin{pmatrix} s_x p_1 \\ s_y p_2 \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ 1 \end{pmatrix}$$



2D Scale

To scale a point by factors (sx, sy) about the origin:

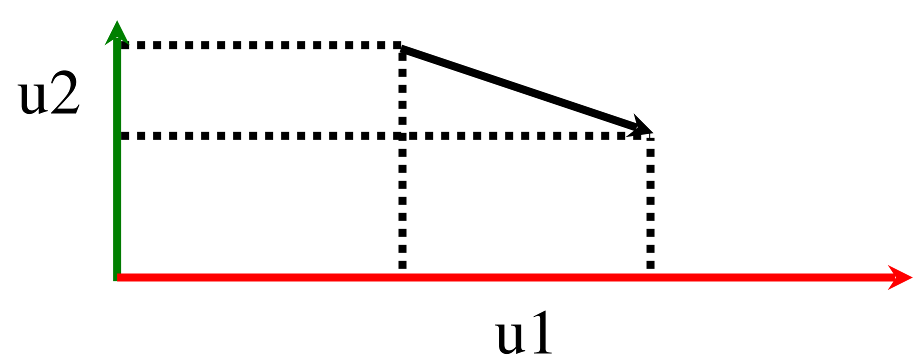
$$\begin{pmatrix} s_x p_1 \\ s_y p_2 \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ 1 \end{pmatrix}$$



2D Scale

Likewise to scale vectors:

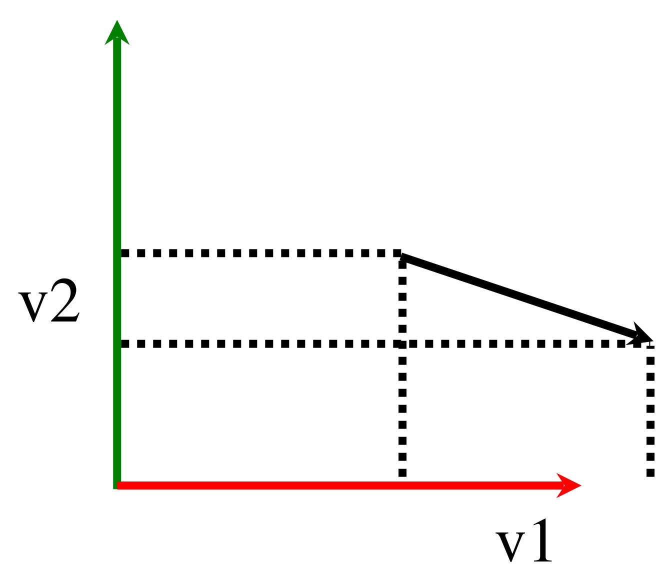
$$\begin{pmatrix} s_x v_1 \\ s_y v_2 \\ 0 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ 0 \end{pmatrix}$$



2D Scale

Likewise to scale vectors:

$$\begin{pmatrix} s_x v_1 \\ s_y v_2 \\ 0 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ 0 \end{pmatrix}$$



Shear

Shear is the unwanted child of affine transformations.

It can occur when you scale axes non-uniformly and then rotate.

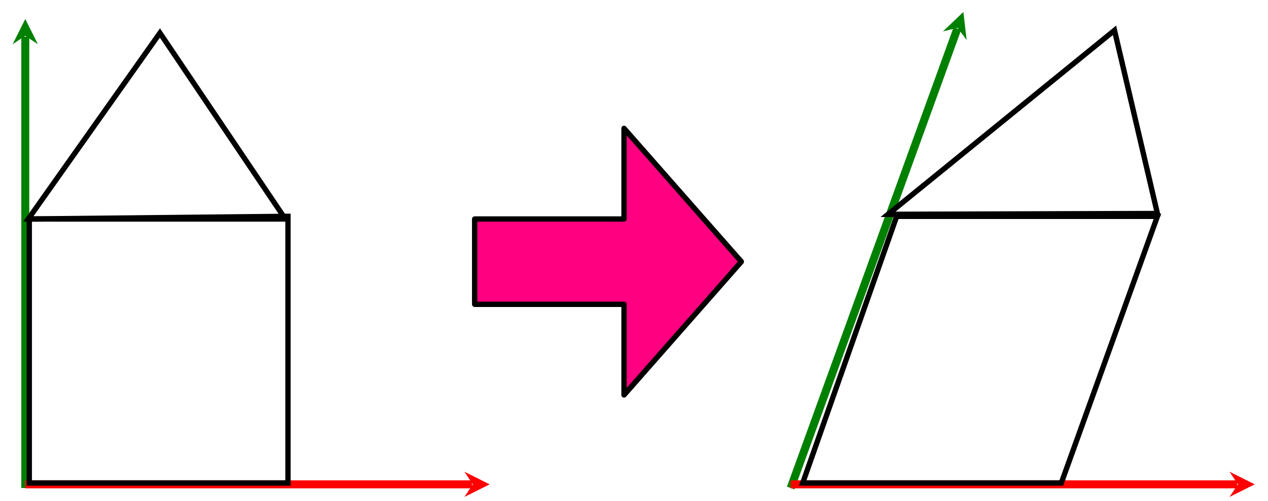
It does not preserve angles.

Usually it is not something you want.

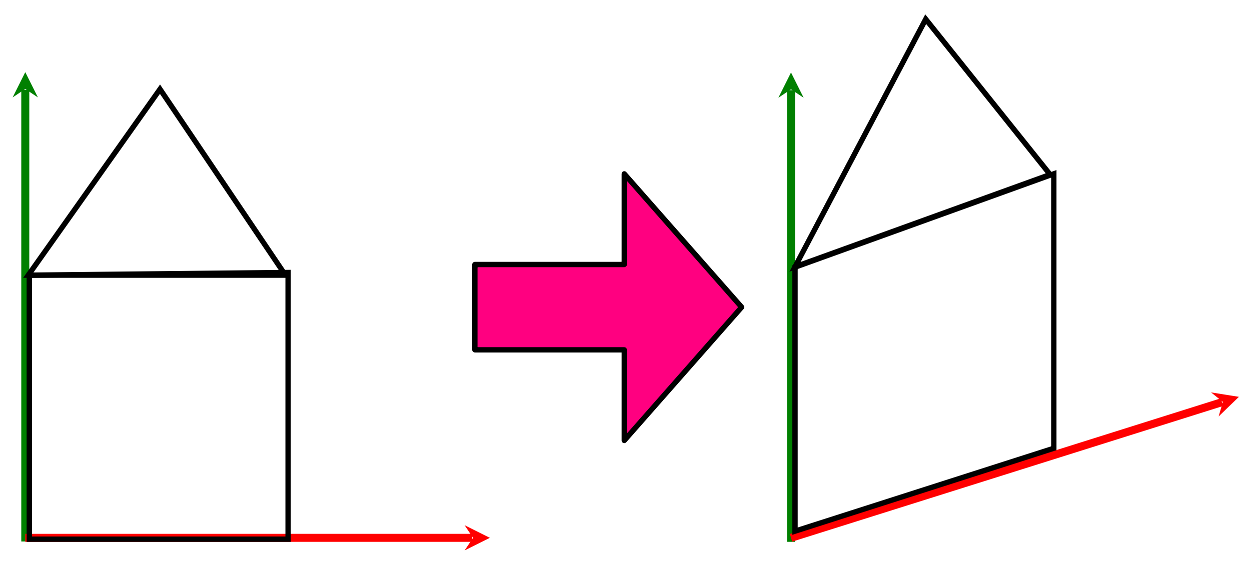
It can be avoided by always scaling uniformly.

Shear

Horizontal:



Vertical:



2D Shear

Horizontal:

$$\begin{pmatrix} q_1 \\ q_2 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & h & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ 1 \end{pmatrix}$$

Vertical:

$$\begin{pmatrix} q_1 \\ q_2 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ v & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ 1 \end{pmatrix}$$

Shear in OpenGL

No shear command in opengl.

Can use gl.glMultMatrixf to set up any matrix.

Matrices are in column major order.

Exercise

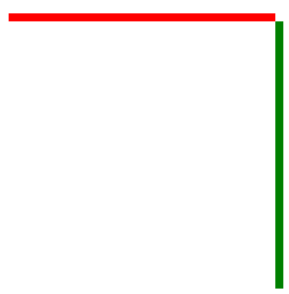
What would the matrix for scaling -1 in the x and y direction look like?

What would the matrix for rotating by 180 degrees look like?

Solution

What would the matrix for scaling -1 in the x and y direction look like?

$$\begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$



Solution

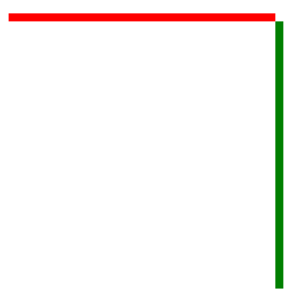
What would the matrix for rotating by 180 degrees look like?

$$\begin{pmatrix} \cos(180) & -\sin(180) & 0 \\ \sin(180) & \cos(180) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Solution

What would the matrix for rotating by 180 degrees look like?

$$\begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$



Composing transformations

We can combine a series of transformations by post-multiplying their matrices. The composition of two affine transformations is also affine.

Eg: Translate, then rotate, then scale:

$$\mathbf{M} = \mathbf{M_T}\mathbf{M_R}\mathbf{M_S}$$

$$Q = \mathbf{M}P = \mathbf{M_T}\mathbf{M_R}\mathbf{M_S}P$$

In OpenGL

```
gl.glMatrixMode(GL2.GL_MODELVIEW);

//Current Transform (CT) is the MODELVIEW //
Matrix

gl.glLoadIndentity();
//CT = identity matrix (I)

gl.glTranslated(dx, dy, 0);
//CT = IT

gl.glRotated(theta, 0, 0, 1);
//CT = ITR

gl.glScaled(sx, sy, 1);
//CT = ITRS
```

In OpenGL

```
gl.glBegin(GL2.GL_POINTS);
{
   gl.glVertex2d(px, py);
   //Point drawn at Q = CT P
   //               Q = ITRS P
}
gl.glEnd();
```

Exercise

What would the value of the current transform be after the following?

```
gl.glMatrixMode(GL2.GL_MODELVIEW);
gl.glLoadIdentity();
gl.glTranslated(1,2,0);
gl.glRotated(90,0,0,1);
```

Solution

CT = I

$$\begin{array}{c} I \\ \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \end{array}$$

Solution

CT = T

$$
\underset{I}{\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}}
\underset{T}{\begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix}}
=
\underset{T}{\begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix}}
$$

Solution

CT = TR

$$
\begin{array}{ccc}
T & R & = \quad TR \\
\begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix} &
\begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} &
\begin{pmatrix} 0 & -1 & 1 \\ 1 & 0 & 2 \\ 0 & 0 & 1 \end{pmatrix}
\end{array}
$$

Exercise

Suppose we continue from our last example and do the following

```
gl.glPushMatrix();

gl.glScaled(2,2,1);

//1. What is CT now?

gl.glPopMatrix();

//2. What is CT now?
```

Solution

CT = TR

$$\underset{\text{TR*}}{\begin{pmatrix} 0 & -1 & 1 \\ 1 & 0 & 2 \\ 0 & 0 & 1 \end{pmatrix}} \underset{\text{S}}{\begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix}} = \underset{\text{TRS}}{\begin{pmatrix} 0 & -2 & 1 \\ 2 & 0 & 2 \\ 0 & 0 & 1 \end{pmatrix}}$$

* TR has been pushed onto the stack

Solution

//1. What is CT now?

TRS

$$\begin{pmatrix} 0 & -2 & 1 \\ 2 & 0 & 2 \\ 0 & 0 & 1 \end{pmatrix}$$

Solution

//2. What is CT now?

It would be restored to the last matrix that got pushed on the stack

$$\begin{pmatrix} 0 & -1 & 1 \\ 1 & 0 & 2 \\ 0 & 0 & 1 \end{pmatrix}$$

Decomposing transformations

Every 2D affine transformation can be decomposed as:

$$\mathbf{M} = \mathbf{M}_{\mathrm{translate}}\mathbf{M}_{\mathrm{rotate}}\mathbf{M}_{\mathrm{scale}}\mathbf{M}_{\mathrm{shear}}$$

If scaling is always uniform in both axes, the shear term can be eliminated:

$$\mathbf{M} = \mathbf{M}_{\mathrm{translate}}\mathbf{M}_{\mathrm{rotate}}\mathbf{M}_{\mathrm{scale}}$$

Decomposing transformations

To decompose the transform, consider the matrix form:

$$\begin{pmatrix} i_1 & j_1 & \phi_1 \\ i_2 & j_2 & \phi_2 \\ 0 & 0 & 1 \end{pmatrix}$$

axes origin

Decomposing transformations

Assuming uniform scaling and no shear

$$\begin{aligned}\text{translation} &= (\phi_1, \phi_2, 1)^\top \\ \text{rotation} &= \arctan(i_1, i_2) \\ \mathit{scale} &= |\mathbf{i}|\end{aligned}$$

Note: arctan(i1,i2) is arctan(i2/i1) aka $\tan^{-1}$(i2/i1) adjusting for i1 being 0. If i1 == 0 (and i2 is not) we get 90 degrees if i2 is positive or -90 if i2 is negative.

Example

$$\begin{pmatrix} 0 & -2 & 1 \\ 2 & 0 & 2 \\ 0 & 0 & 1 \end{pmatrix}$$

Origin: (1,2)

i: (0,2)

j: (-2,0)

Translation: (1,2)

Rotation: arctan(0,2) = 90 degrees

Scale = |i| = |j| = 2

Also we can tell that axes are still

perpendicular as **i.j** = 0

Exercise

$$\begin{pmatrix} 1.414 & -1.414 & 0.5 \\ 14.414 & 1.414 & -2 \\ 0 & 0 & 1 \end{pmatrix}$$

What are the axes of the coordinate frame this matrix represents? What is the origin? Sketch it.

What is the scale of each axis?

What is the angle of each axis?

Are the axes perpendicular?

Solution

$$\begin{pmatrix} 1.414 & -1.414 & 0.5 \\ 1.414 & 1.414 & -2 \\ 0 & 0 & 1 \end{pmatrix}$$

Origin: 0.5,-2

i: 1.414,1.414

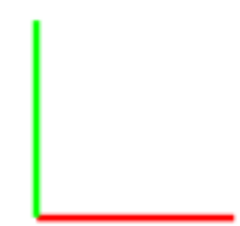
j: -1.414,1.414

Rotation: arctan(1.414,1.414)

= 45 degrees

Scale: |i| = 2

Also we can tell that axes are still perpendicular as **i.j** = 0



Inverse Transformations

If the local-to-global transformation is:

$$Q = \mathbf{M_T M_R M_S} P$$

then the global-to-local transformation is the inverse:

$$P = \mathbf{M_S^{-1} M_R^{-1} M_T^{-1}} Q$$

Inverse Transformations

Inverses are easy to compute:

$$\begin{aligned}
\text{translation:} \quad {\mathbf{M_T}}^{-1}(d_x, d_y) &= \mathbf{M_T}(-d_x, -d_y) \\
\text{rotation:} \quad {\mathbf{M_R}}^{-1}(\theta) &= \mathbf{M_R}(-\theta) \\
\text{scale:} \quad {\mathbf{M_S}}^{-1}(s_x, s_y) &= \mathbf{M_S}(1/s_x, 1/s_y) \\
\text{shear:} \quad {\mathbf{M_H}}^{-1}(h) &= \mathbf{M_H}(-h)
\end{aligned}$$

Local to World Exercise

Suppose the following transformations had been applied:

gl.glTranslated(3,2,0);

gl.glRotated(-45,0,0,1);

gl.glScaled(0.5,0.5,1);

What point in the local co-ordinate frame would correspond to the world co-ordinate Q (2,-1)?

Solution

$$P(\text{local}) = M^{-1}\, Q(\text{world})$$

$$= S^{-1}R^{-1}T^{-1}\, Q$$

$$\underset{S^{-1}}{\begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix}} \underset{R^{-1}}{\begin{pmatrix} 0.7 & -0.7 & 0 \\ 0.7 & 0.7 & 0 \\ 0 & 0 & 1 \end{pmatrix}} = \underset{S^{-1}R^{-1}}{\begin{pmatrix} 1.4 & -1.4 & 0 \\ 1.4 & 1.4 & 0 \\ 0 & 0 & 1 \end{pmatrix}}$$

Solution

P(local) = M-1 Q(world)

$= S^{-1}R^{-1}T^{-1}\ Q$

$$\underbrace{\begin{pmatrix} 1.4 & -1.4 & 0 \\ 1.4 & 1.4 & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{S^{-1}R^{-1}} \underbrace{\begin{pmatrix} 1 & 0 & -3 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{pmatrix}}_{T^{-1}} = \underbrace{\begin{pmatrix} 1.4 & -1.4 & -1.4 \\ 1.4 & 1.4 & -7 \\ 0 & 0 & 1 \end{pmatrix}}_{S^{-1}R^{-1}T^{-1}}$$

Solution

P(local) = M-1 Q(world)

$= S^{-1}R^{-1}T^{-1}\ Q$

$$\underset{S^{-1}R^{-1}T^{-1}}{\begin{pmatrix} 1.4 & -1.4 & -1.4 \\ 1.4 & 1.4 & -7 \\ 0 & 0 & 1 \end{pmatrix}} \underset{Q}{\begin{pmatrix} 2 \\ -1 \\ 1 \end{pmatrix}} = \underset{P}{\begin{pmatrix} 2.8 \\ -5.6 \\ 1 \end{pmatrix}}$$

Assignment

```
            ROOT
          /       \
     Table        Lego Man
    /                  \
Cup                     Lego Man Hand
```

Assignment Change Parent

```
        ROOT
      /        \
Table        Lego Man
                 \
           Lego Man Hand
                    \
                    Cup
```

Lerping

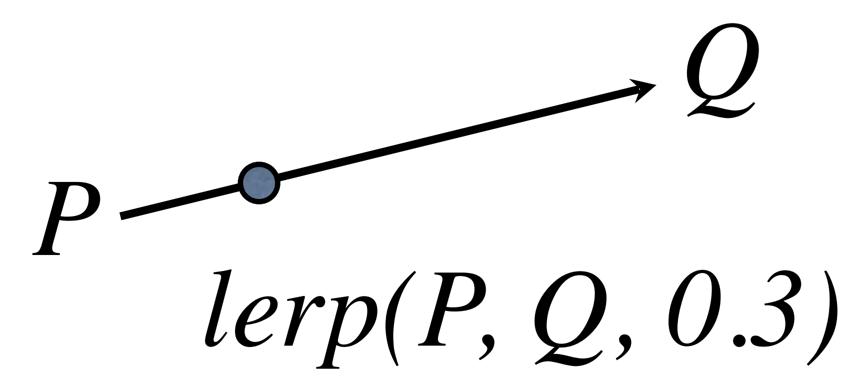
We can add affine combinations of points:

$$\frac{1}{2}(p_1, p_2, 1)^\top + \frac{1}{2}(q_1, q_2, 1)^\top = (\frac{p_1 + q_1}{2}, \frac{p_2 + q_2}{2}, \mathbf{1})^\top$$

We often use this to do linear interpolation between points:

$$lerp(P, Q, t) = P + t(Q - P)$$

$$lerp(P,Q,t) = P(1-t) + tQ$$



Lerping Exercise

Using linear interpolation, what is the midpoint between P(4,9) and B=(3,7).

