COMP3421

Depth, Clipping

Robert Clifton-Everest

Email: robertce@cse.unsw.edu.au

Recap: The transformation pipeline

- To transform a point:

$$P = (p_x, p_y, p_z)^\top$$

- Extend to homogeneous coordinates:

$$P = (p_x, p_y, p_z, 1)^\top$$

- Multiply by model matrix to get world coordinates:

$$P_w = \mathbf{M}_{\mathbf{model}} P$$

- Multiply by view matrix to get camera (eye) coordinates:

$$P_c = \mathbf{M}_{\mathbf{view}} P_w$$

Recap: The transformation pipeline

- Multiply by projection matrix to get CVV coordinates (with fourth component):

$$P_{cvv} = \mathbf{M_P} P_c$$

- Clip to remove points outside CVV.
- Perspective division to eliminate fourth component.

$$P_n = \frac{1}{p_w} P_{cvv}$$

- Viewport transformation to window coordinates.

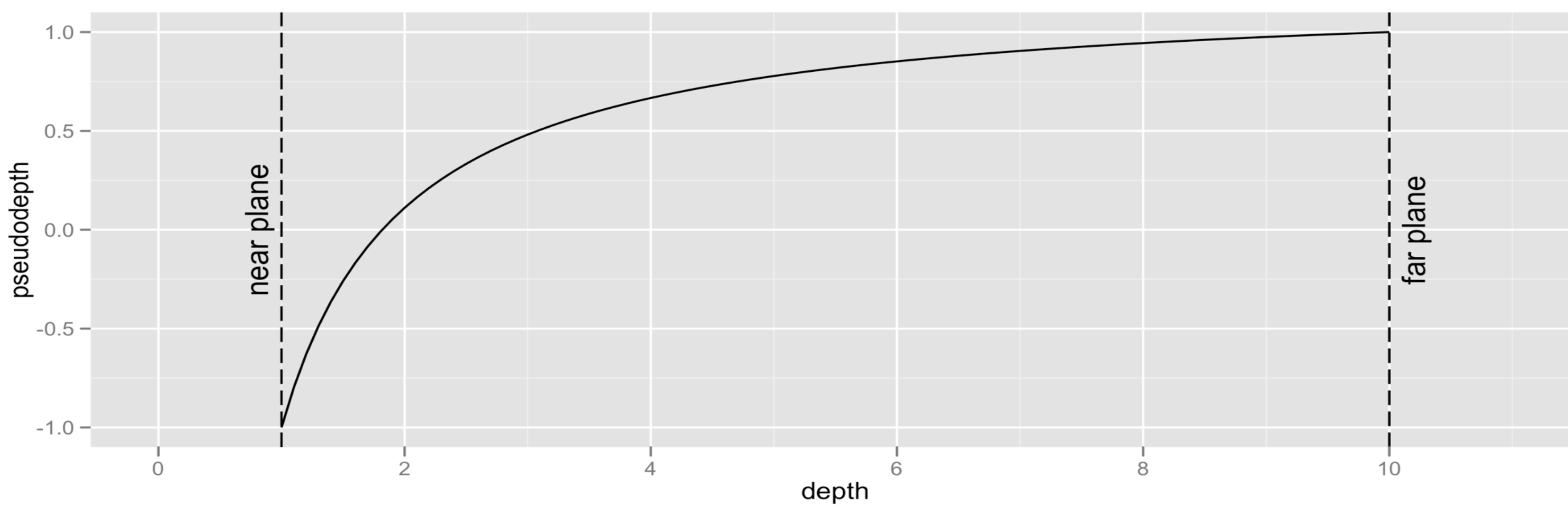
$$P_v = \mathbf{M_{viewport}} P_n$$

Recap: The transformation pipeline

- In vertex shaders, gl_Position is the point in CVV coordinates.

- The subsequent stages of transformation are (by default) performed internally by the OpenGL implementation

Recap: Pseudodepth



Not linear. More precision for objects closer to the near plane. Rounding errors worse towards far plane.

Tip: Avoid setting near and far needlessy small/ big for better use of precision

Computing pseudodepth

We know how to compute the pseudo depth of a vertex.

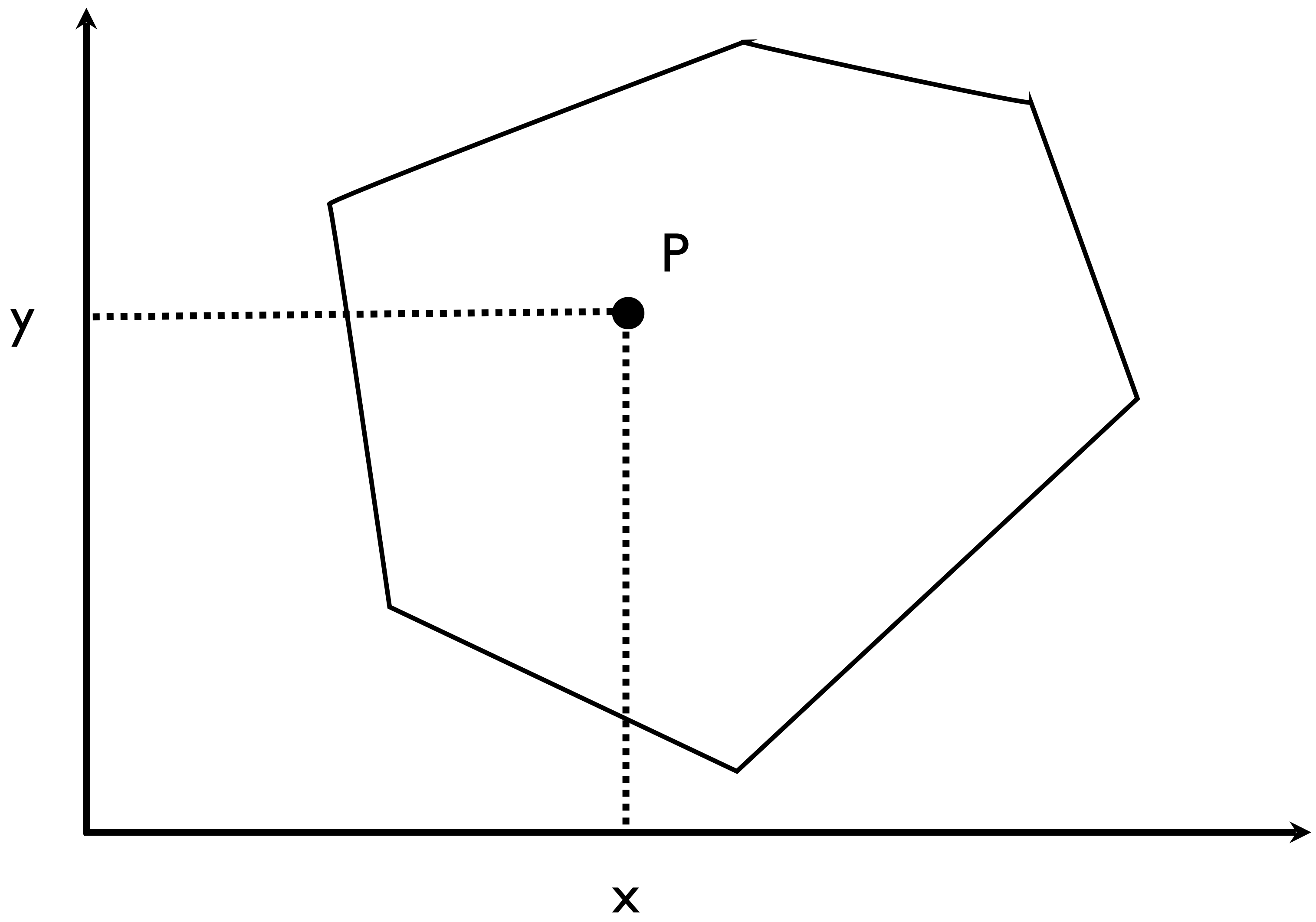
How do we compute the depth of a fragment?

We use bilinear interpolation based on the depth values for the triangle vertices.