Lerping Solution

Using linear interpolation, what is the midpoint between P(4,9) and B=(3,7).

Would be at t = 0.5 so

$$\begin{aligned} \mathrm{lerp}(P,B,t) &= (4,9)(1 - 0.5) + (0.5)(3,7) \\ &= (2,4.5) + (1.5,3.5) \\ &= (3.5,8) \end{aligned}$$

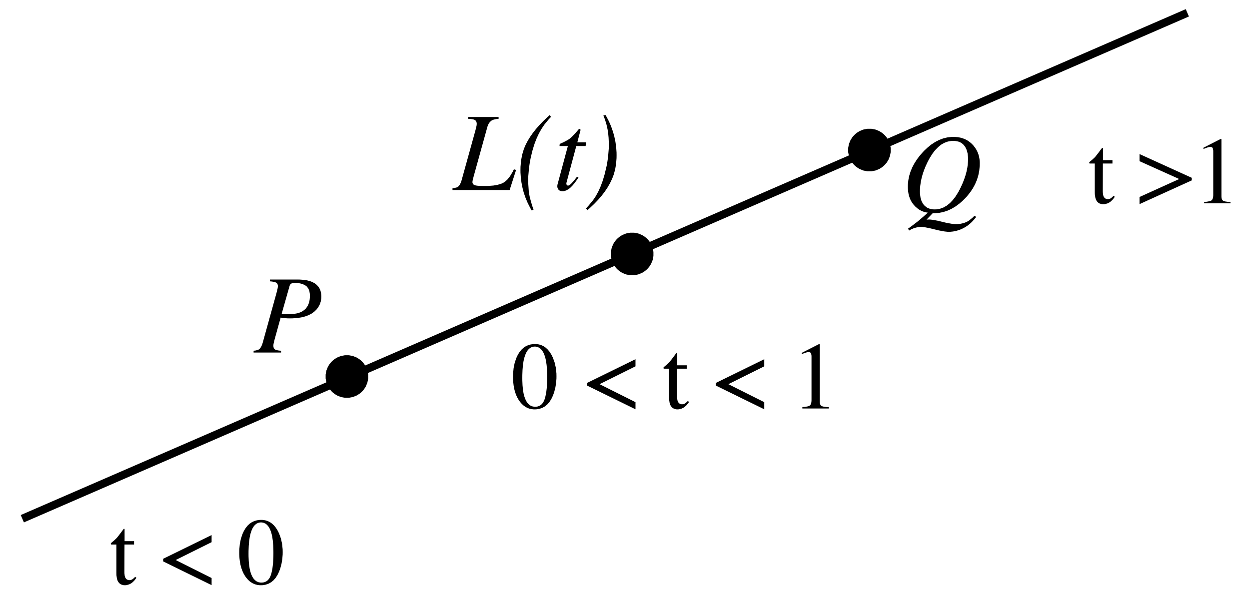
Lines

Parametric form:

L(t) = P + t**v**

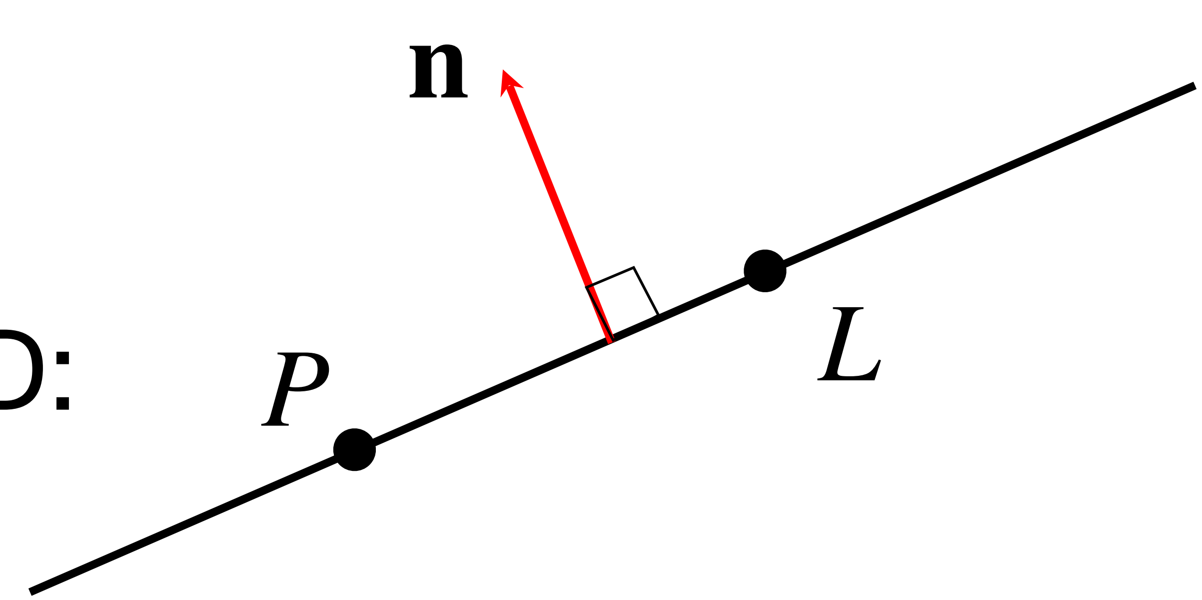
v = Q - P

$$L(t) = P + t(Q - P)$$



Point-normal form in 2D:

$$\mathbf{n} \cdot (P - L) = 0$$



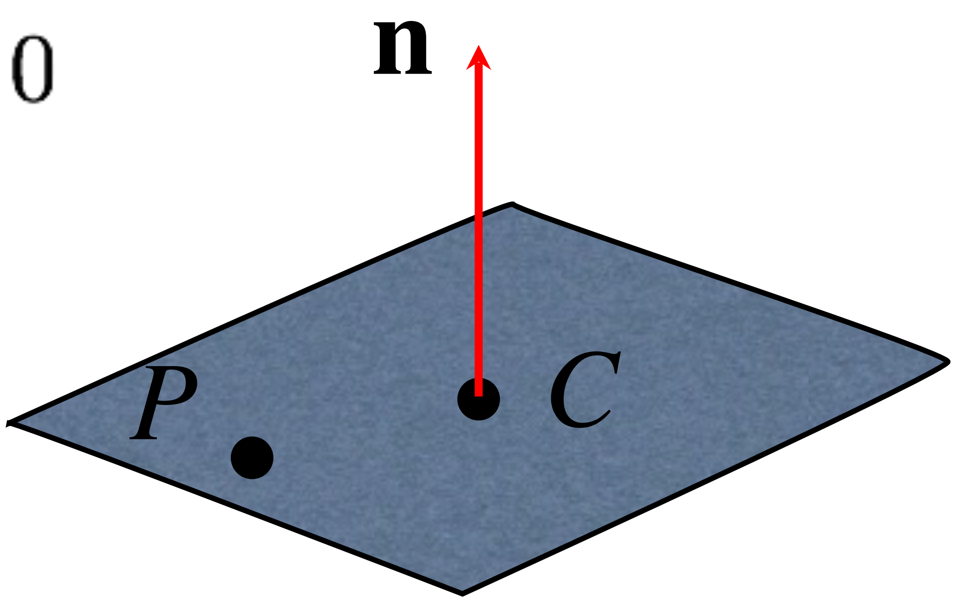
Planes in 3D

Parametric form:

$$P(s,t) = C + s\mathbf{a} + t\mathbf{b}$$

Point-normal form:

$$\mathbf{n} \cdot (P - C) = 0$$



Line intersection

Two lines

$$L_{AB}(t) = A + (B - A)t$$
$$L_{CD}(u) = C + (D - C)u$$

Solve simultaneous equations:

$$(B - A)t = (C - A) + (D - C)u$$

Line Intersection Example

A = (0,3) B = (12,7)

$$L_{AB}(t) = A + (B - A)t$$

$$L_{CD}(u) = C + (D - C)u$$

C = (2,0) D = (7,20)

$L_{AB}(t) = (0,3) + (12-0,7-3)t = (0,3) + (12,4)t$

$L_{CD}(u) = (2,0) + (7-2,20-0)u = (2,0) + (5,20)u$

Intersect for values of t and u where

$L_{AB}(t) = L_{CD}(u)$

Line Intersection Example…

$(0,3) + (12,4)t = (2,0) + (5,20)u$

In 2D that is 2 equations, one for x and y

$0 + 12t = 2 + 5u$

$3 + 4t = 0 + 20u$

Solve for t and u: $t = 0.25$, $u = 0.2$

Substitute into either line equation to get intersection at (3,4)

Line Intersection Example 2

Find where the L(t) = A + **c**t intersects with the line **n**.(P-B) = 0 where

A(2,3), **c** = (4,-4), **n** = (6,8), B=(7,7)

(6,8).((A + **c**t) – (7,7)) = 0

(6,8).((2,3) + (4,-4) t – (7,7)) = 0

(6,8).(2+4t-7, 3-4t-7) = 0

(6,8).(-5+4t, -4-4t) = 0

Line Intersection …

$(6,8).(-5+4t,-4-4t) = 0$

$6(-5+4t) + 8(-4-4t) = 0$

$-30 + 24t - 32 - 32t = 0$

$t = 62/-8 = -7.75$

$P = A + \mathbf{c}t = (2, 3) + (4, -4)*(-7.75)$

$= (-29, 34)$

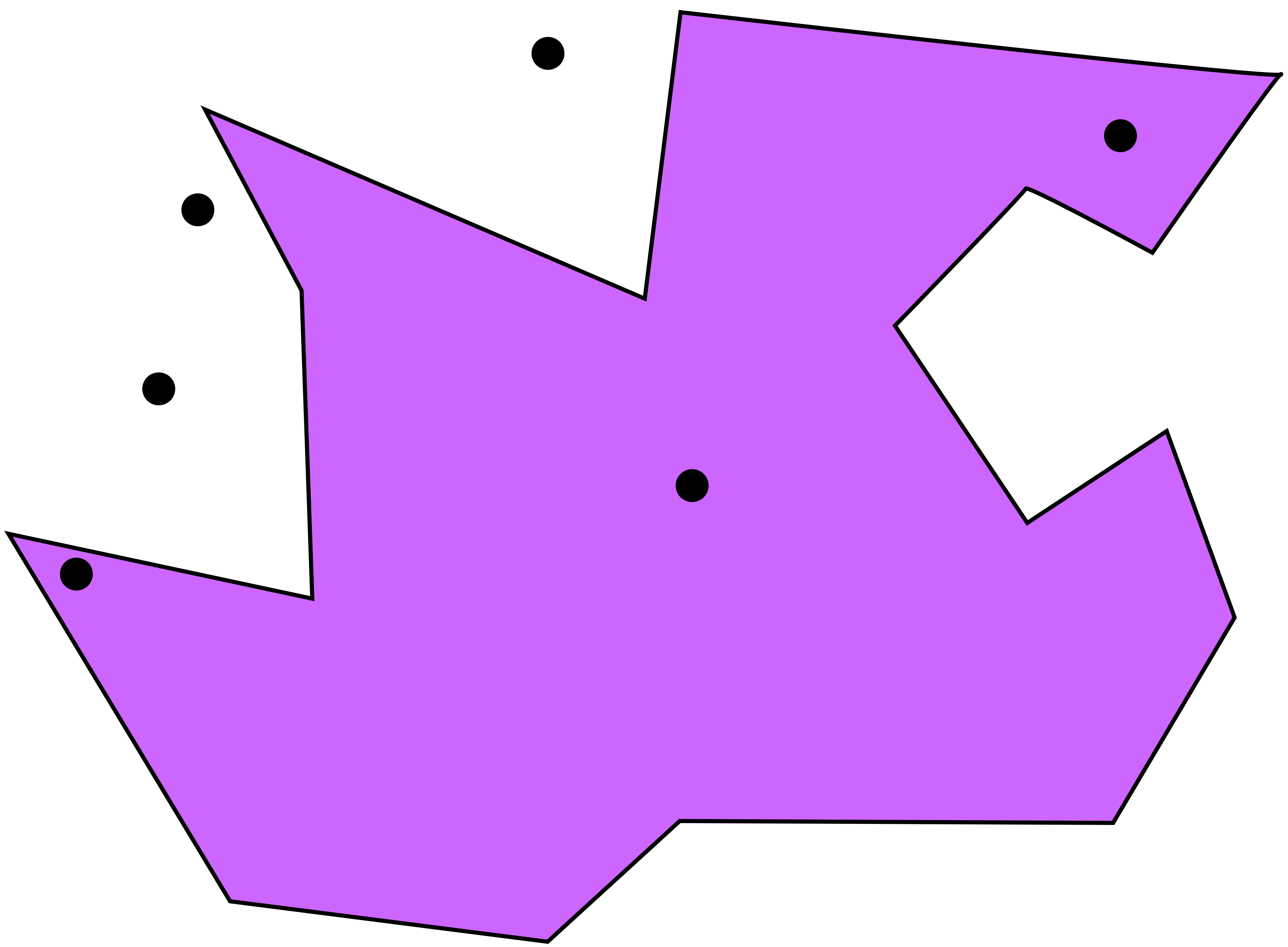
Point in Polygon

For any ray from the point

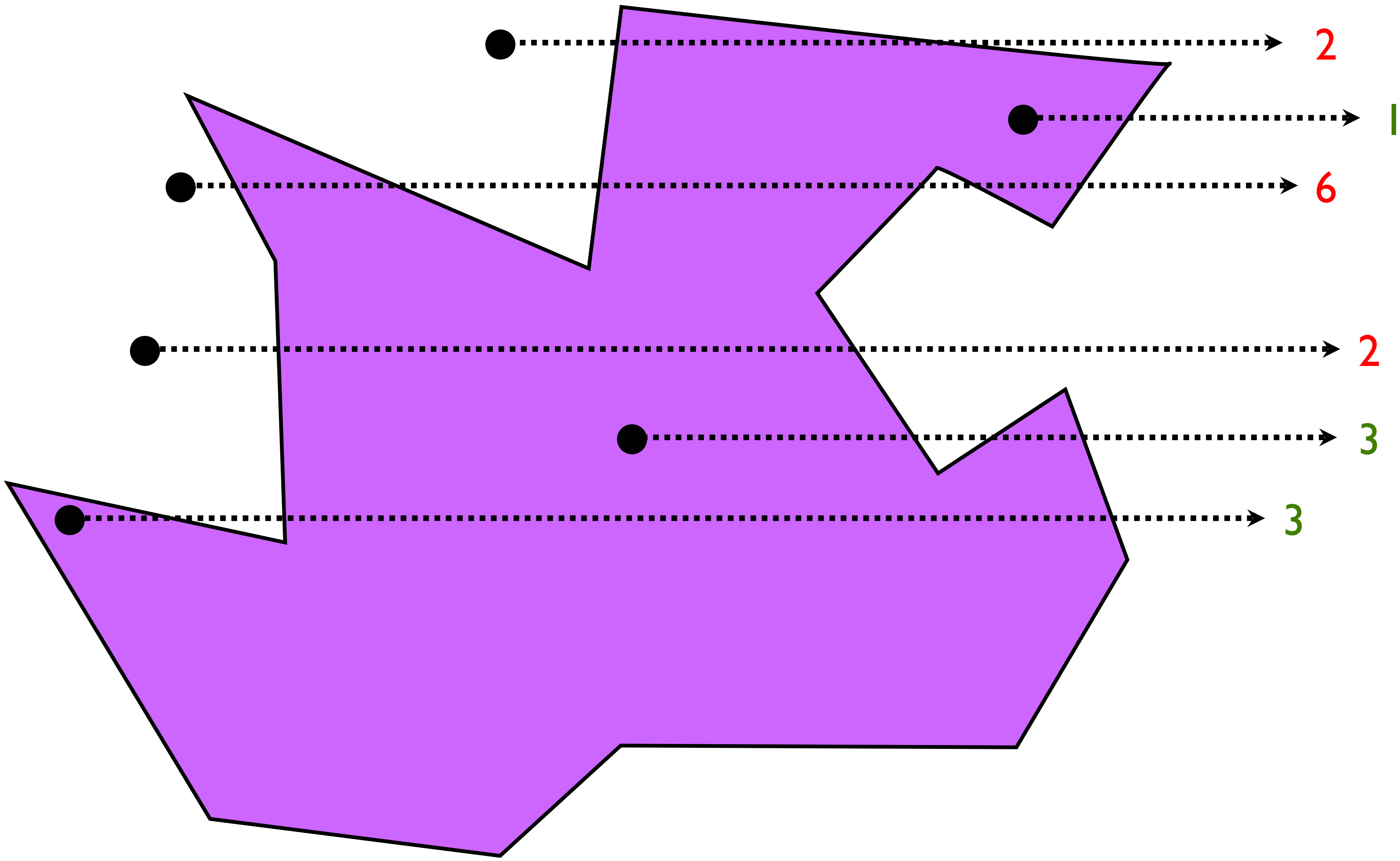
Count the number of crossings with the polygon

If there is an odd number of crossings the point is inside

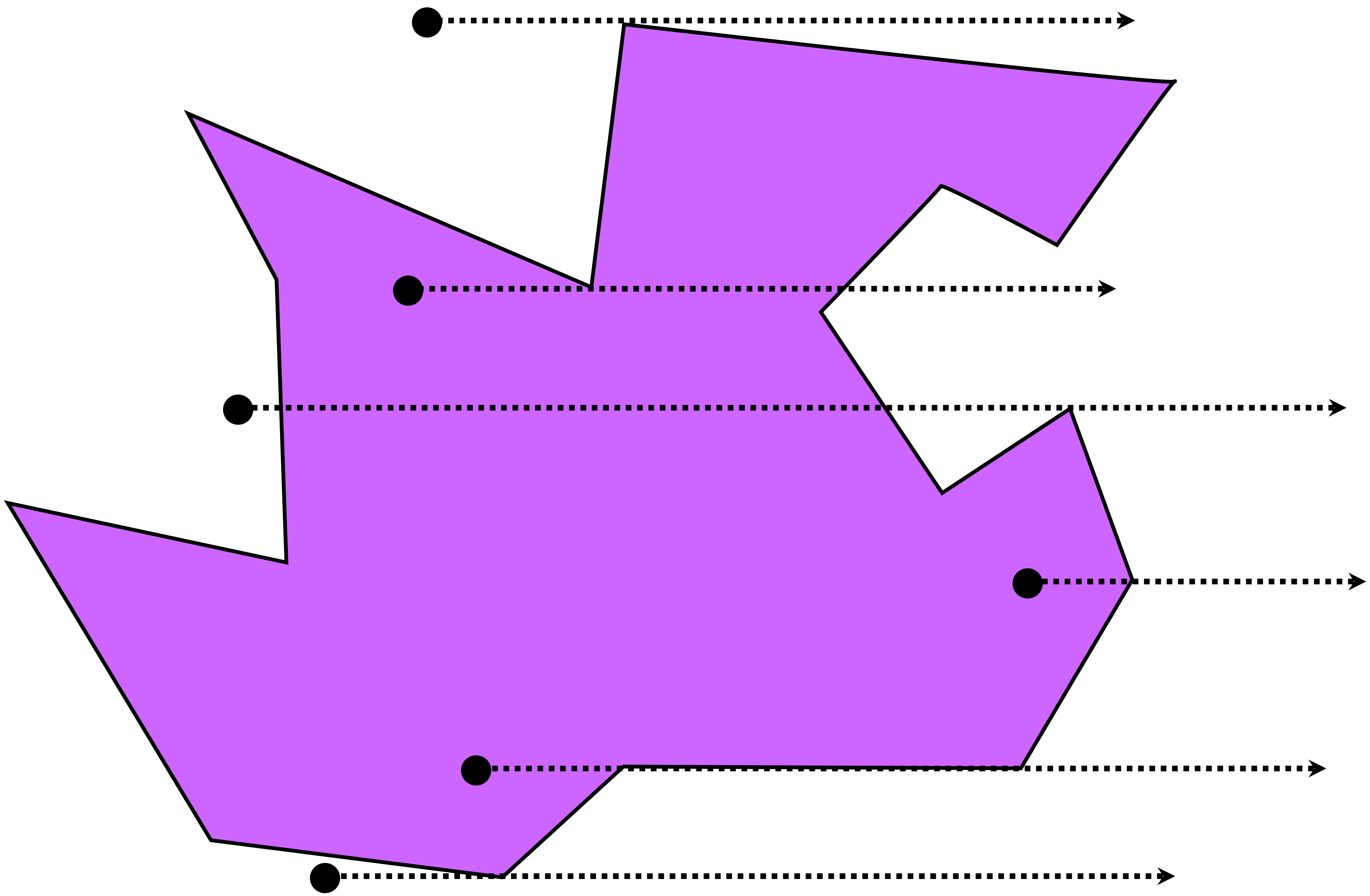
Point in polygon



Point in polygon

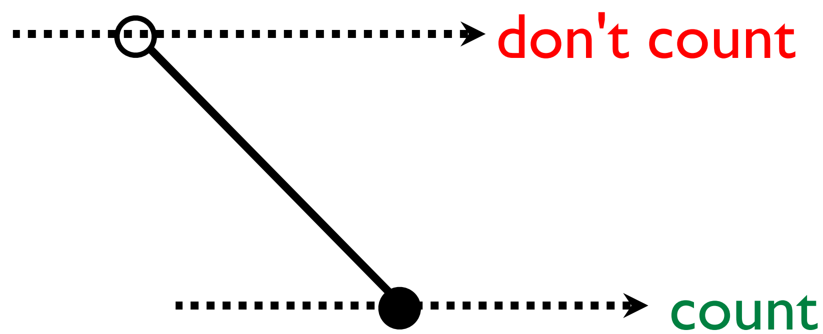


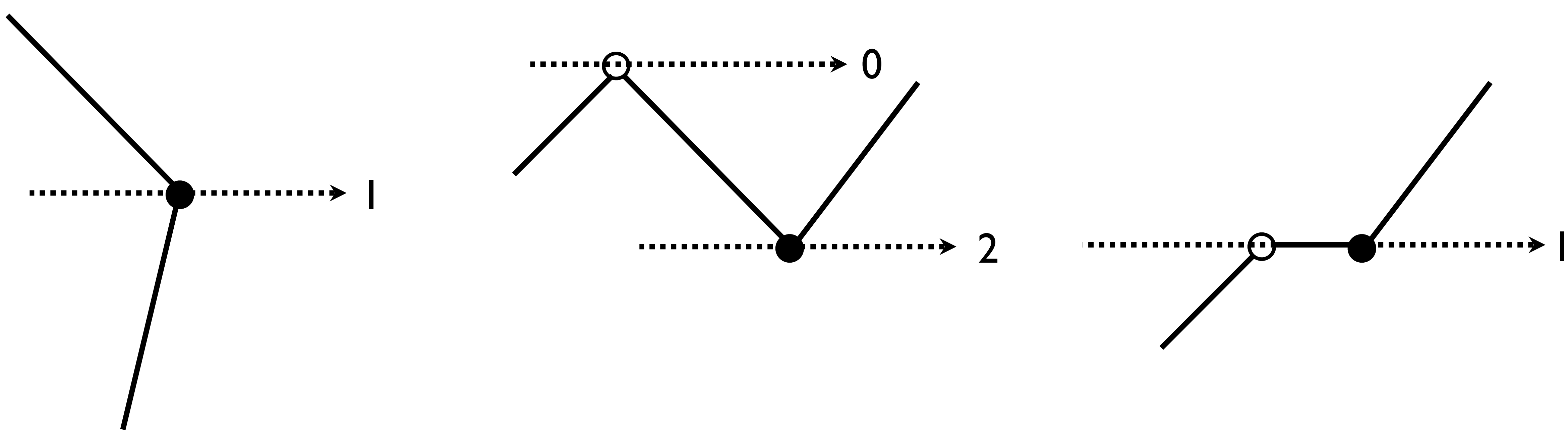
Difficult points



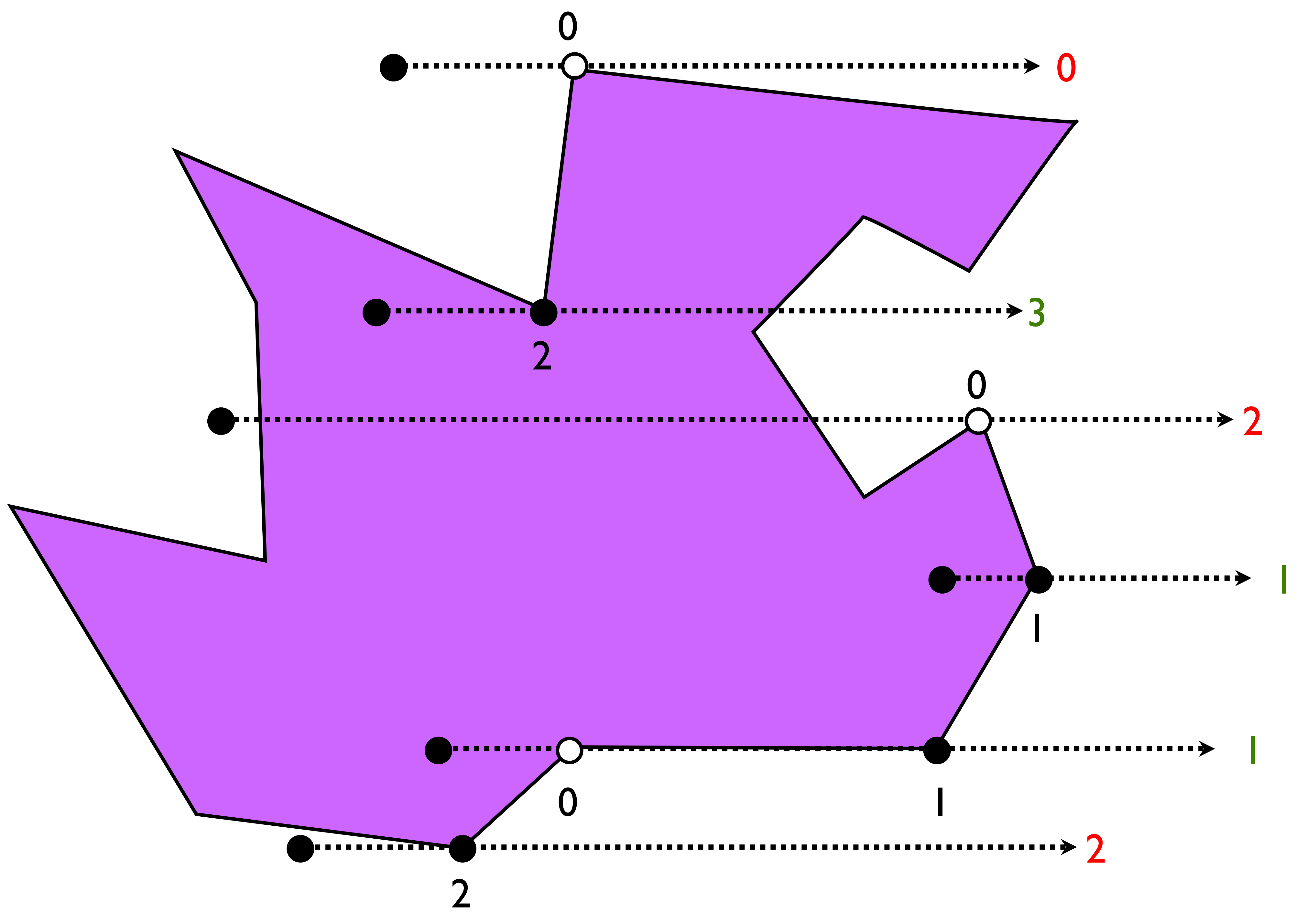
Solution

Only count crossings at the lower vertex of an edge.