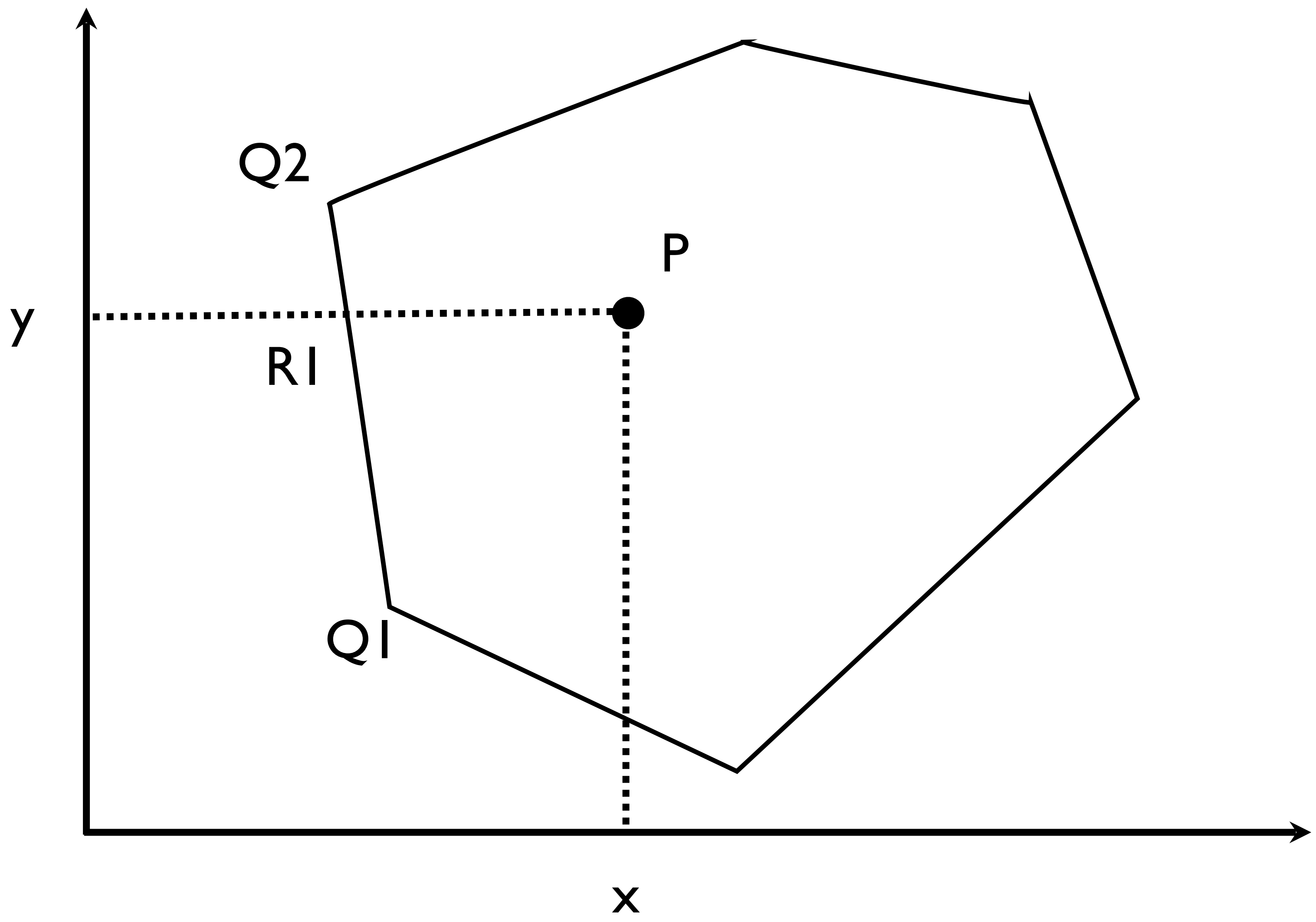
Bilinear interpolation is lerping in 2 dimensions.

It works for any polygon.