Point in polygon



Computational Geometry

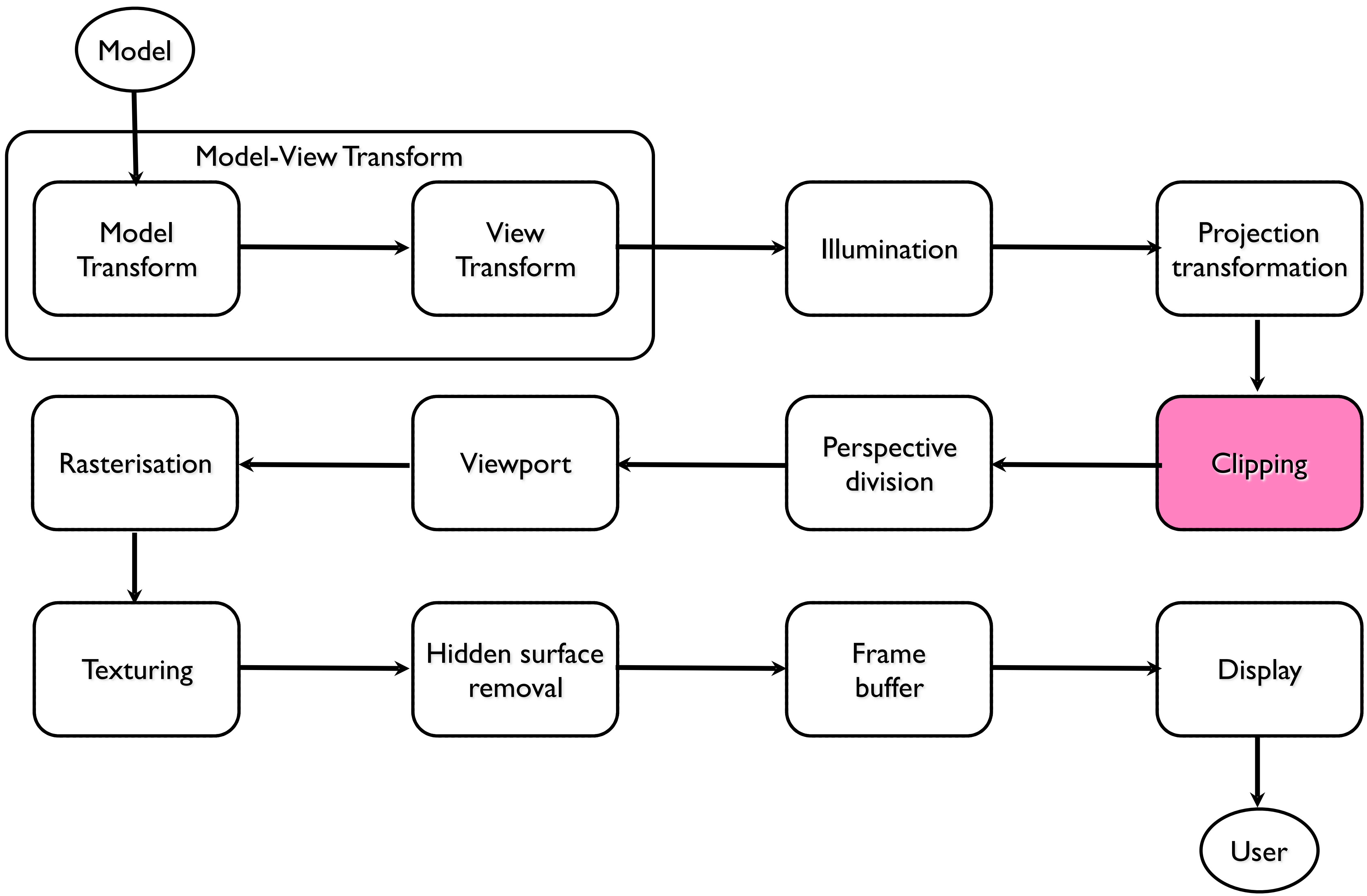
Computational Geometry in C, O'Rourke

http://cs.smith.edu/~orourke/books/compgeom.html

CGAL
Computational Geometry Algorithms Library

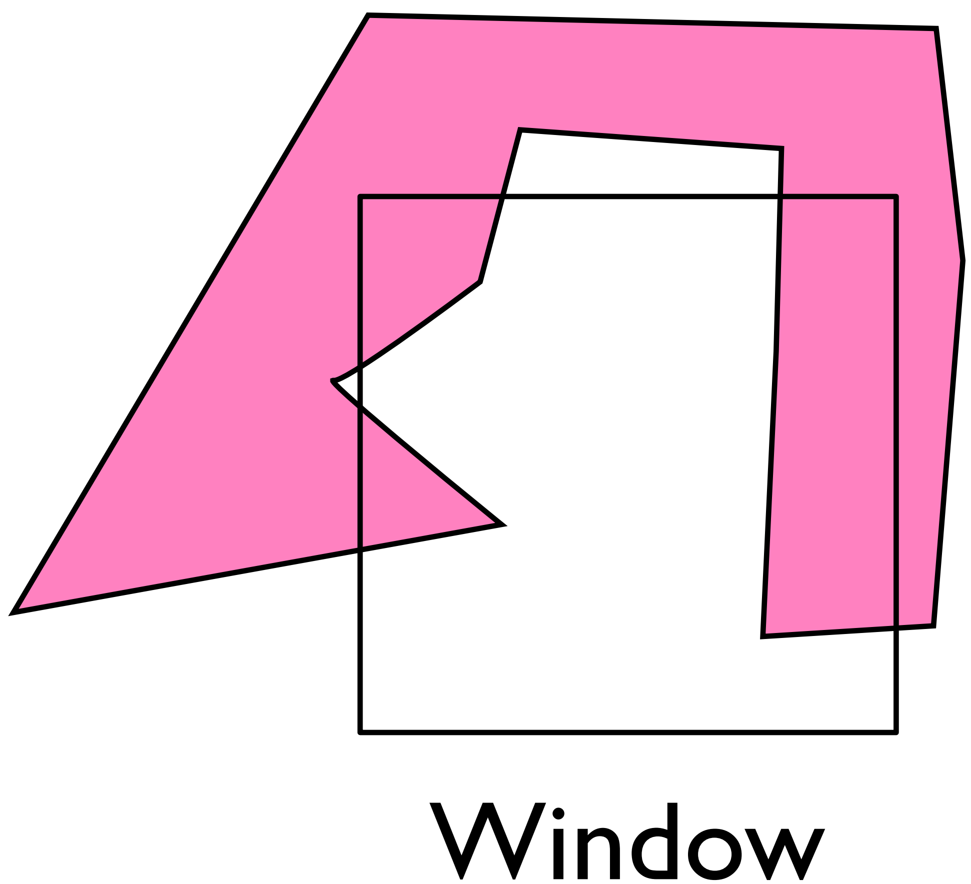
http://cgal.org/

The graphics pipeline



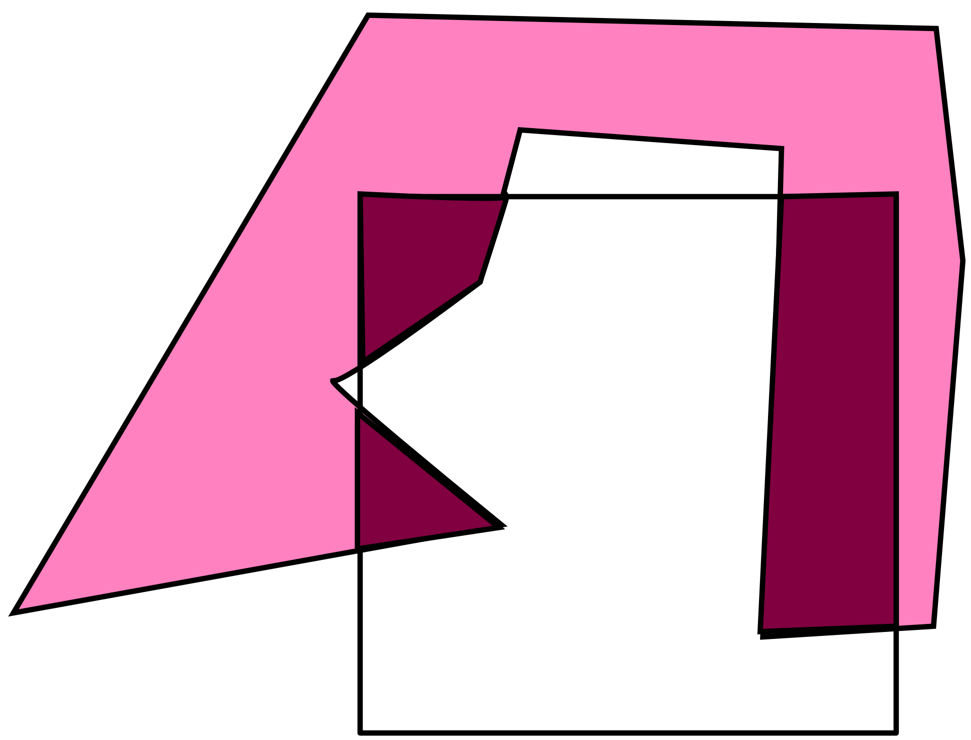
Clipping

The world is often much bigger than the camera window. We only want to render the parts we can see.



Clipping

The world is often much bigger than the camera window. We only want to render the parts we can see.



Window

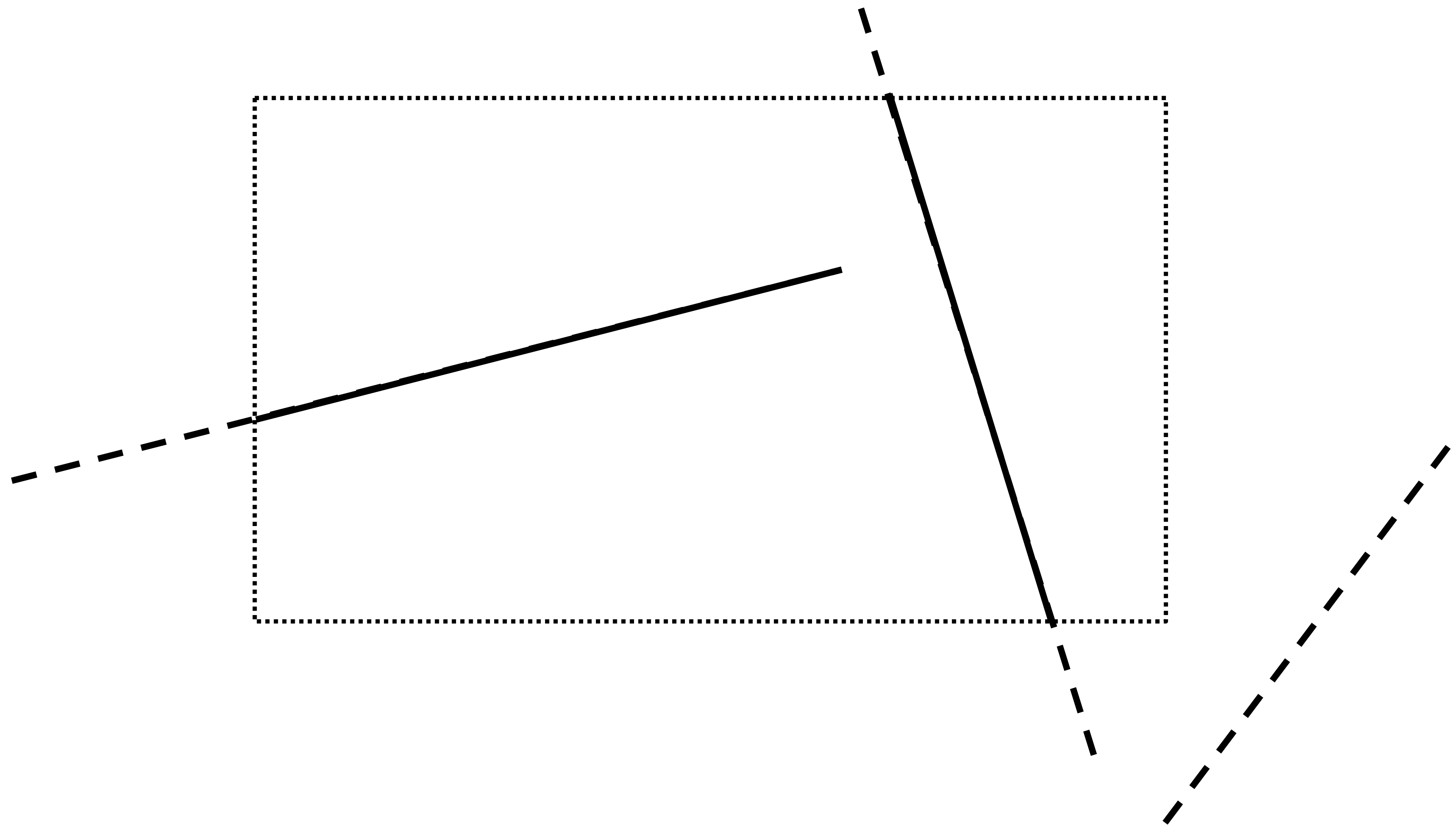
Clipping algorithms

There are a number of different clipping algorithms:

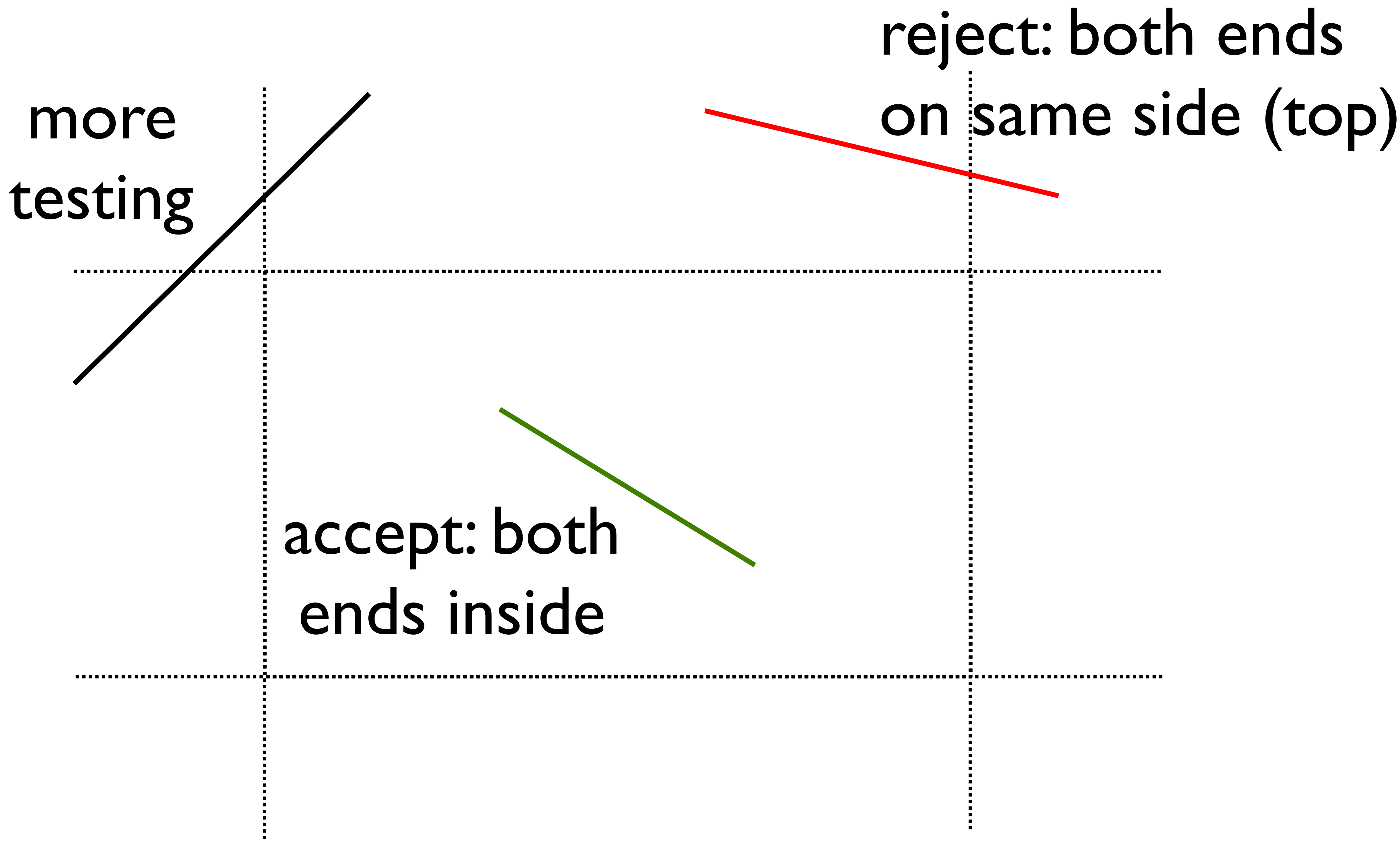
- Cohen-Sutherland (line vs rect)
- Cyrus-Beck (line vs convex poly)
- Sutherland-Hodgman
  (poly vs convex poly)
- Weiler-Atherton (poly vs poly)

Cohen-Sutherland

Clipping lines to an axis-aligned rectangle.



Trivial accept/reject



Labelling

1100	0100	0110
1000	0000	0010
1001	0001	0011

Label ends

```
Outcode(x, y):

 code = 0;
 if (x < left)    code |= 8;

 if (y > top)     code |= 4;

 if (x > right)   code |= 2;

 if (y < bottom)  code |= 1;
 return code;
```

Clip Once

```
ClipOnce(px, py, qx, qy):

 p = Outcode(px, py);
 q = Outcode(qx, qy);

 if (p == 0 && q == 0) {
     // trivial accept
 }

 if (p & q != 0) {
     // trivial reject
 }
```

Clip Once

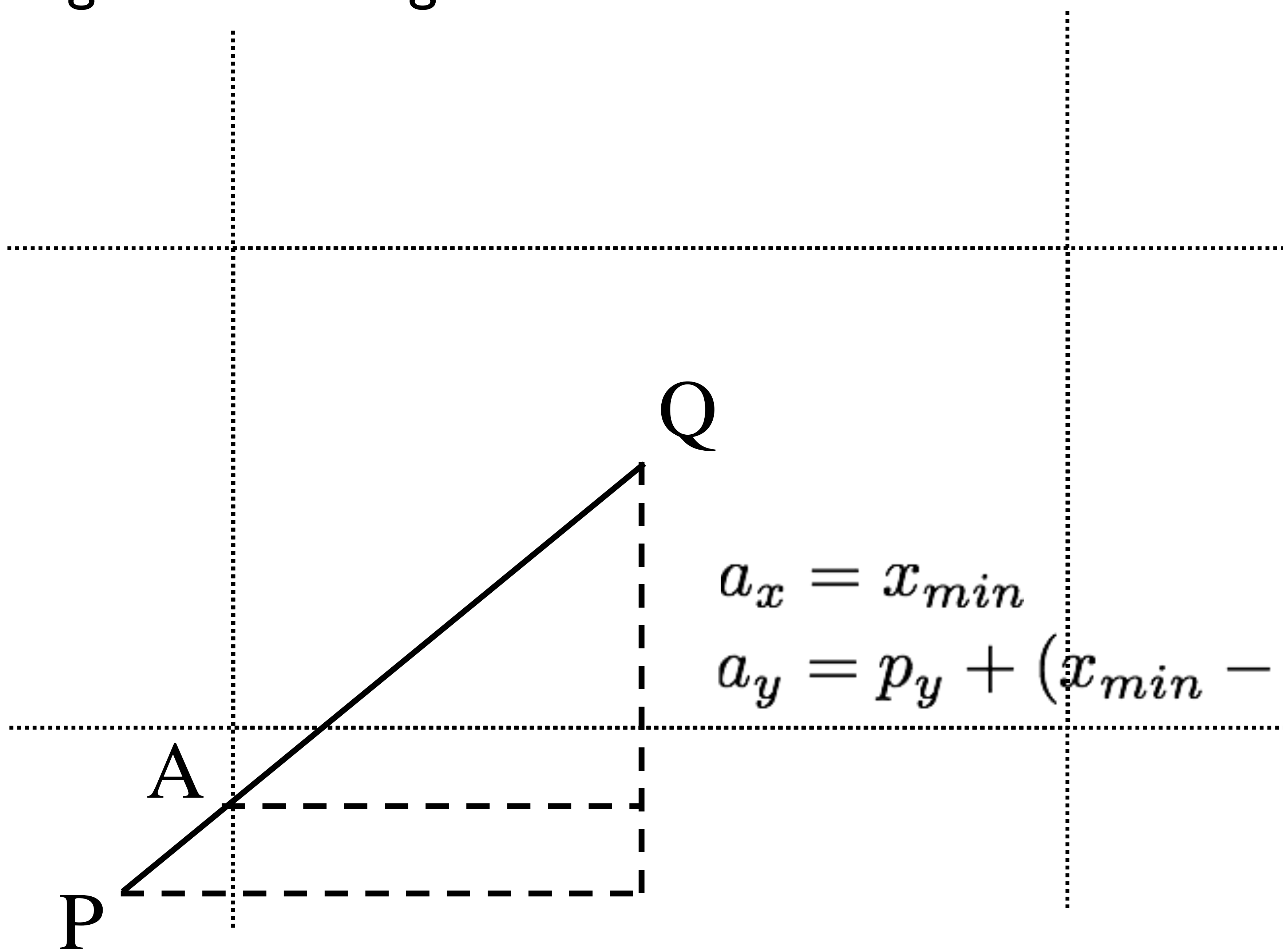
```
// cont...

if (p != 0) {
    // p is outside, clip it

}
else {
    // q is outside, clip it
}
```

Clip Loop

```
Clip(px, py, qx, qy):
  accept = false;
  reject = false;
  while (!accept && !reject):
    ClipOnce(px, py, qx, qy)
```

Clipping a point

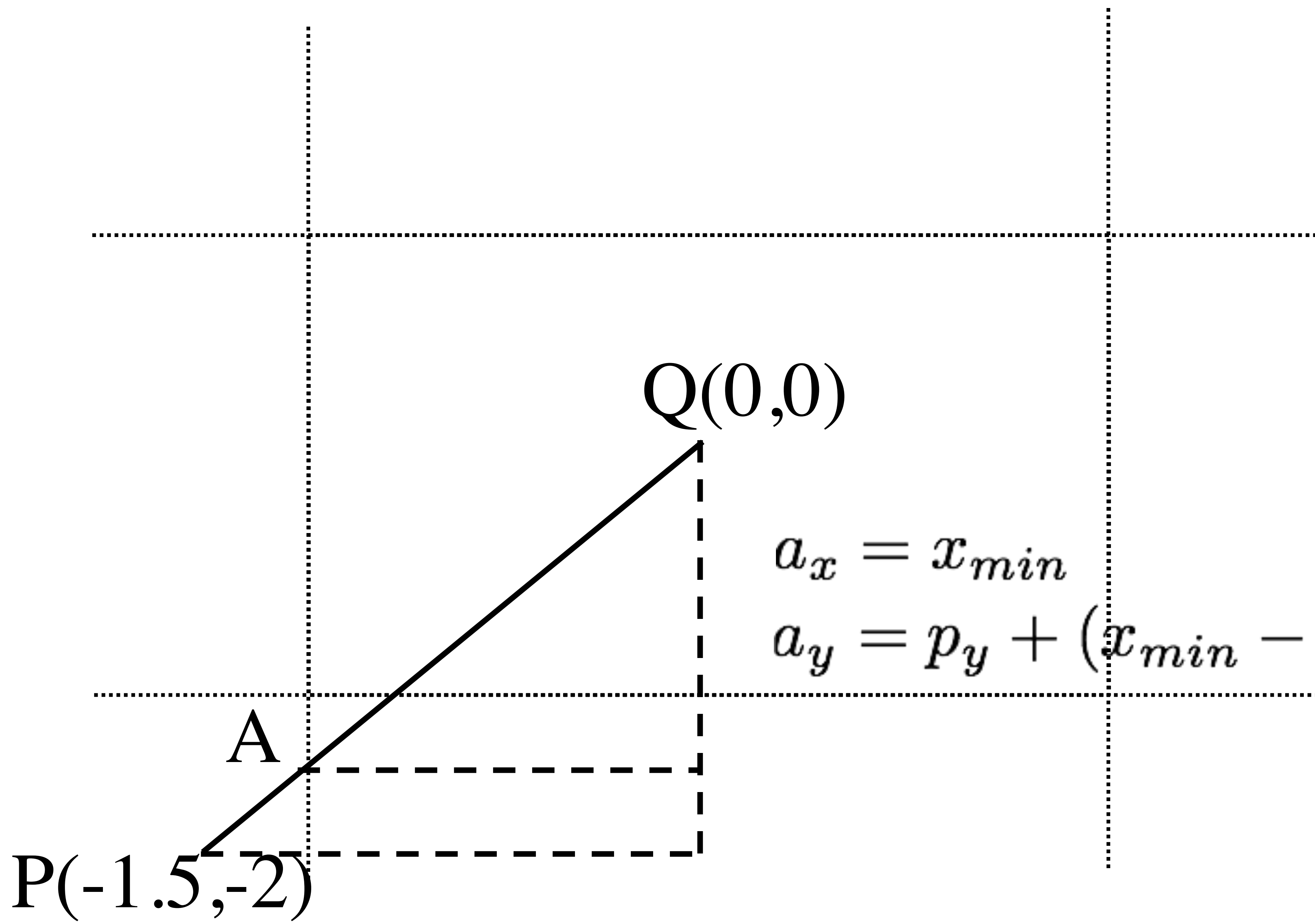
Using similar triangles:



$$a_x = x_{min}$$
$$a_y = p_y + (x_{min} - p_x)\frac{q_y - p_y}{q_x - p_x}$$

Clipping a point

Assume bottom left of clipping rectangle is (-1,-1)



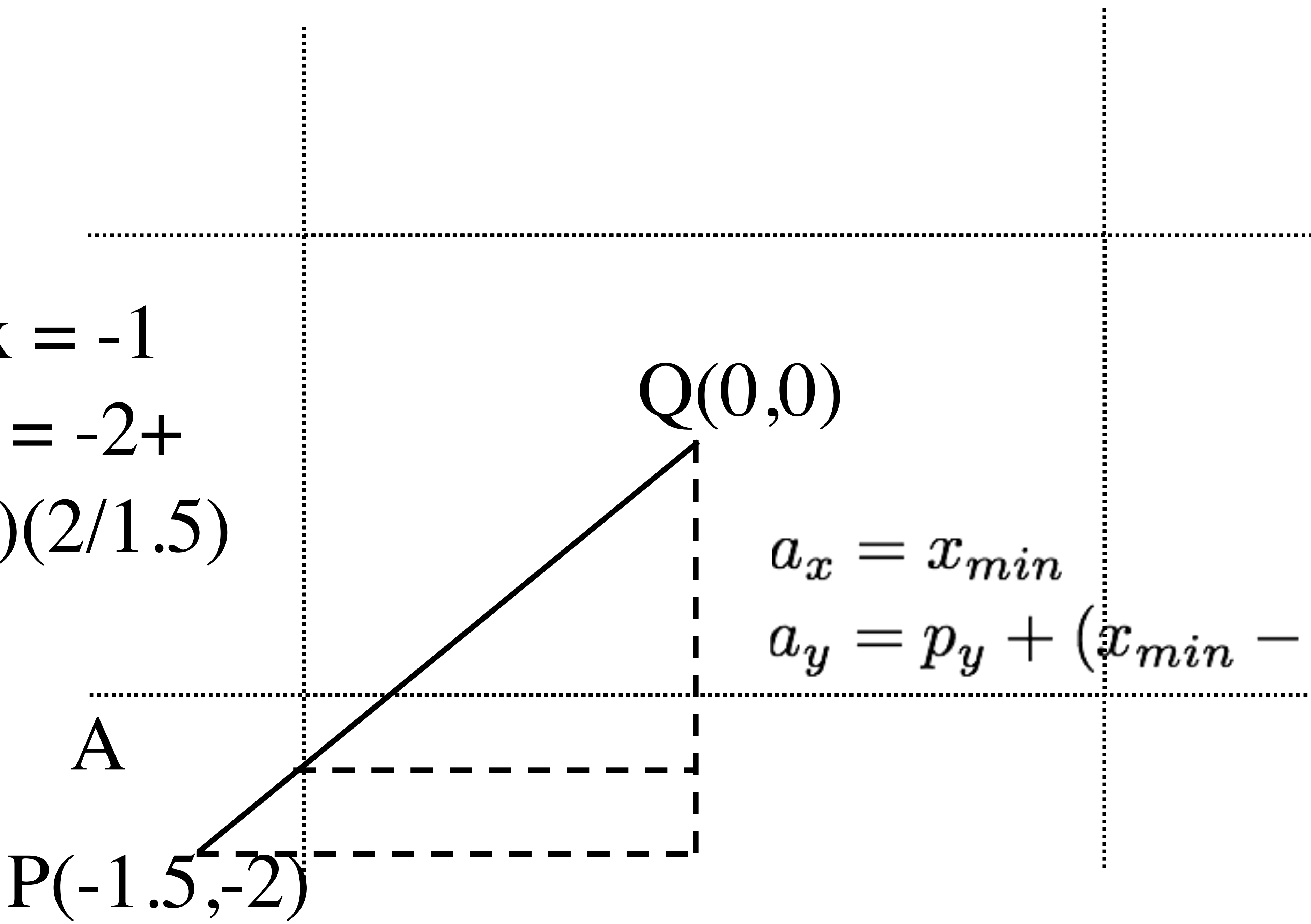
$$a_x = x_{min}$$
$$a_y = p_y + (x_{min} - p_x)\frac{q_y - p_y}{q_x - p_x}$$

Clipping a point

Assume bottom left of clipping rectangle is (-1,-1)

ax = -1

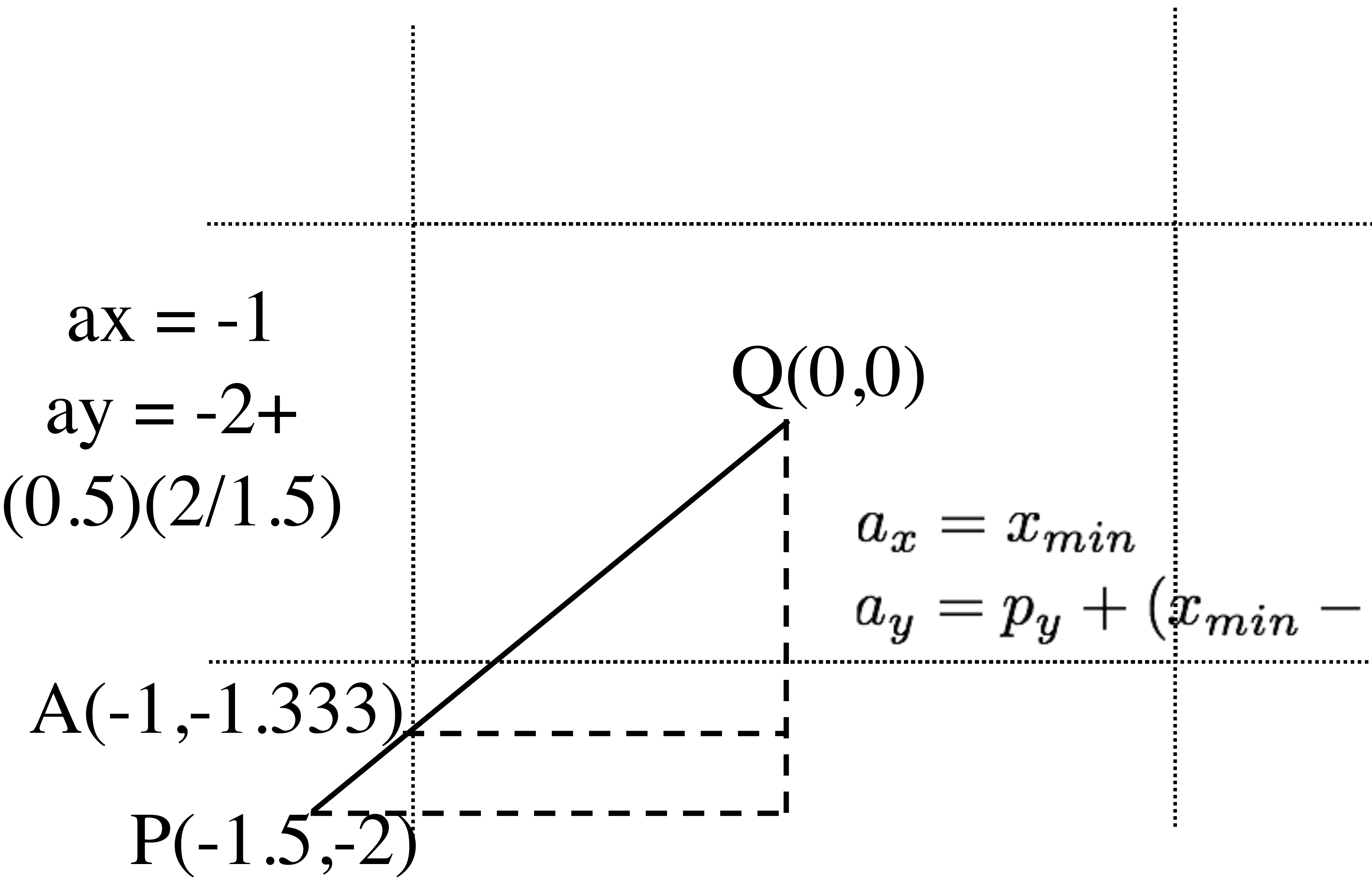
ay = -2+ (0.5)(2/1.5)



$$a_x = x_{min}$$
$$a_y = p_y + (x_{min} - p_x)\frac{q_y - p_y}{q_x - p_x}$$

Clipping a point

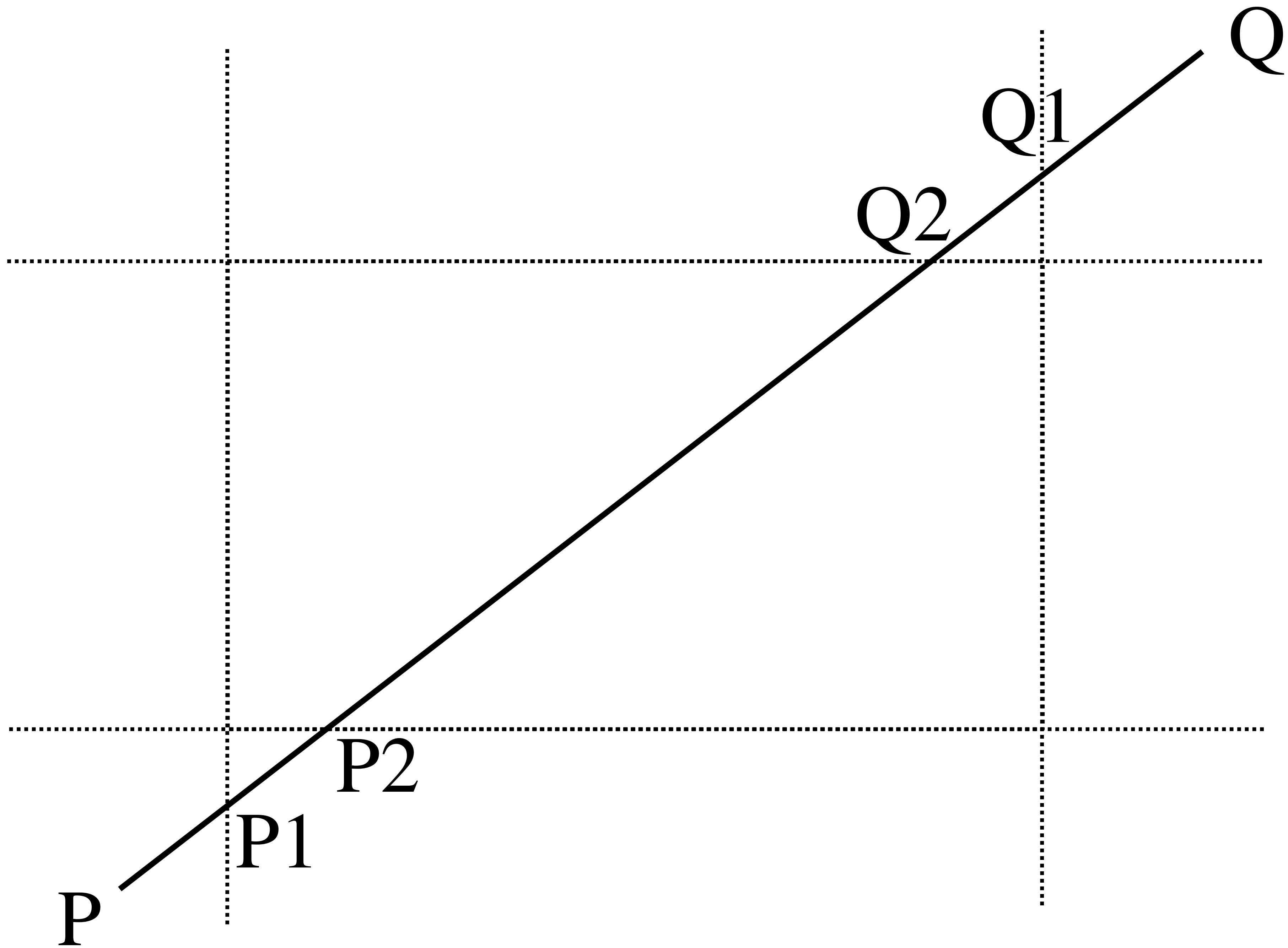
Assume bottom left of clipping rectangle is (-1,-1)



ax = -1
ay = -2+
(0.5)(2/1.5)

$$a_x = x_{min}$$
$$a_y = p_y + (x_{min} - p_x)\frac{q_y - p_y}{q_x - p_x}$$

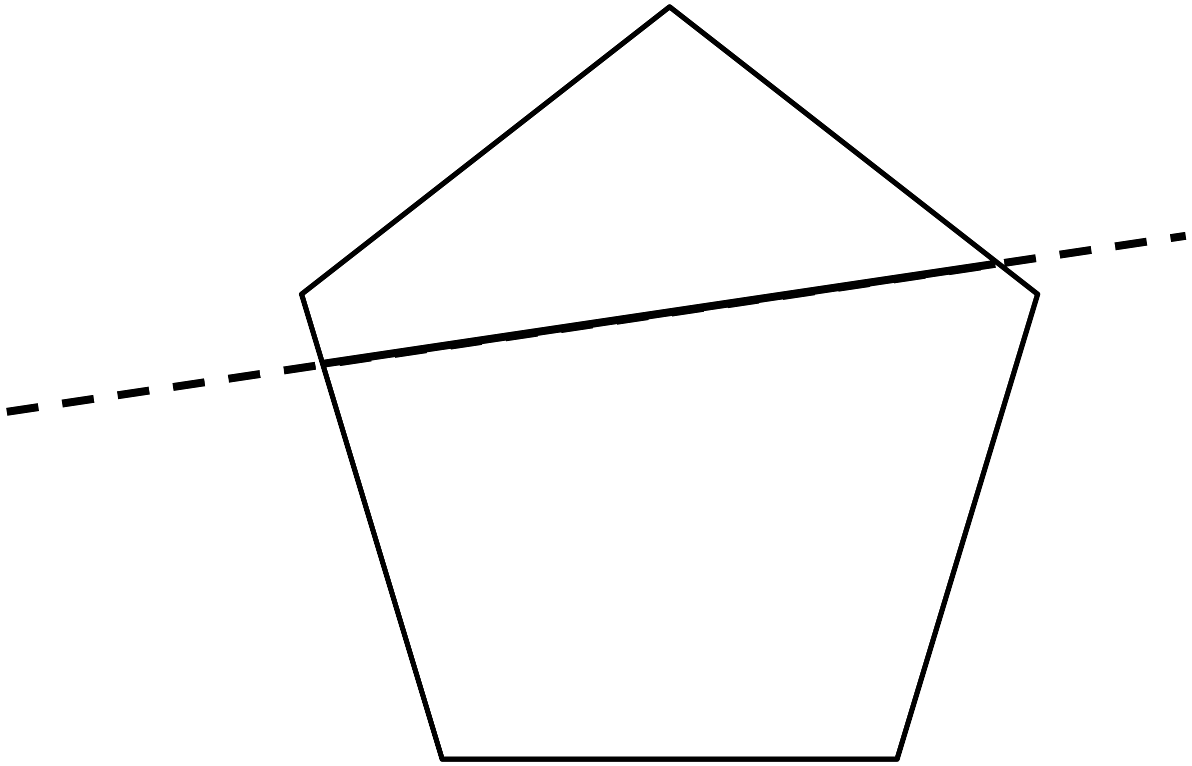
Case needing 4 Clips



Assignment 1 testing

Cyrus Beck

Clipping a line to a convex polygon.