Bilinear interpolation

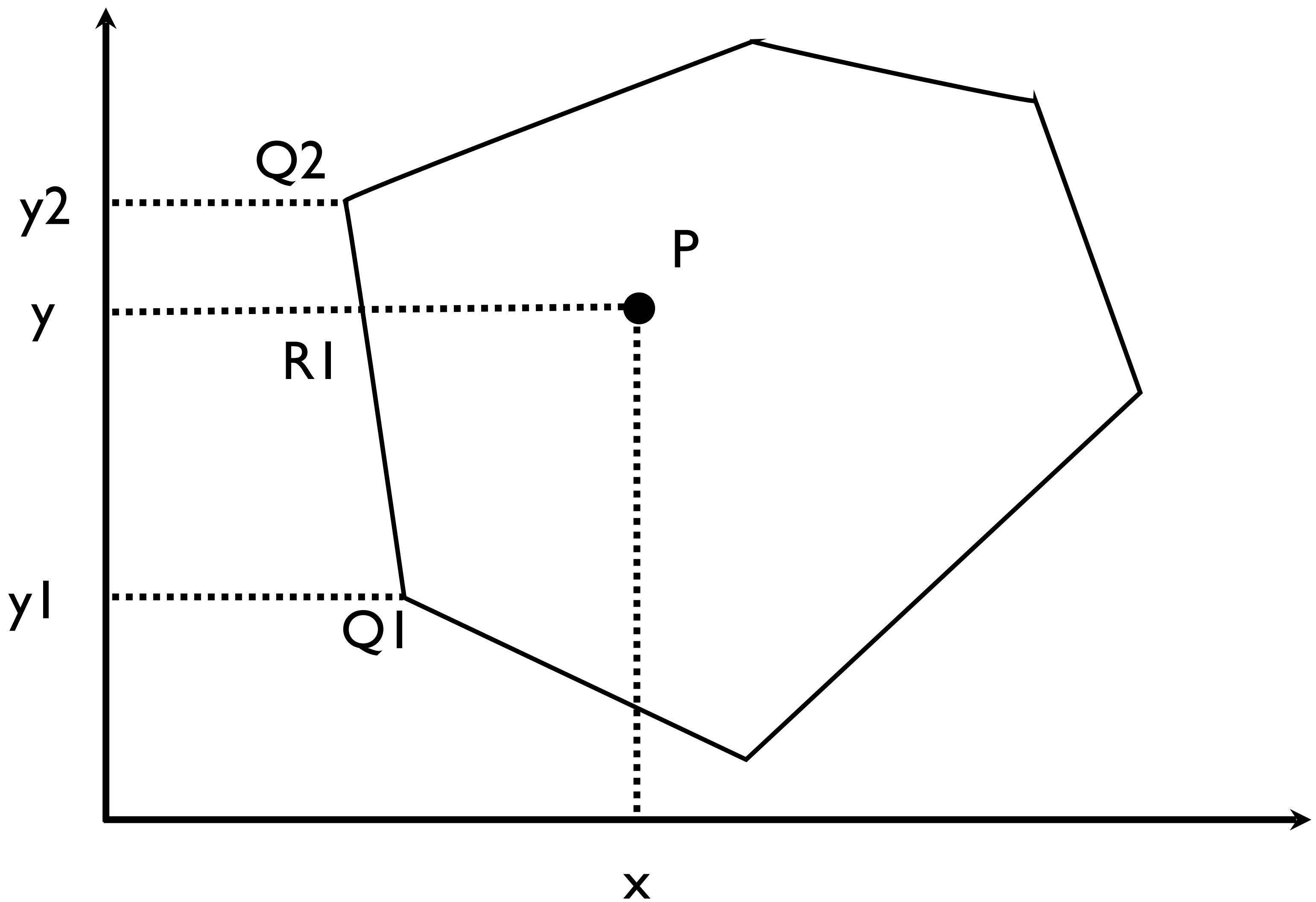


Bilinear interpolation



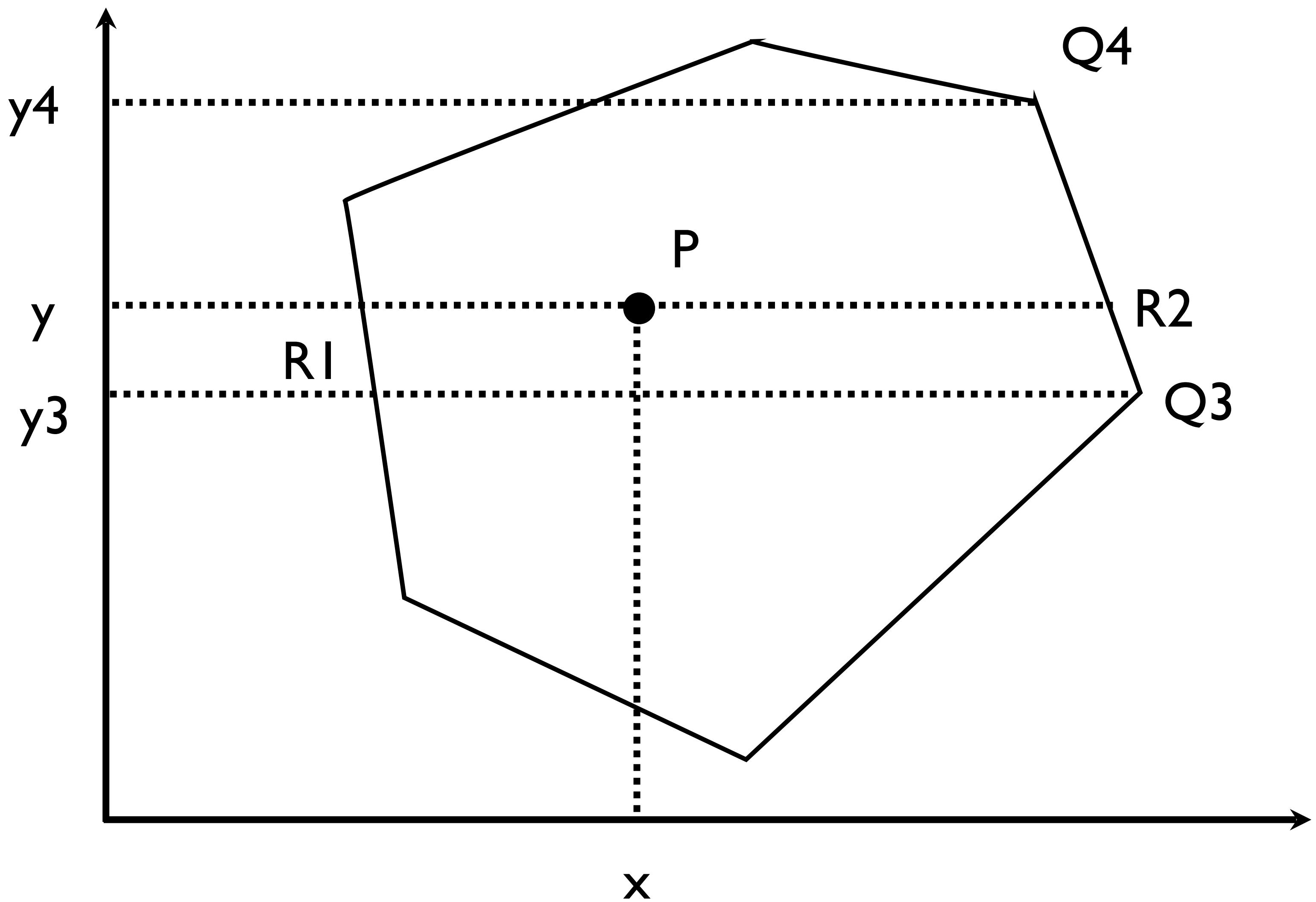
Bilinear interpolation

$$f(R_1) = \frac{y - y_1}{y_2 - y_1} f(Q_2) + \frac{y_2 - y}{y_2 - y_1} f(Q_1)$$



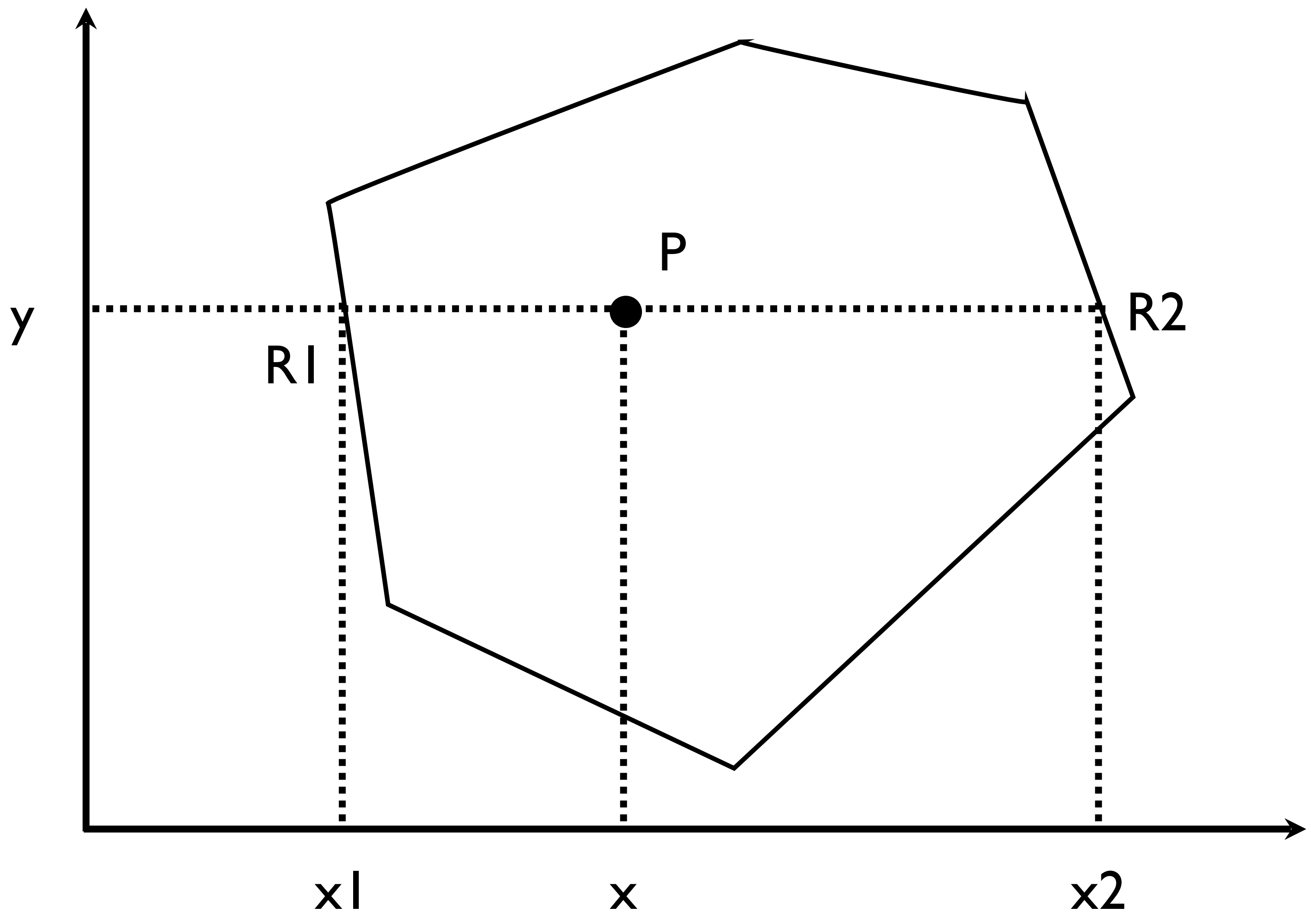
Bilinear interpolation

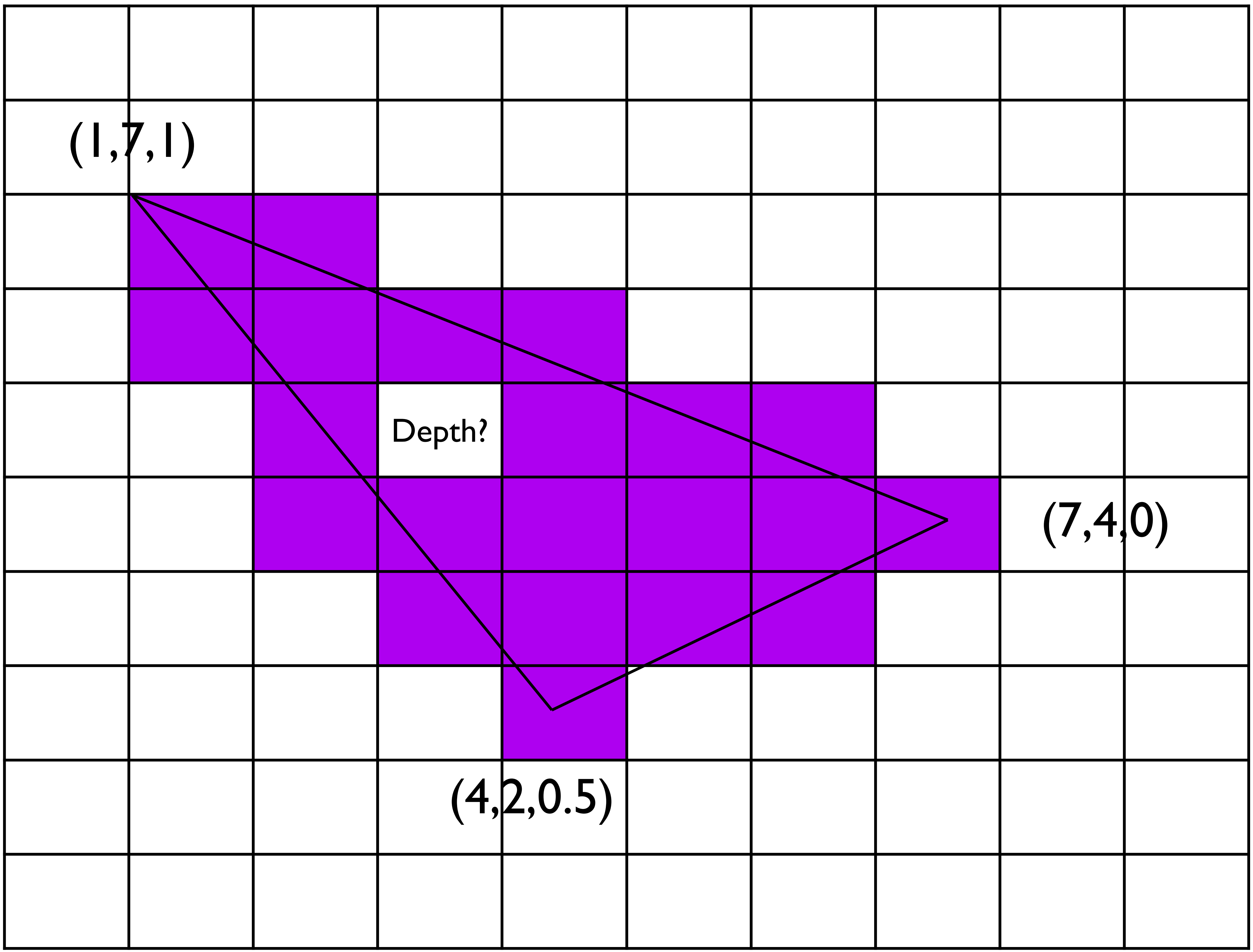
$$f(R_2) = \frac{y - y_3}{y_4 - y_3} f(Q_4) + \frac{y_4 - y}{y_4 - y_3} f(Q_3)$$