Ray colliding with segment

Parametric ray:

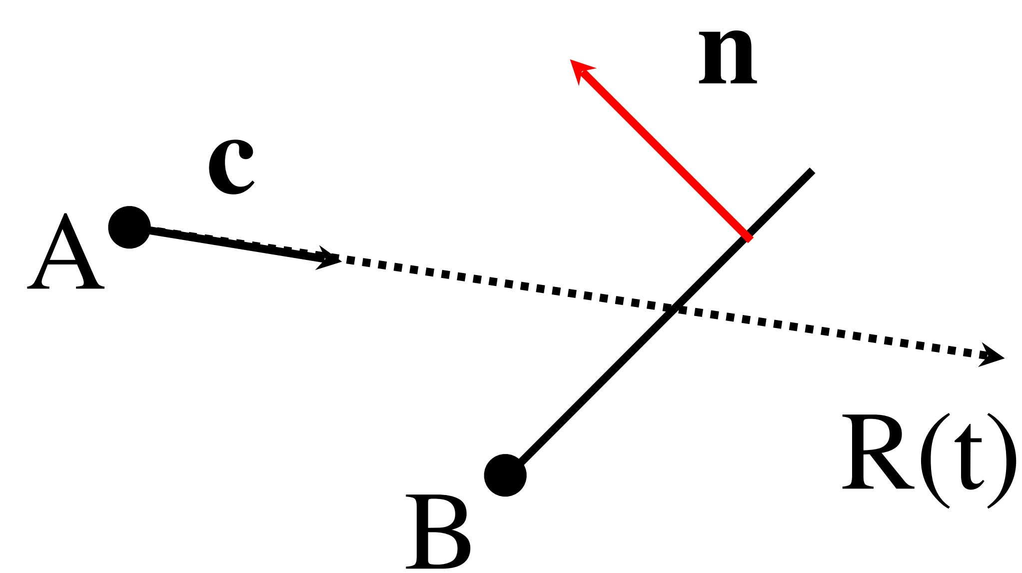
$$R(t) = A + \mathbf{c}t$$

Point normal segment:

$$\mathbf{n} \cdot (P - B) = 0$$

Collide when:

$$\mathbf{n} \cdot (R(t_{hit}) - B) = 0$$

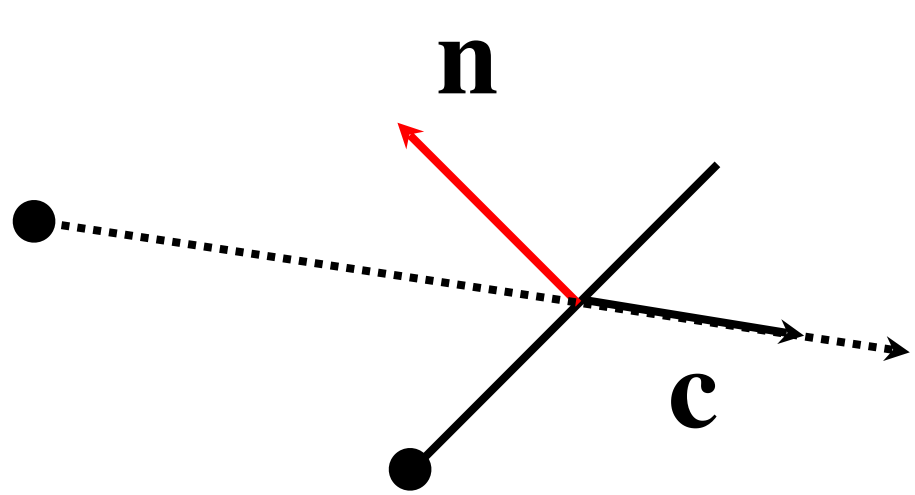


Hit time / point

$$\mathbf{n} \cdot (R(t_{hit}) - B) = 0$$
$$\mathbf{n} \cdot (A + \mathbf{c}t_{hit} - B) = 0$$
$$\mathbf{n} \cdot (A - B) + \mathbf{n} \cdot \mathbf{c}t_{hit} = 0$$
$$t_{hit} = \frac{\mathbf{n} \cdot (B - A)}{\mathbf{n} \cdot \mathbf{c}}$$
$$P_{hit} = A + \mathbf{c}t_{hit}$$

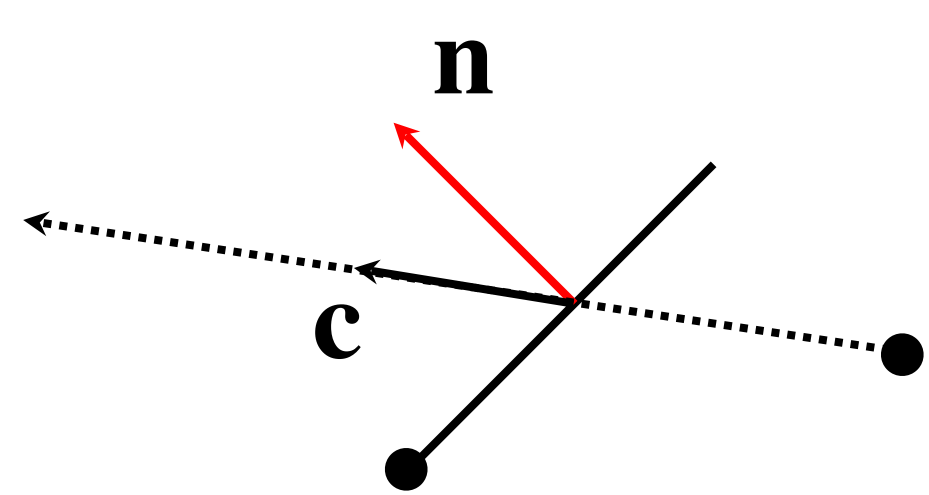
Entering / exiting

Assuming all normals point out of the polygon:



$\mathbf{n} \cdot \mathbf{c} < 0$

entering



$\mathbf{n} \cdot \mathbf{c} > 0$

exiting

Cyrus-Beck

Initialise t_{in} to 0 and t_{out} to 1

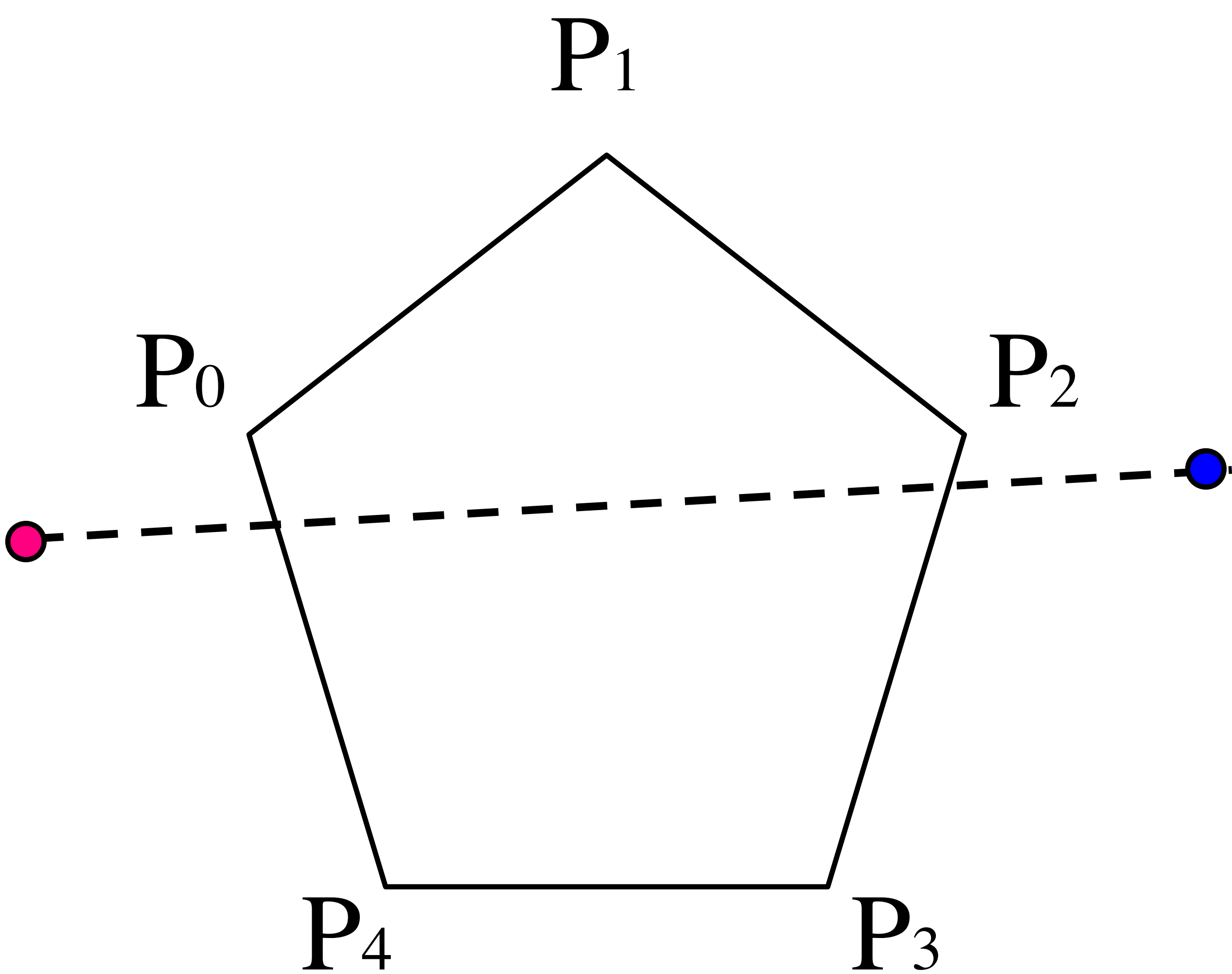
Compare the ray to each edge of the (convex) polygon.

Compute t_{hit} for each edge.

Keep track of maximum t_{in}

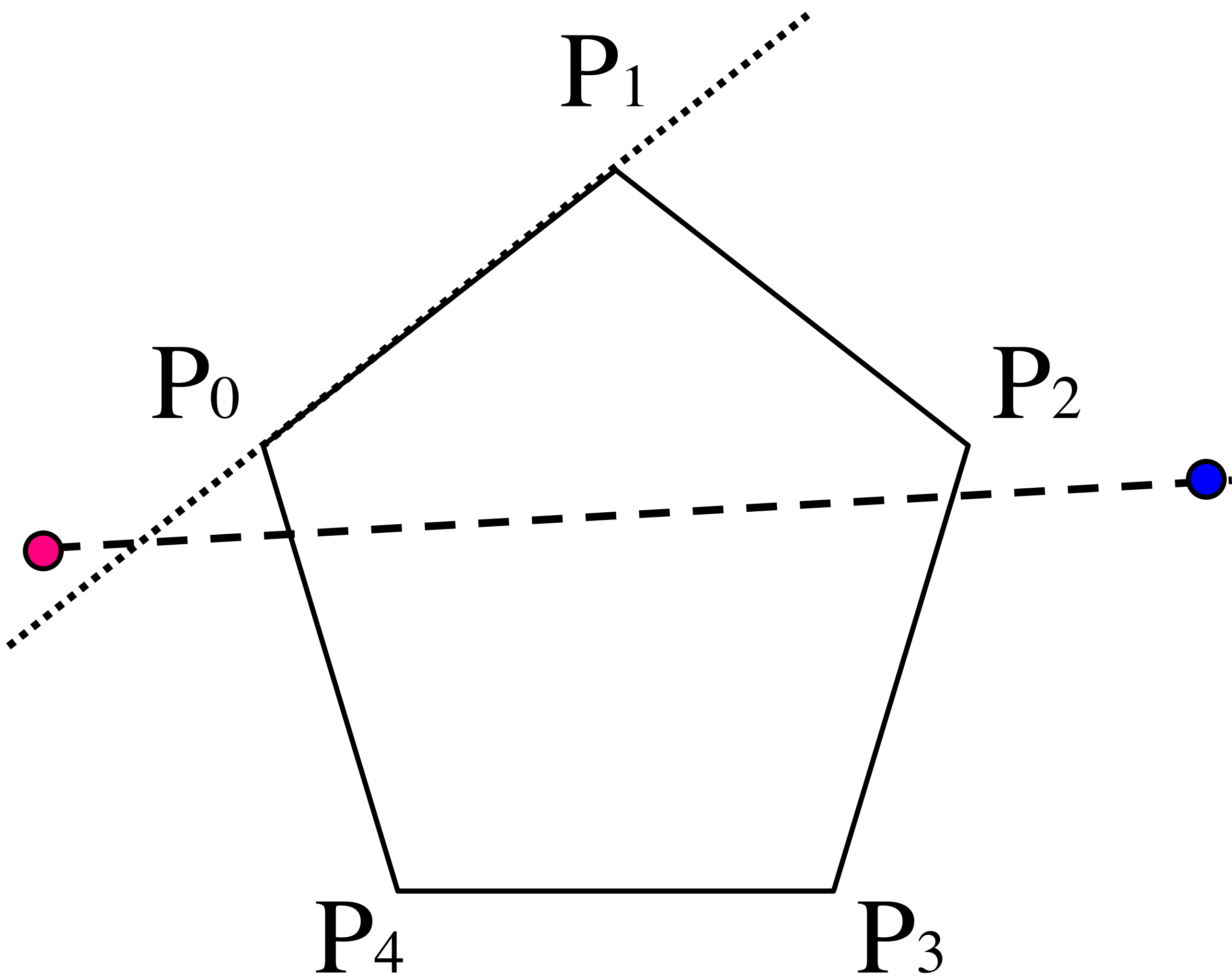
Keep track of minimum t_{out}.