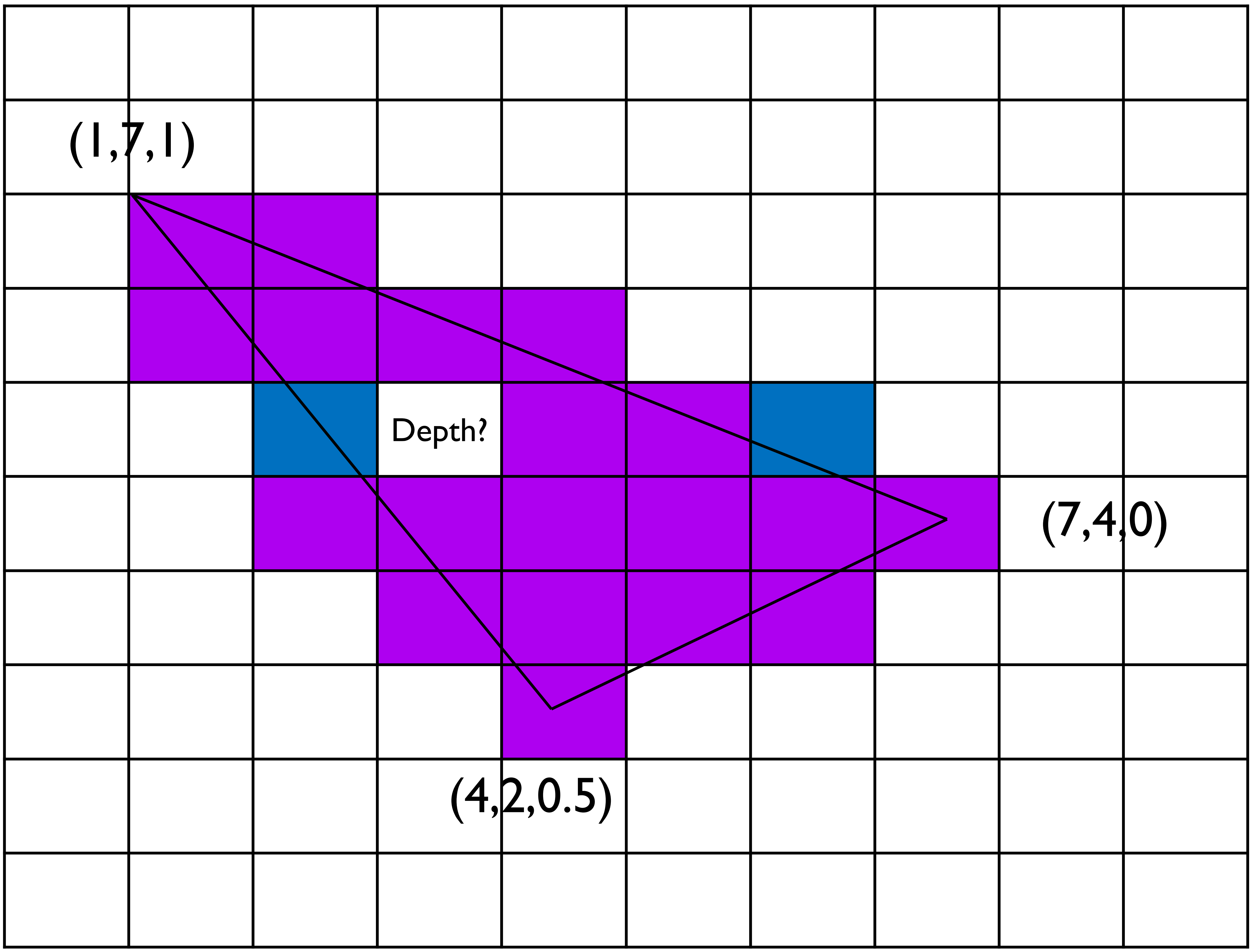


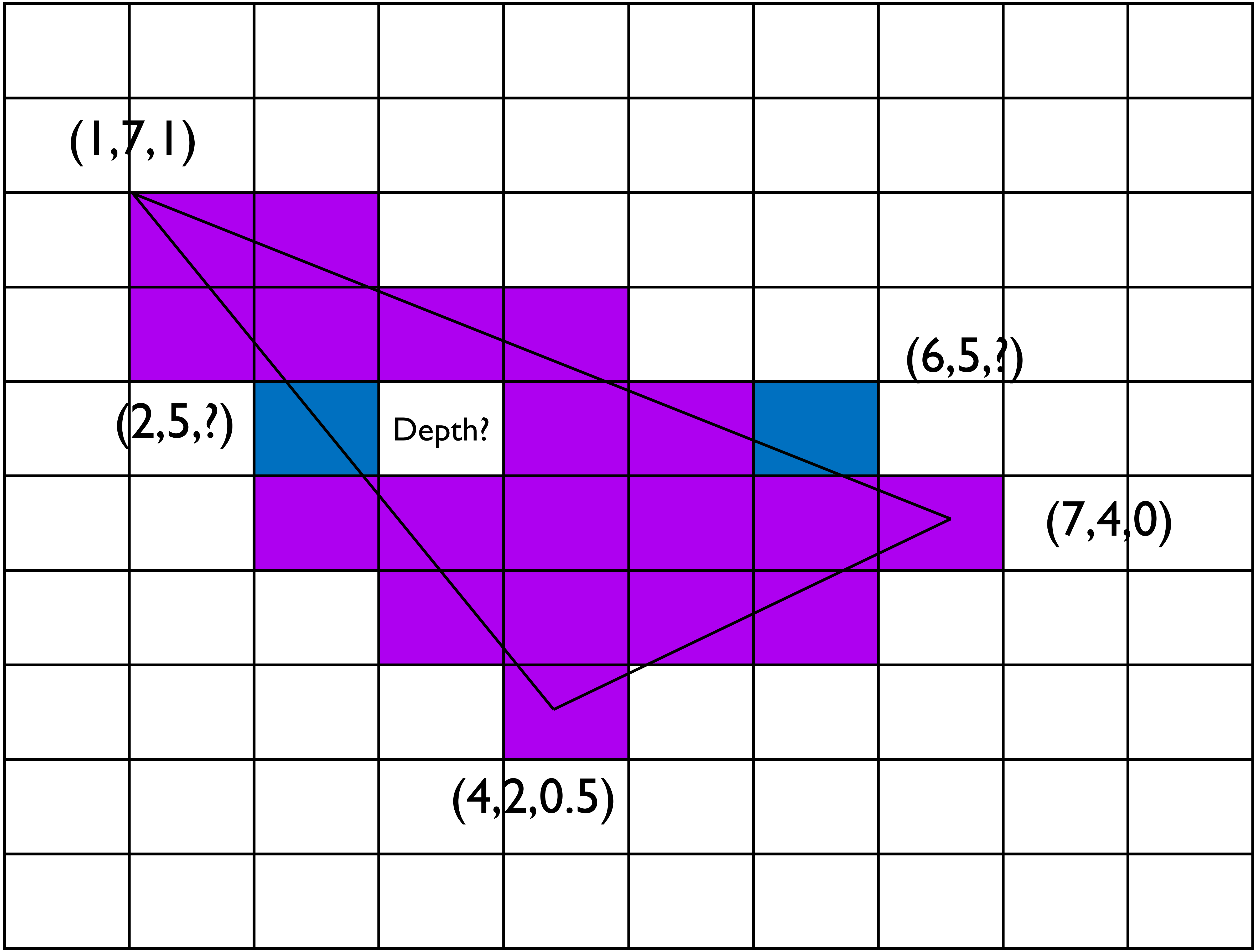
Bilinear interpolation

$$f(P) = \frac{x - x_1}{x_2 - x_1} f(R_2) + \frac{x_2 - x}{x_2 - x_1} f(R_1)$$




(1,7,1)
Depth?
(7,4,0)
(4,2,0.5)


(1,7,1)
Depth?
(7,4,0)
(4,2,0.5)


(1,7,1)
(6,5,?)
(2,5,?)
Depth?
(7,4,0)
(4,2,0.5)

Interpolation - Y

$$f(R_1) = \frac{y - y_1}{y_2 - y_1} f(Q_2) + \frac{y_2 - y}{y_2 - y_1} f(Q_1)$$

$$Q_1 = (4,2,0.5)$$

$$Q_2 = (1,7,1)$$

$$f(R_1) = \frac{5-2}{7-2}(1) + \frac{7-5}{7-2}(0.5)$$

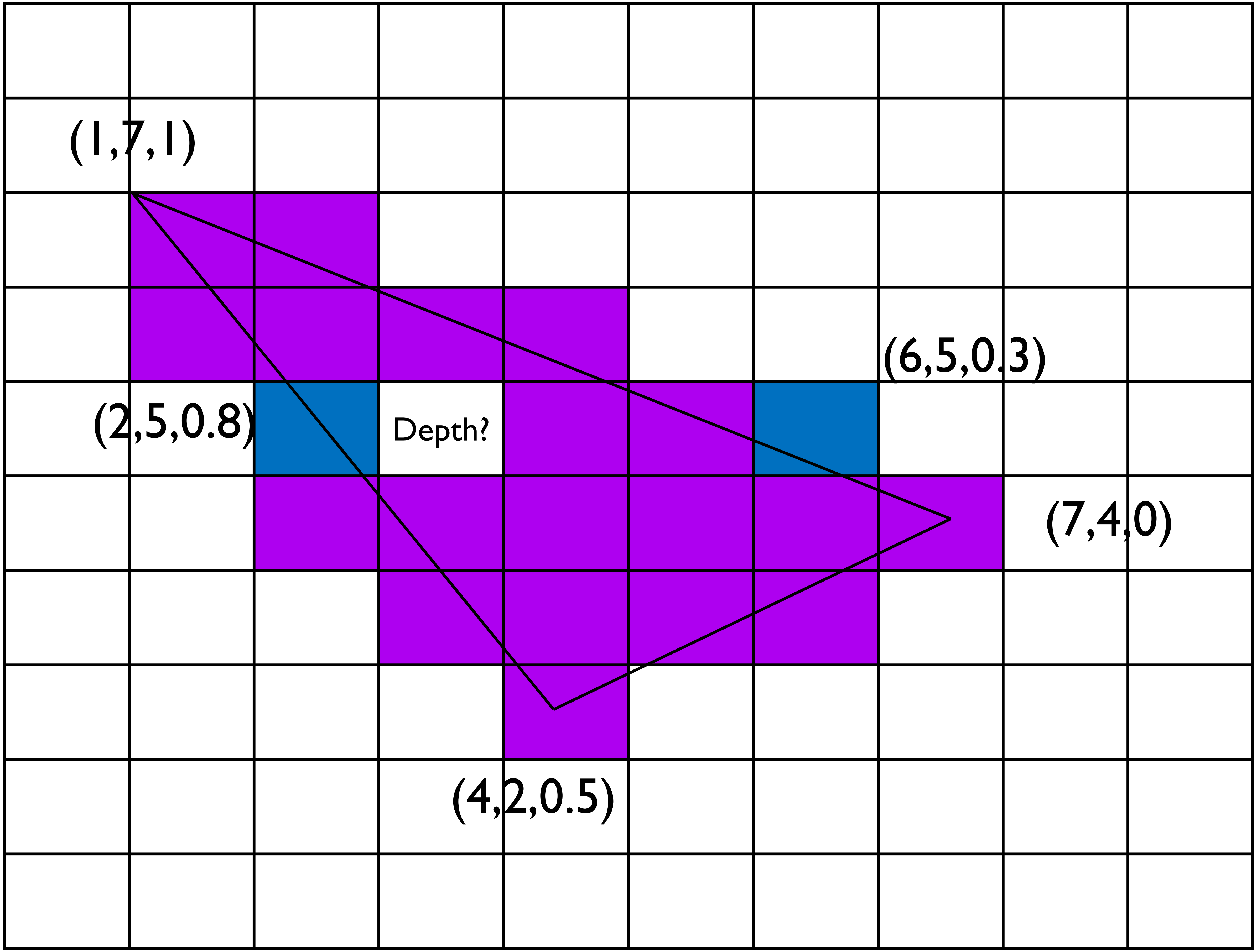
$$= \frac{3}{5} + \frac{1}{5}$$

$$= 0.8$$

Interpolation - Y

$$f(R_2) = \frac{y - y_3}{y_4 - y_3} f(Q_4) + \frac{y_4 - y}{y_4 - y_3} f(Q_3)$$

$$\begin{aligned}
Q_3 &= (7,4,0) \\
Q_4 &= (1,7,1) \\
f(R_2) &= \frac{5-4}{7-4}(1) + \frac{7-5}{7-4}(0) \\
&= \frac{1}{3} \\
&\approx 0.3
\end{aligned}$$



(1,7,1)
(6,5,0.3)
(2,5,0.8)
Depth?
(7,4,0)
(4,2,0.5)

Interpolation - X

$$f(P) = \frac{x - x_1}{x_2 - x_1} f(R_2) + \frac{x_2 - x}{x_2 - x_1} f(R_1)$$

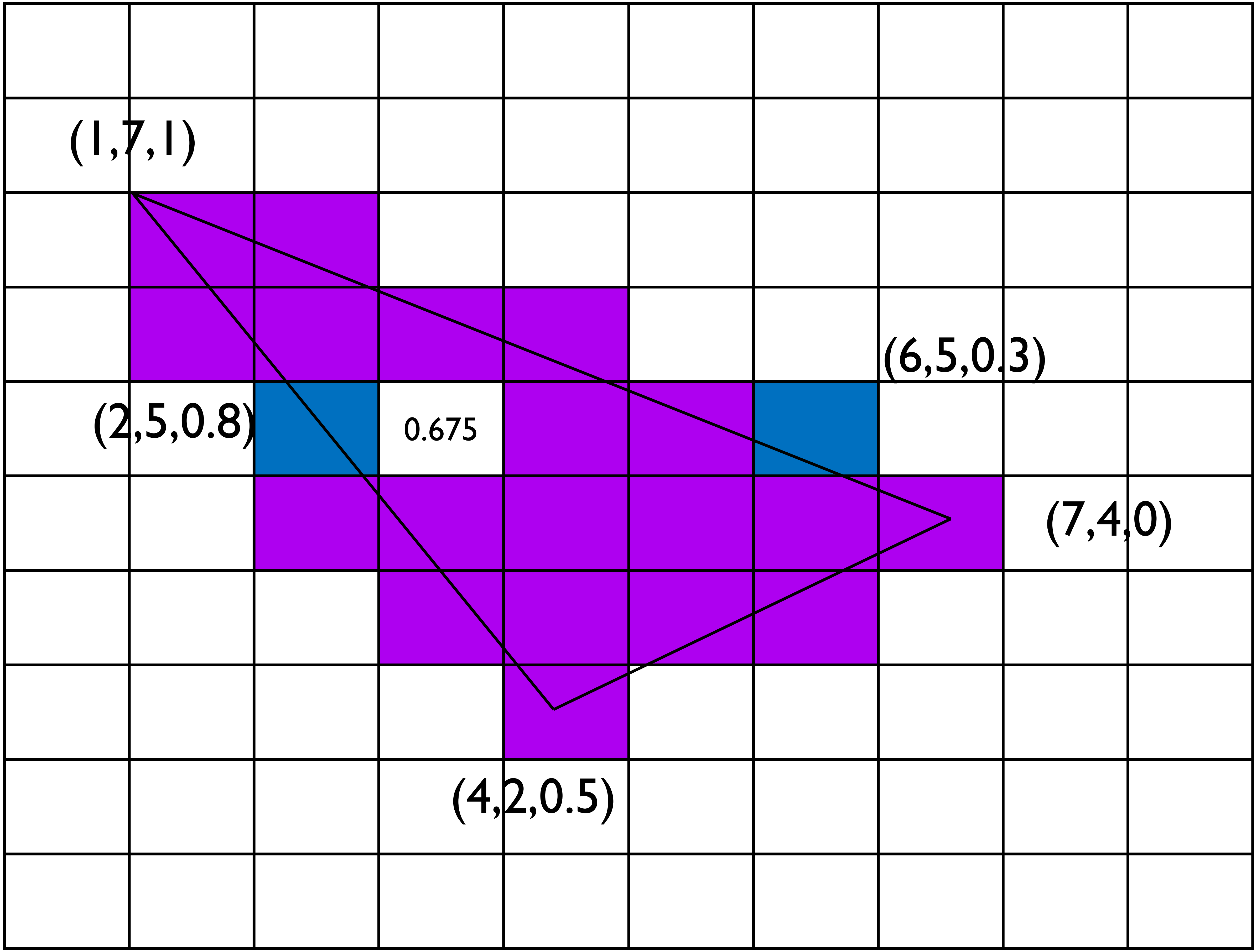
$$R_1 = (2,4,0.8)$$

$$R_2 = (6,5,0.3)$$

$$f(P) = \frac{3-2}{6-2}(0.3) + \frac{6-3}{6-2}(0.8)$$

$$= \frac{1}{4}(0.3) + \frac{3}{4}(0.8)$$

$$= 0.675$$


(1,7,1)
(6,5,0.3)
(2,5,0.8)
0.675
(7,4,0)
(4,2,0.5)

Z-fighting

The depth buffer has limited precision (usually 16 bits).

If two polygons are (almost) parallel small rounding errors will cause them to "fight" for which one will be in front, creating strange effects.

glPolygonOffset

When you have two overlapping polygons you can get Z-fighting.

To prevent this, you can offset one of the two polygons using glPolygonOffset().

This method adds a small offset to the pseudodepth of any vertices added after the call. You can use this to move a polygon slightly closer or further away from the camera.

glPolygonOffset

To use glPolygonOffset you must first enable it. You can enable offsetting for points, lines and filled areas separately:

```
gl.glEnable(
    GL2.GL_POLYGON_OFFSET_POINT);
gl.glEnable(
    GL2.GL_POLYGON_OFFSET_LINE);
gl.glEnable(
    GL2.GL_POLYGON_OFFSET_FILL);
```

glPolygonOffset

The method takes two parameters:

```
gl.glPolygonOffset(
              factor, units);
```

The offset added to the pseudodepth is calculated as:

$$o = m * factor + r * units$$

m is the depth slope of the polygon
r is the smallest resolvable difference in depth

glPolygonOffset

Usually you will call this as either:

```
//Push polygon back a bit
gl.glPolygonOffset(1.0, 1.0);

//Push polygon forward a bit
gl.glPolygonOffset(-1.0, -1.0);
```

If this does not give you the results you need play around with values or check the (not very clear) documentation

glPolygonOffset

You should also disable it when you have finished (as it is state based)

```
gl.glDisable(
    GL2.GL_POLYGON_OFFSET_POINT);
gl.glDisable(
    GL2.GL_POLYGON_OFFSET_LINE);
gl.glDisable(
    GL2.GL_POLYGON_OFFSET_FILL);
```

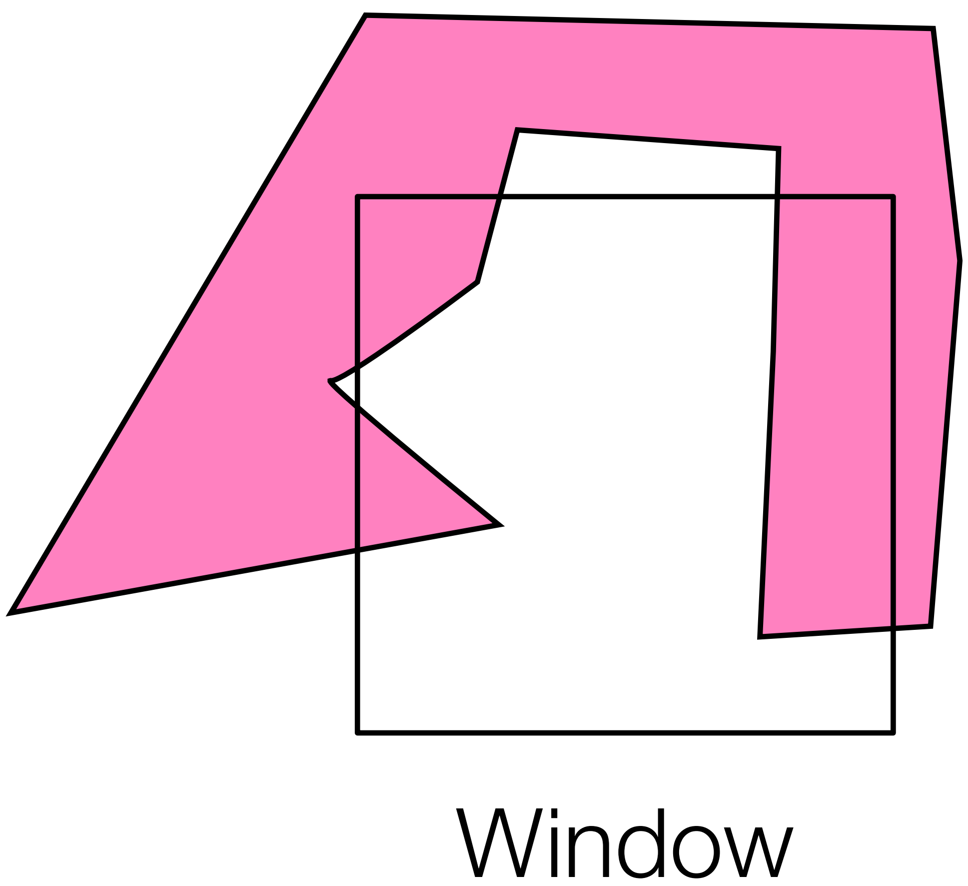
glPolygonOffset

Why not just position it slightly closer?

In a perspective projection, that causes a gap when you look at it sideways.