Example



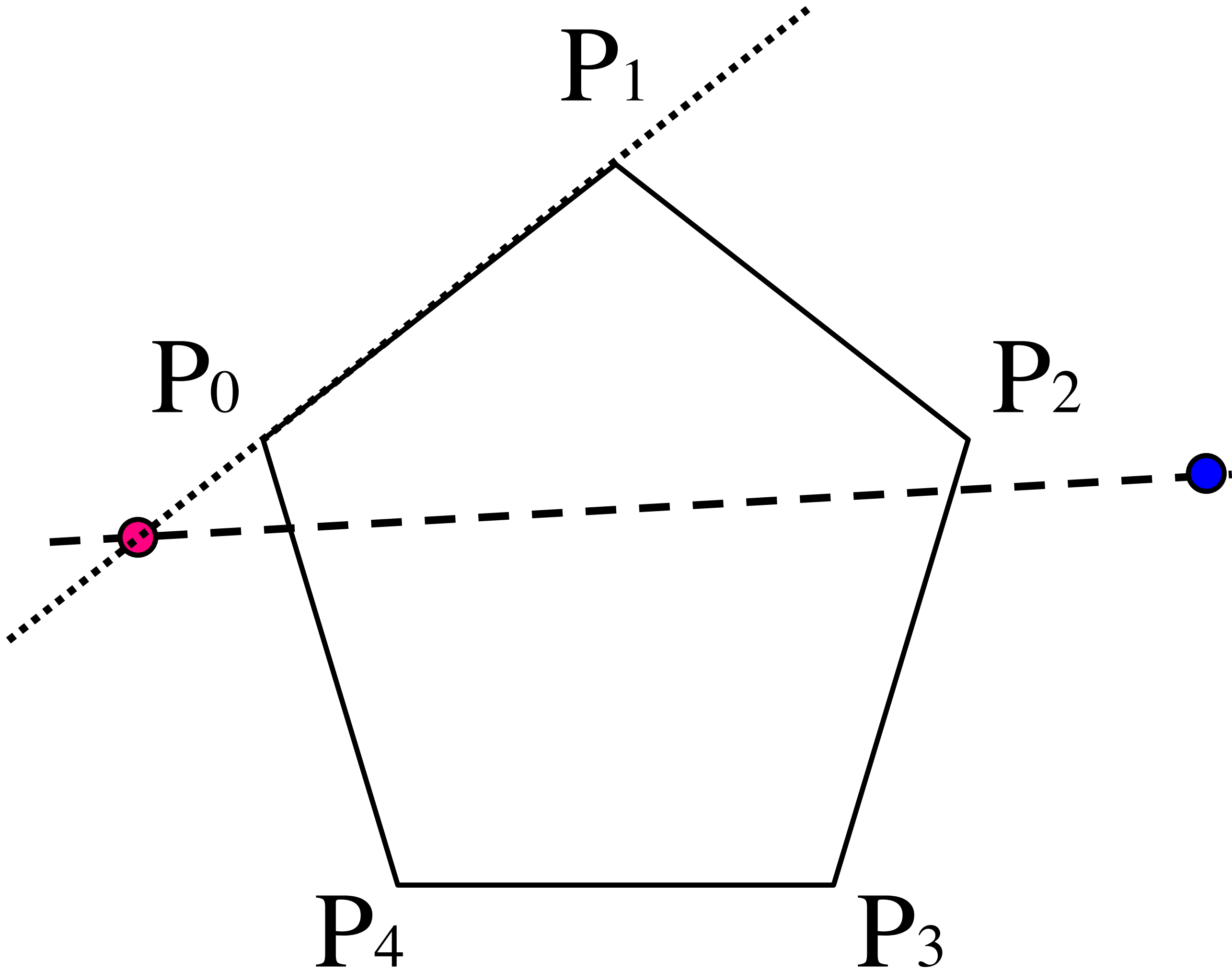
t_{in}	t_{out}
0	1

Example



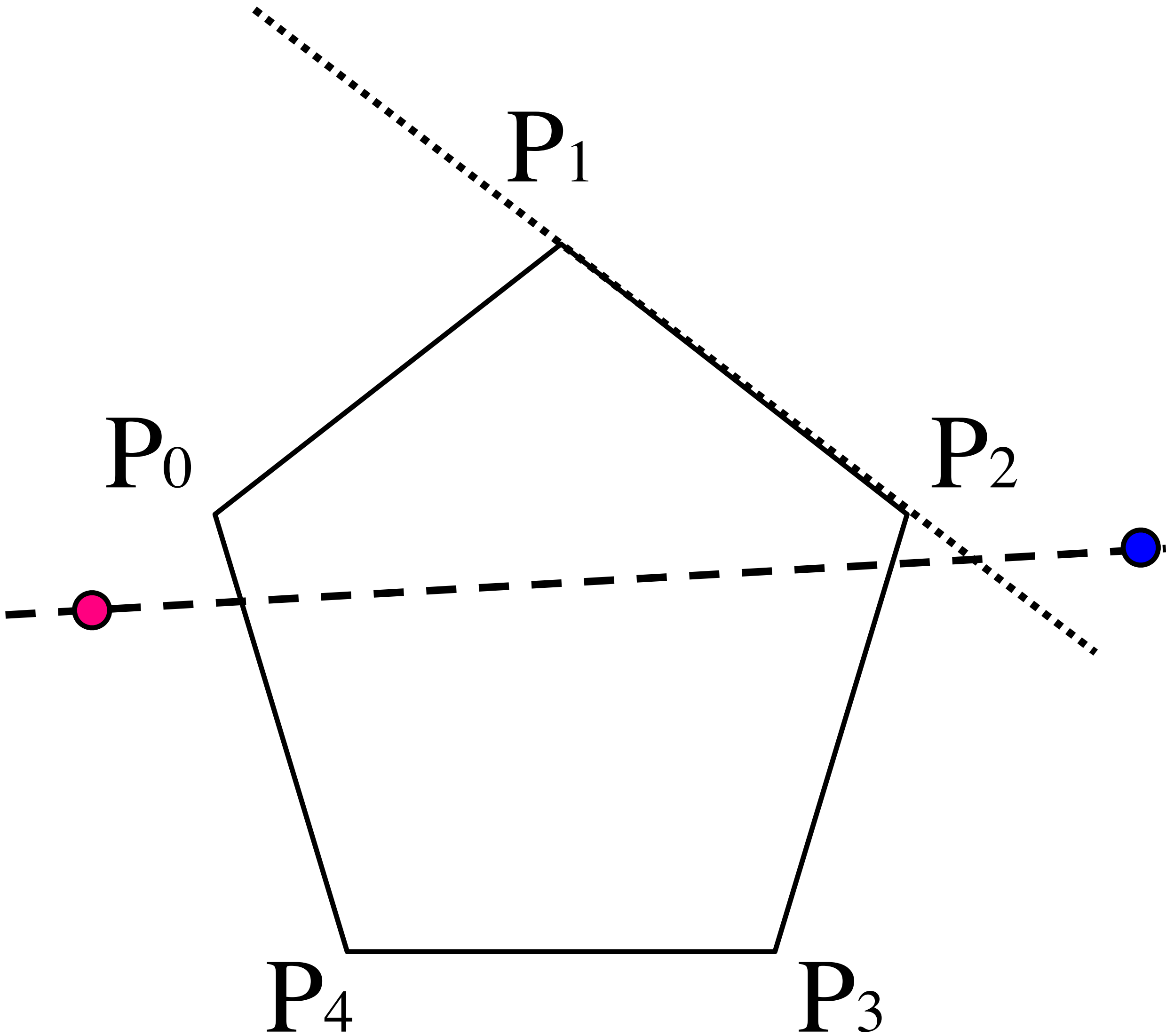
t_{in}	t_{out}
0	1

Example



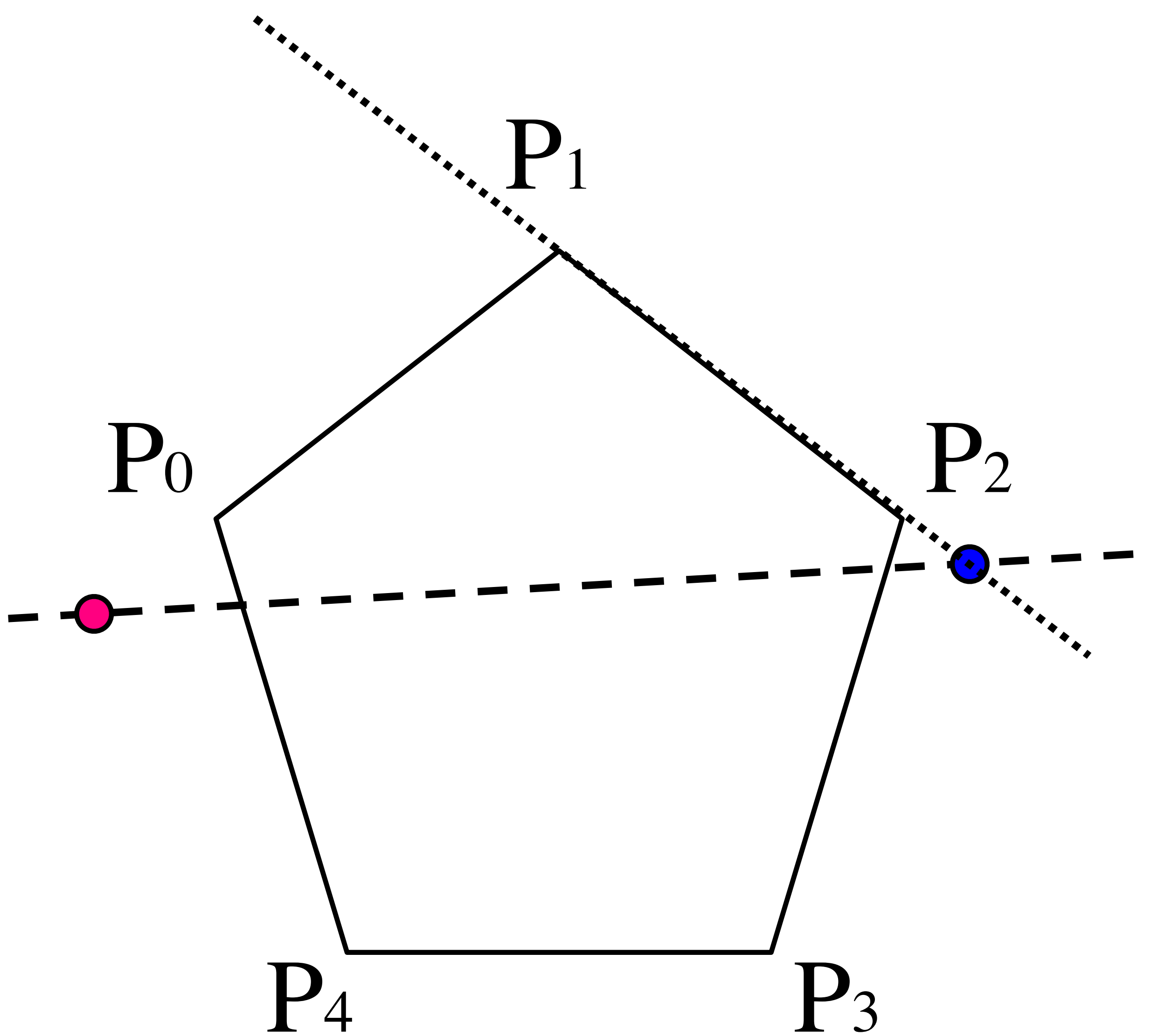
t_{in}	t_{out}
0	1
0.1	1

Example



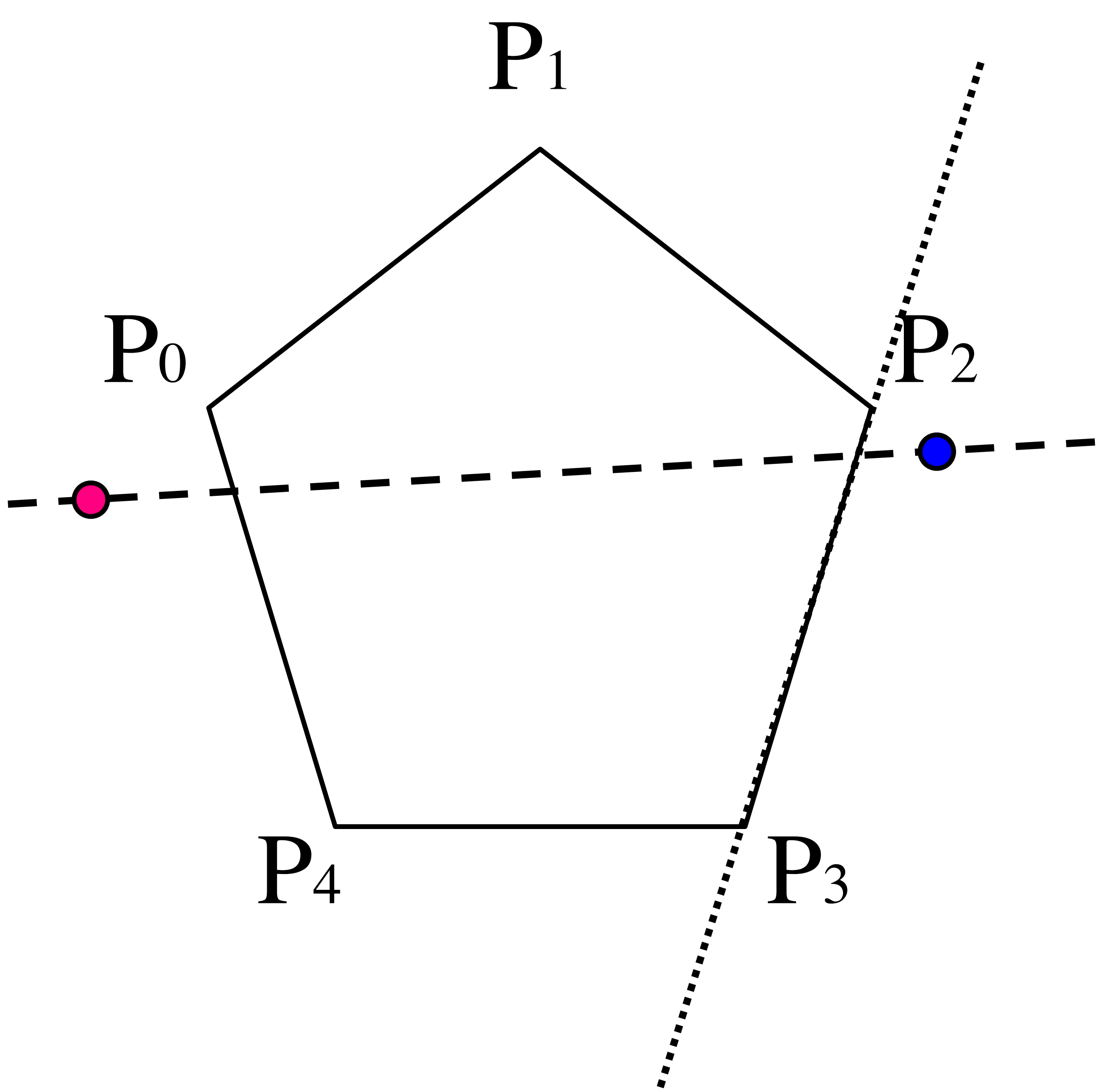
t_{in}	t_{out}
0	1
0.1	1

Example



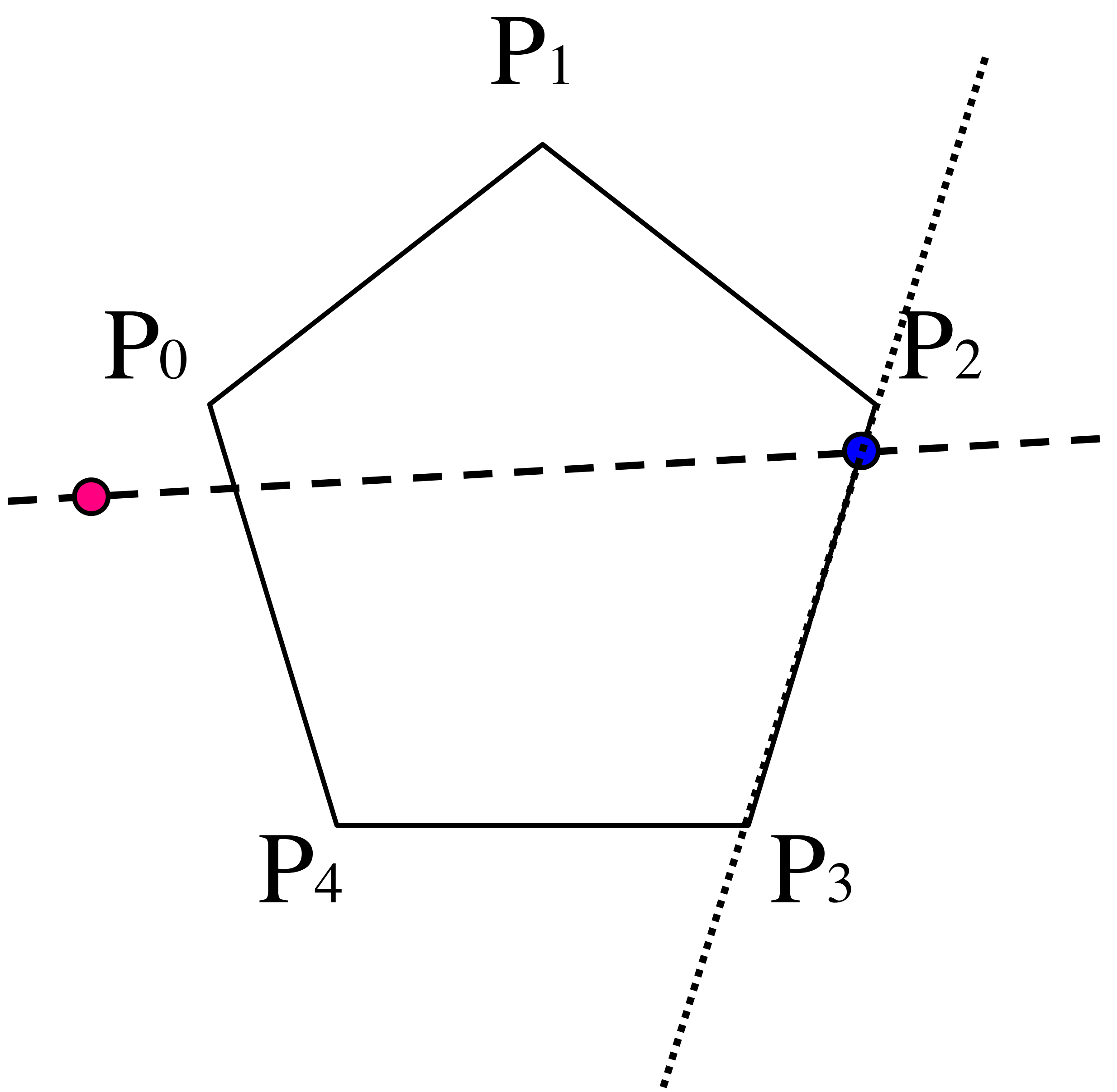
t_{in}	t_{out}
0	1
0.1	1
0.1	0.9

Example



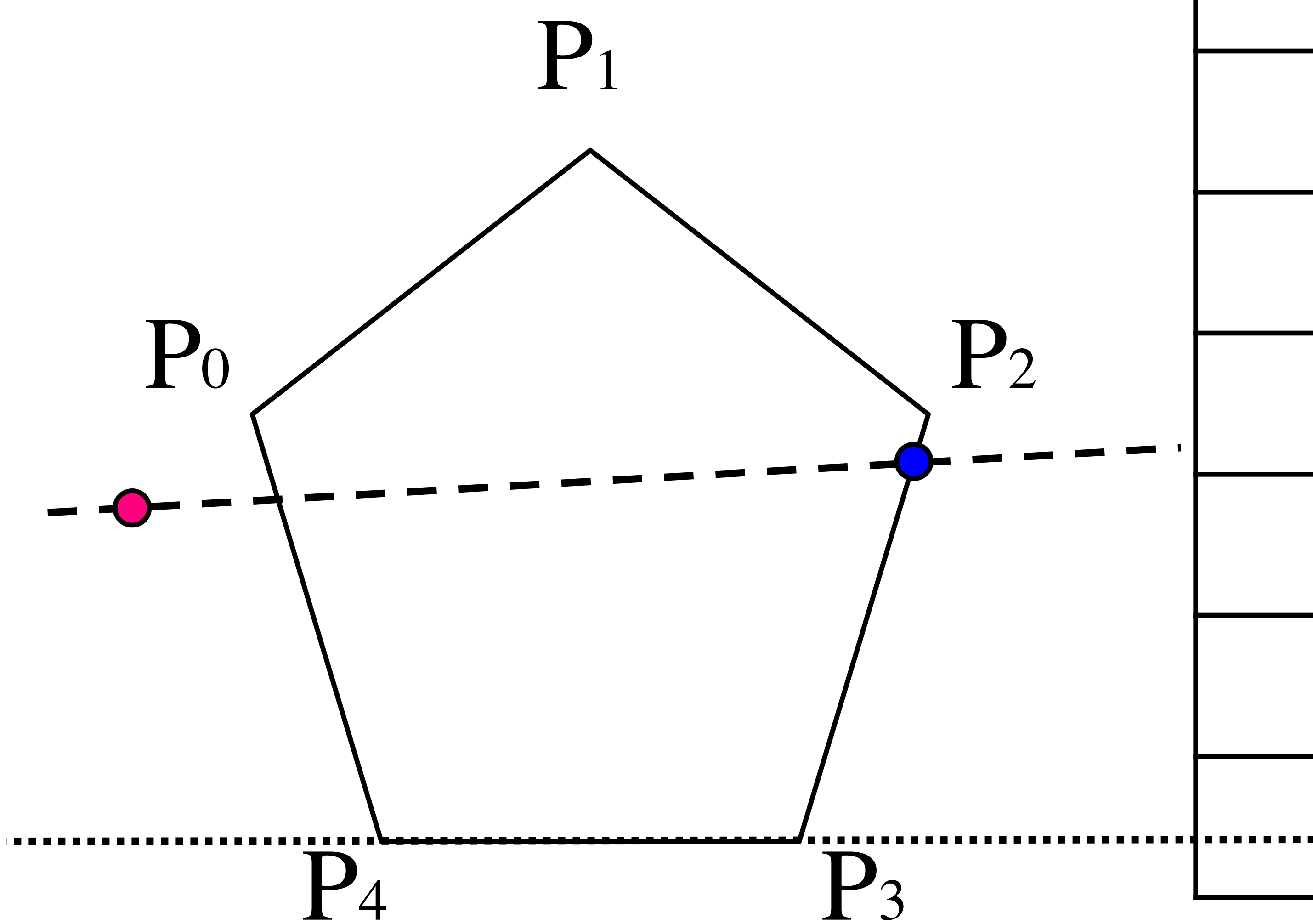
t_{in}	t_{out}
0	1
0.1	1
0.1	0.9

Example



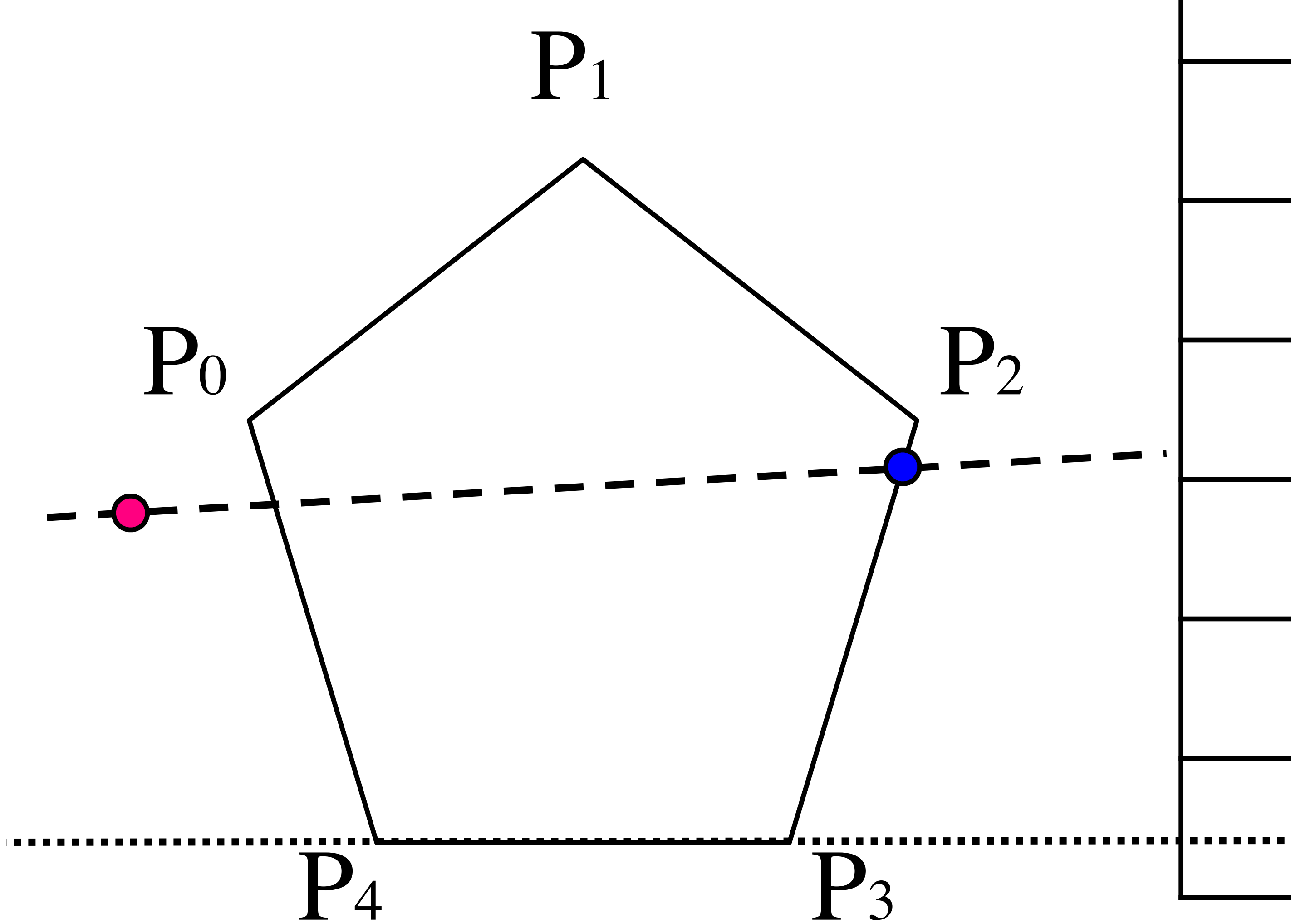