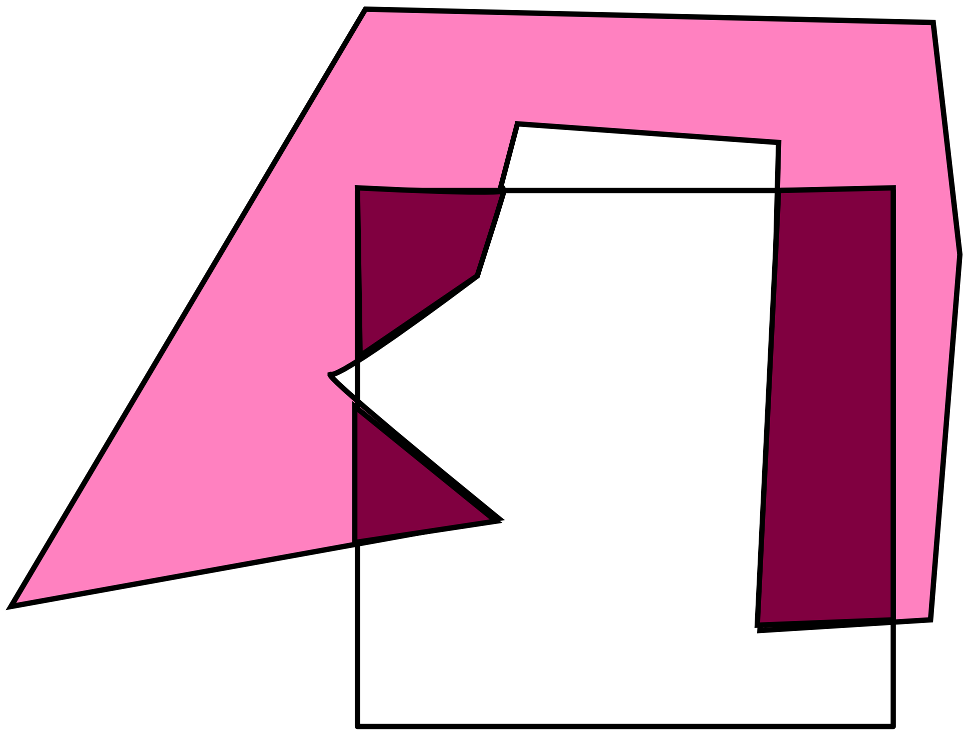
Clipping

- 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.

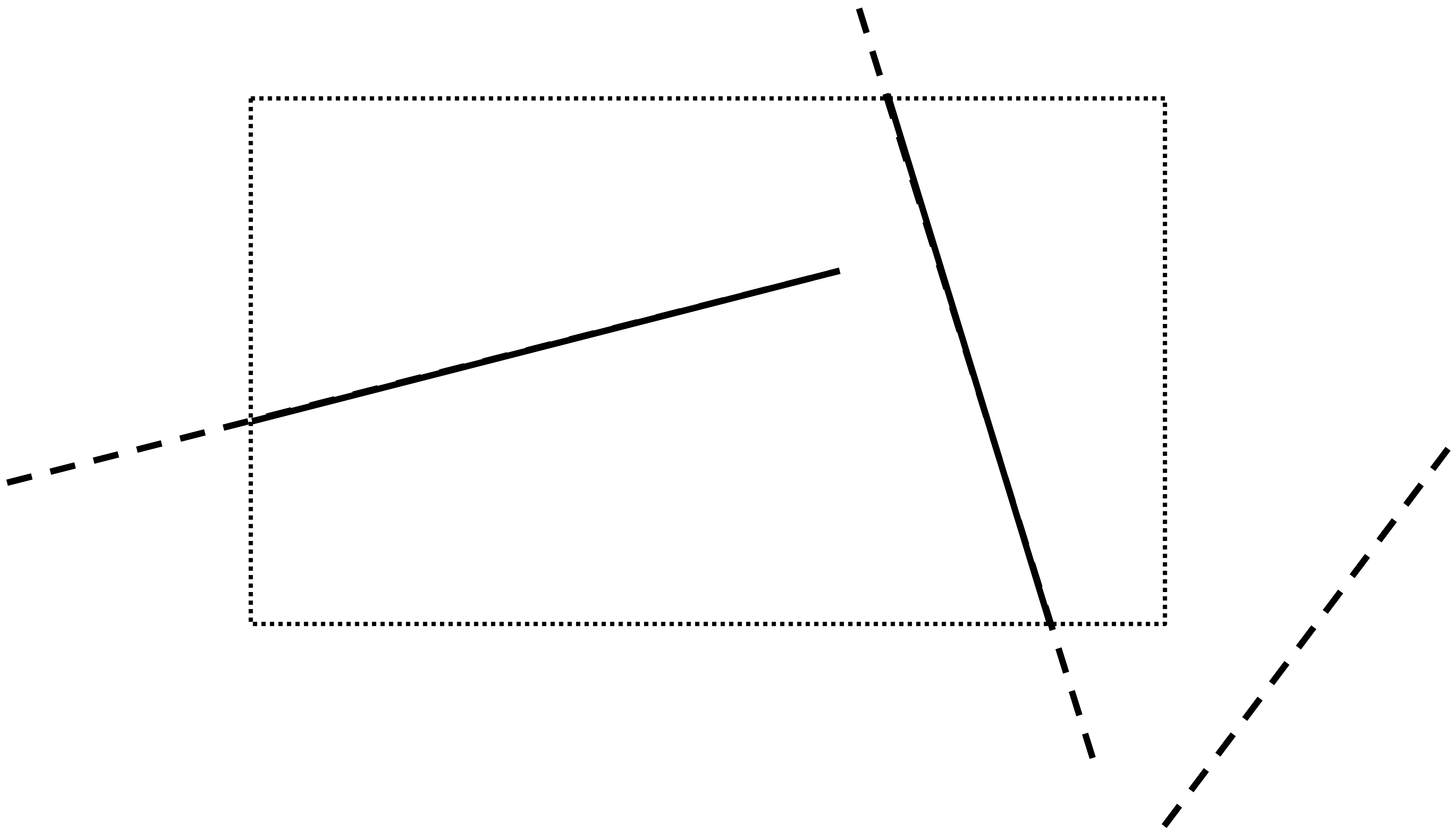


Clipping algorithms

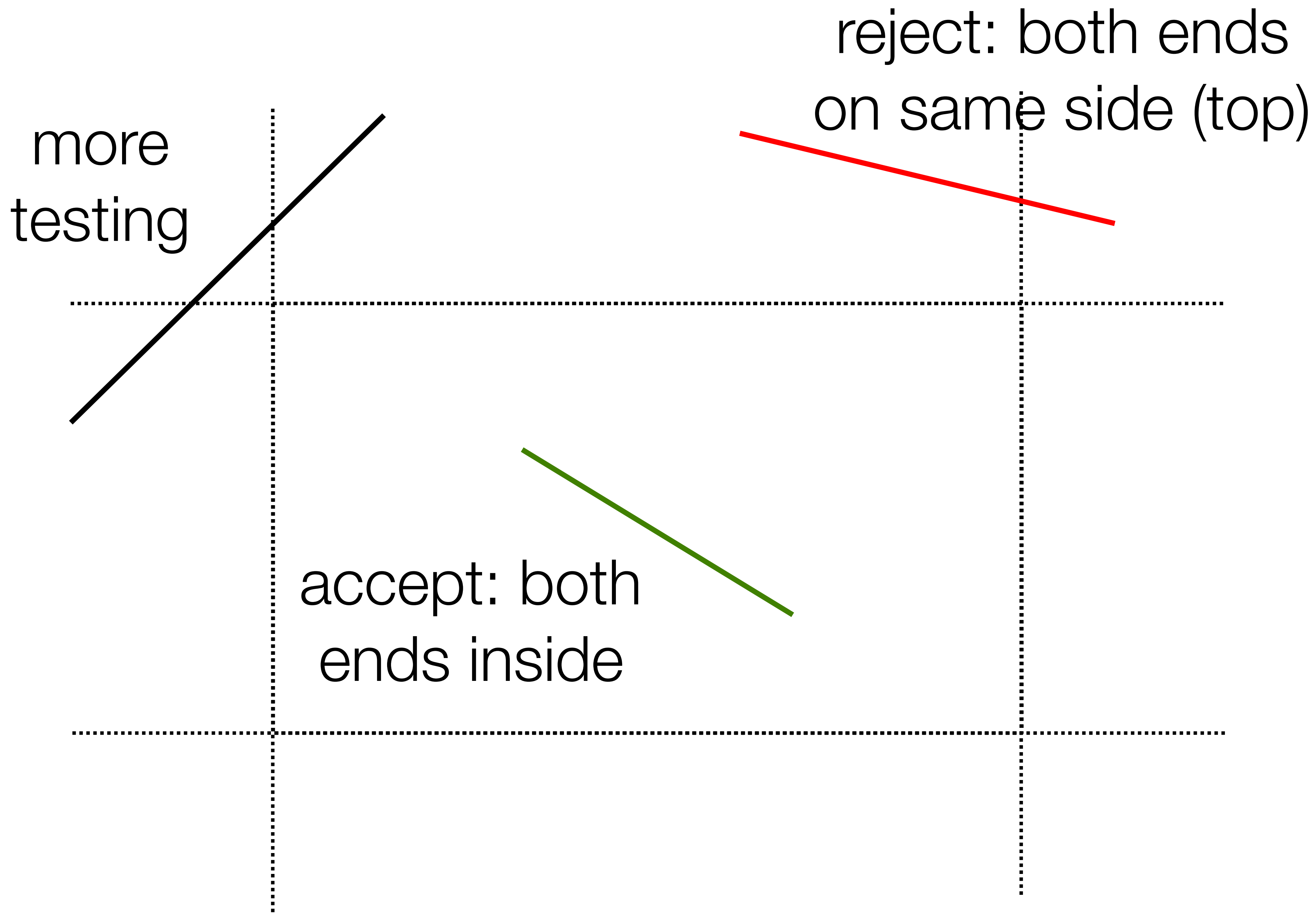
- There are a number of different clipping algorithms:
  - Cohen-Sutherland (line vs rect)
  - Cyrus-Beck (line vs convex poly)
  - Sutherland-Hodgman (poly vs convex poly)
  - Weiler-Atherton (poly vs poly)