t_{in}	t_{out}
0	1
0.1	1
0.1	0.9
0.1	0.85

Example



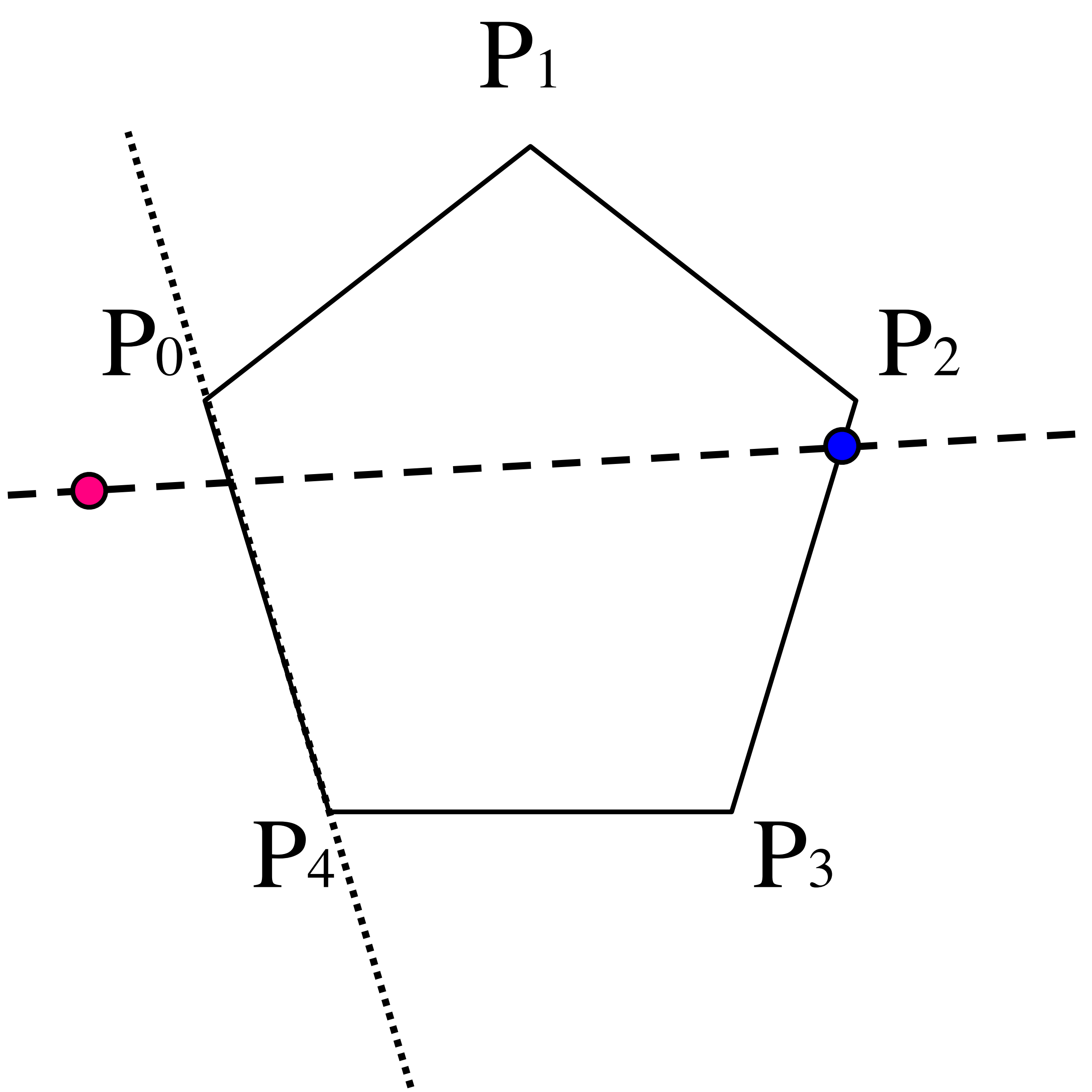
t_{in}	t_{out}
0	1
0.1	1
0.1	0.9
0.1	0.85

Example



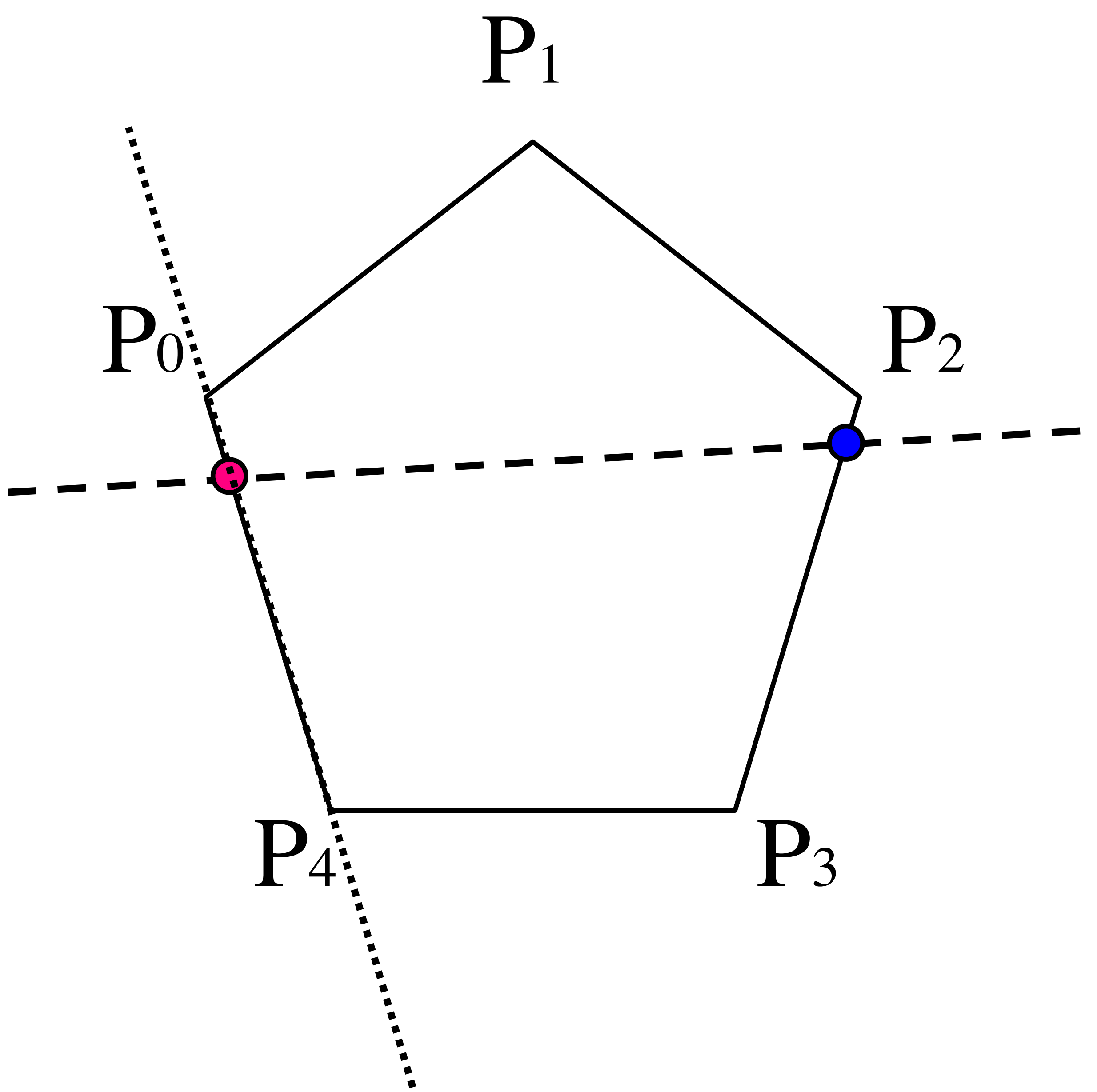
t_{in}	t_{out}
0	1
0.1	1
0.1	0.9
0.1	0.85
0.1	0.85

Example



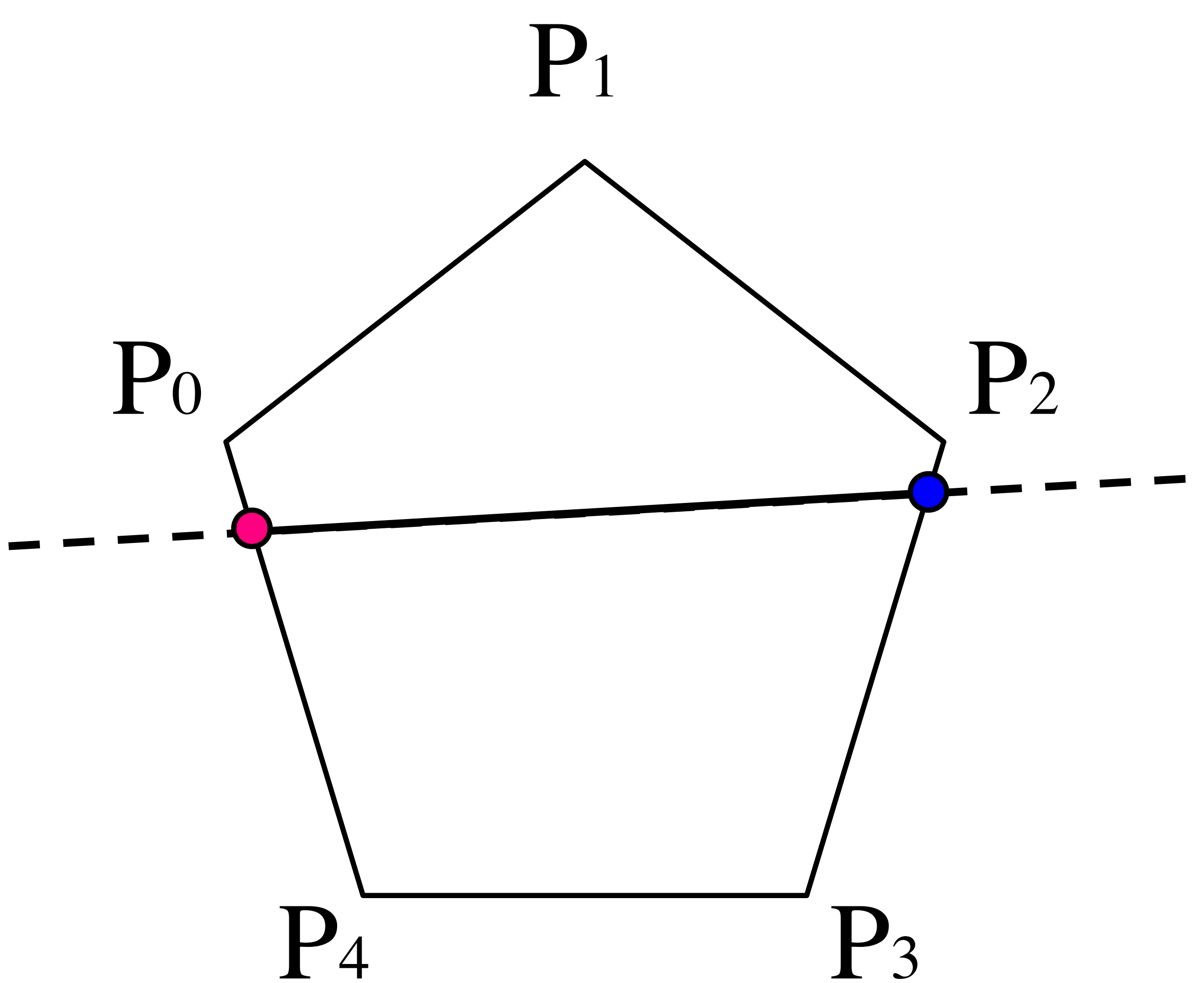
t_{in}	t_{out}
0	1
0.1	1
0.1	0.9
0.1	0.85
0.1	0.85

Example



t_{in}	t_{out}
0	1
0.1	1
0.1	0.9
0.1	0.85
0.1	0.85
0.2	0.85

Example



t_{in}	t_{out}
0	1
0.1	1
0.1	0.9
0.1	0.85
0.1	0.85
0.2	0.85