Cohen-Sutherland

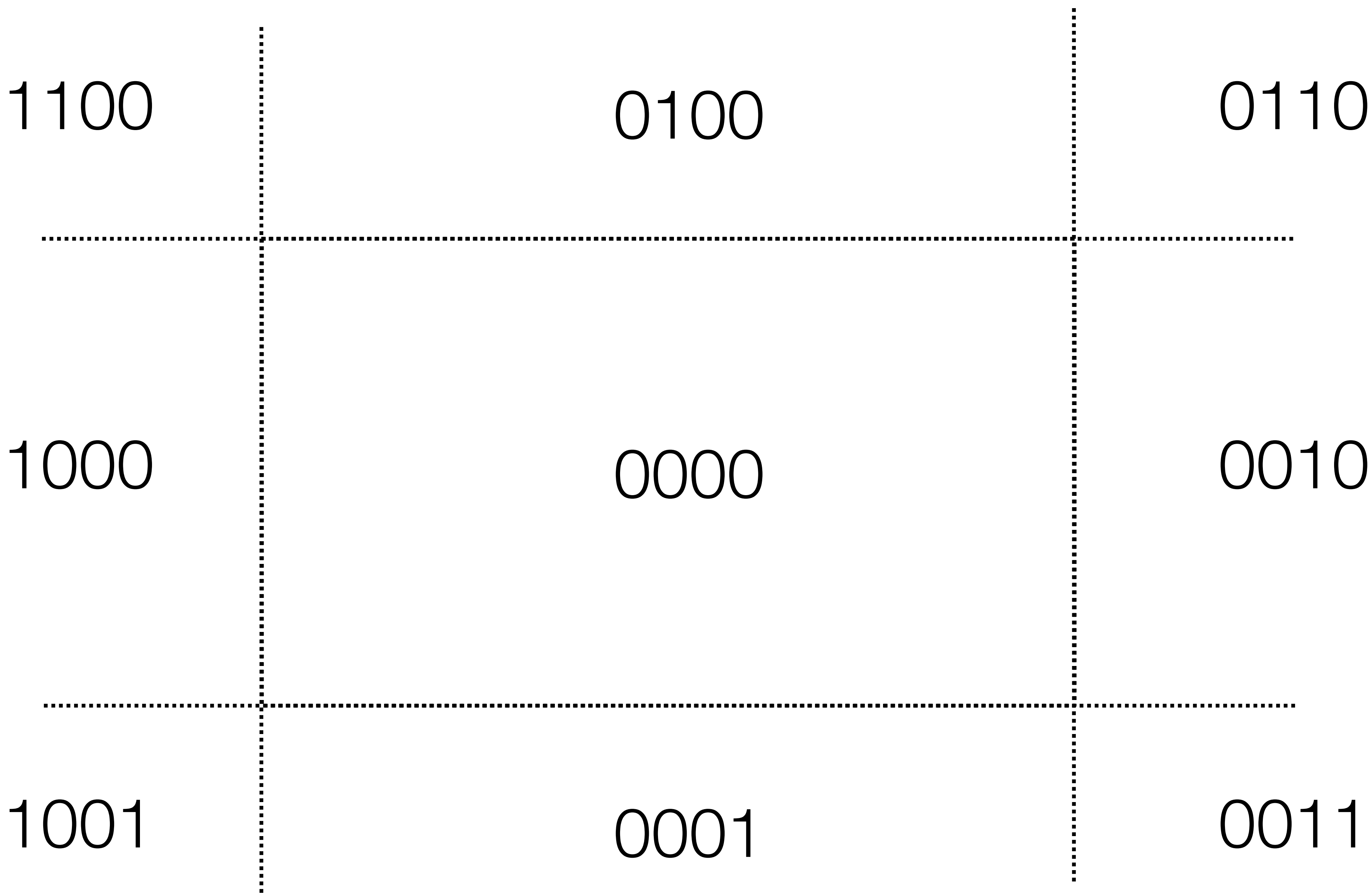
- Clipping lines to an axis-aligned rectangle.



Trivial accept/reject



Labelling



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

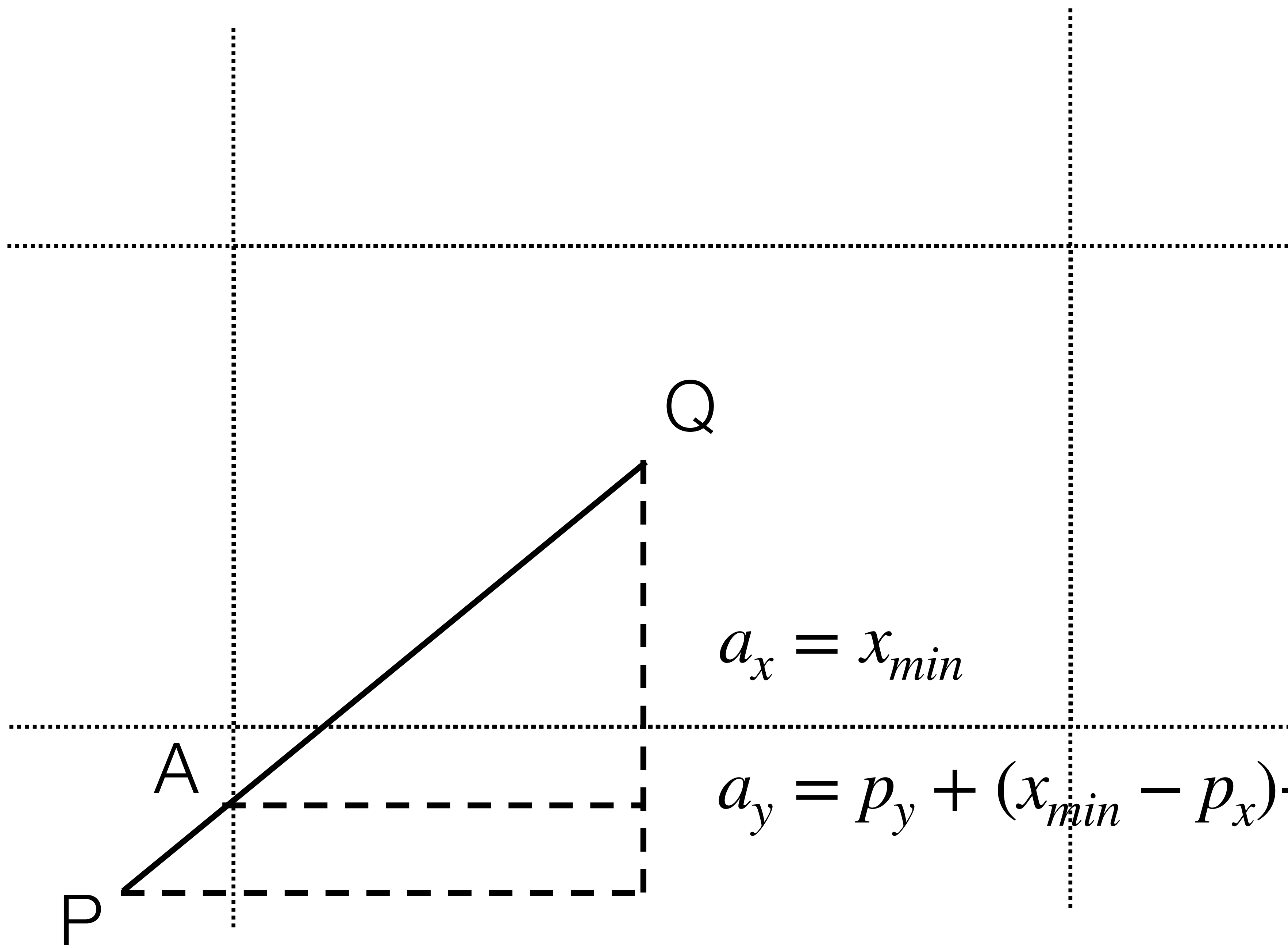
```
// cont...

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

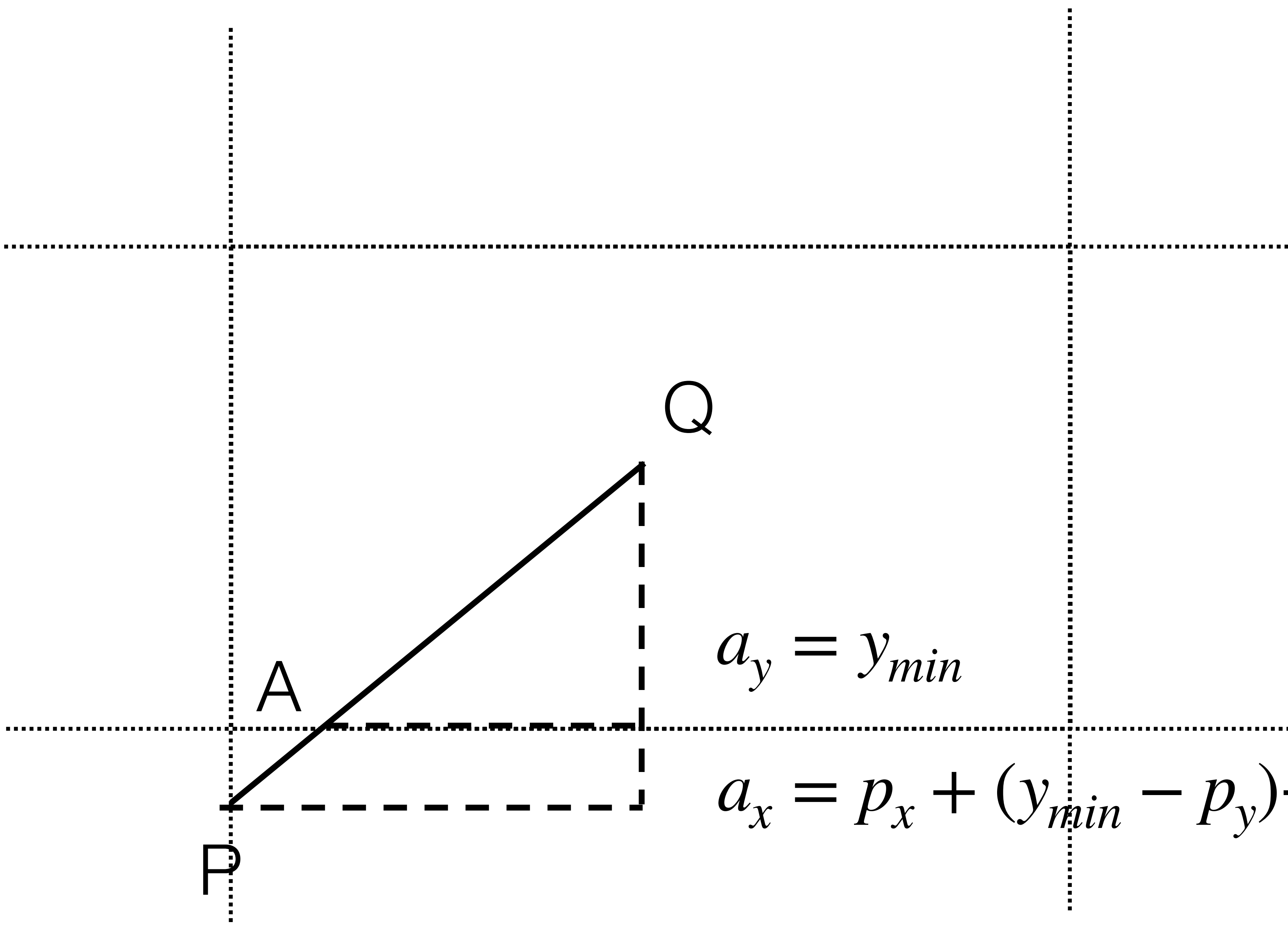
Using similar triangles:



$$a_x = x_{min}$$

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

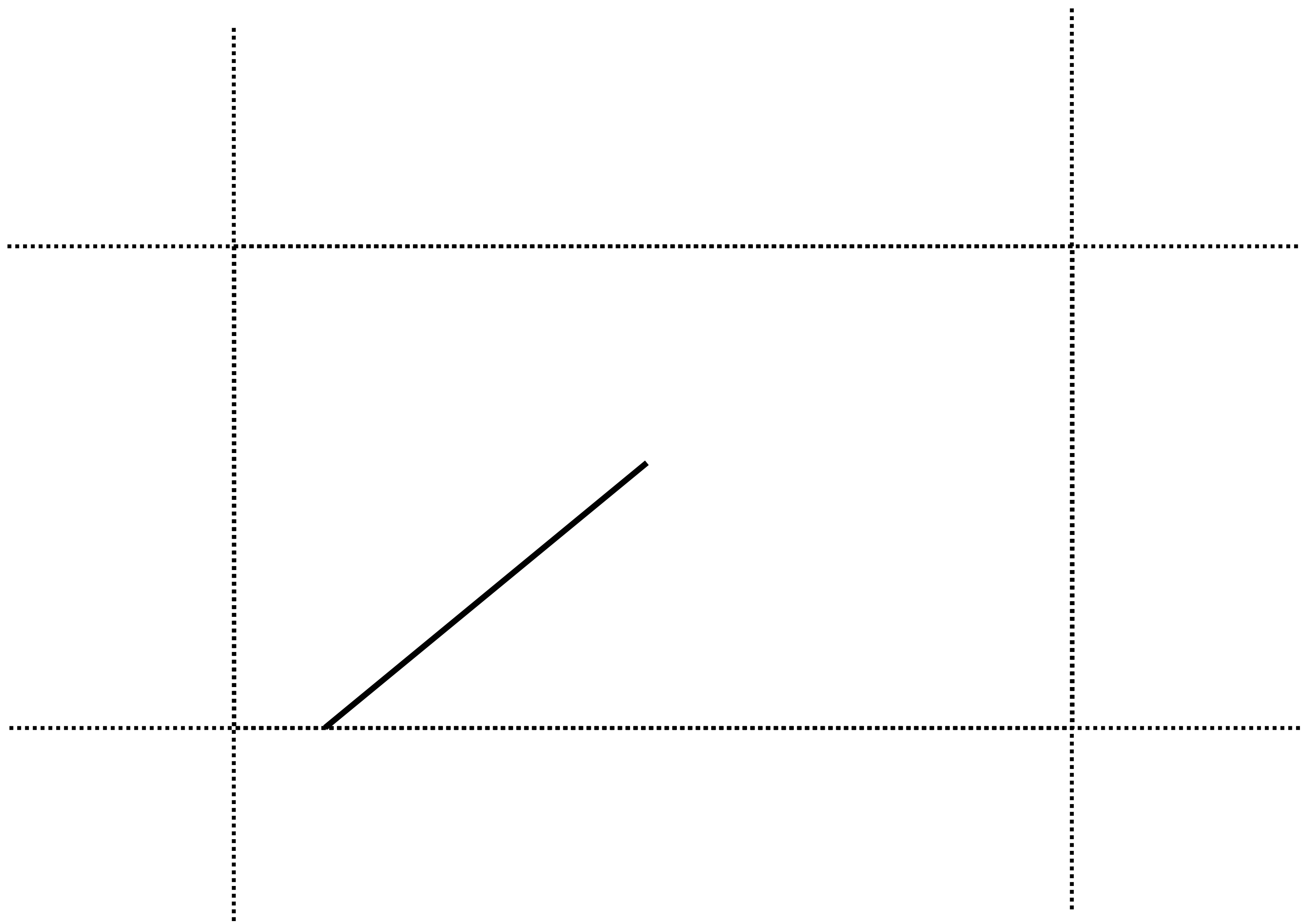
Clipping a point



$$a_y = y_{min}$$

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

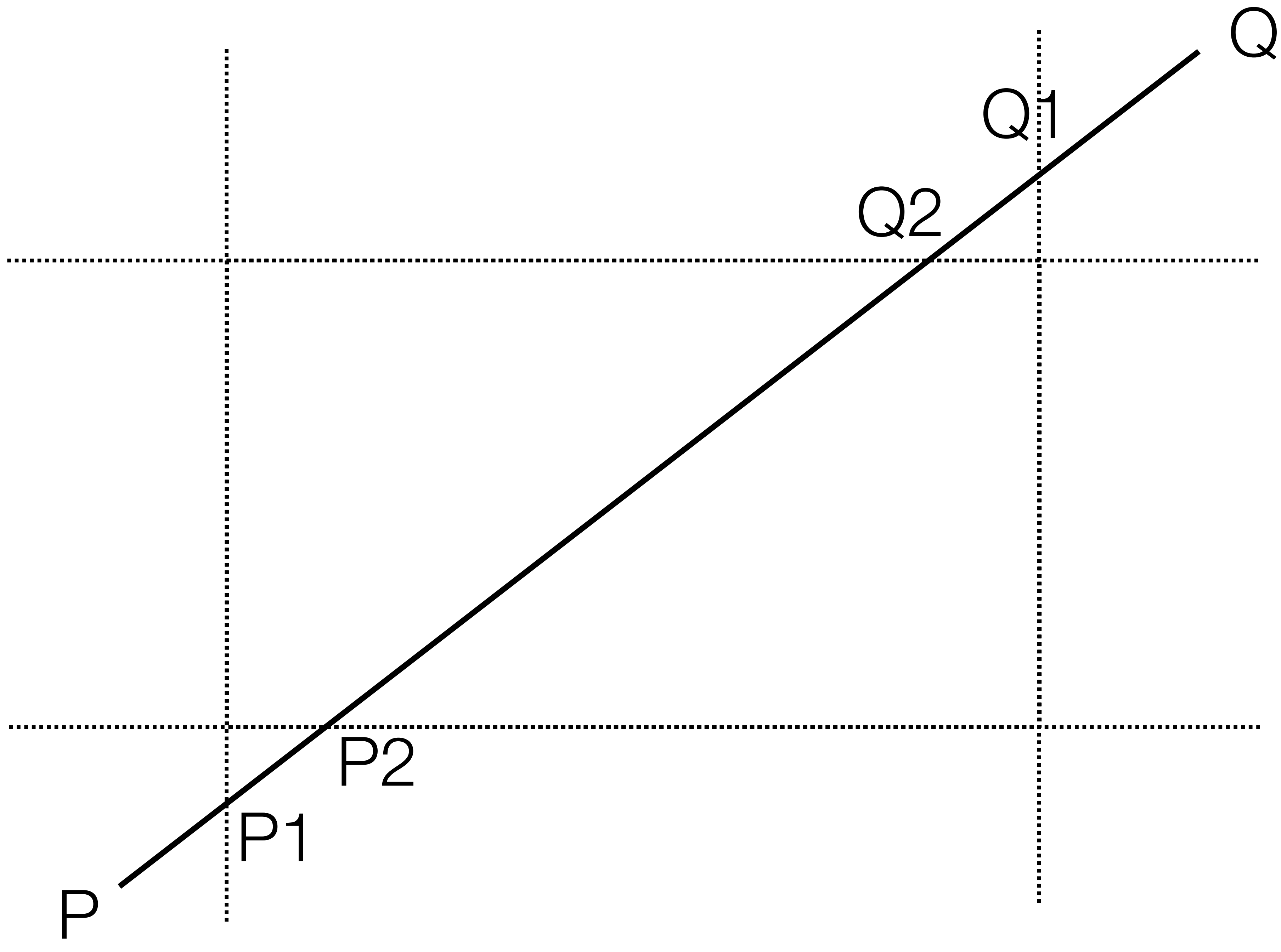
Clipping a point



Exercise

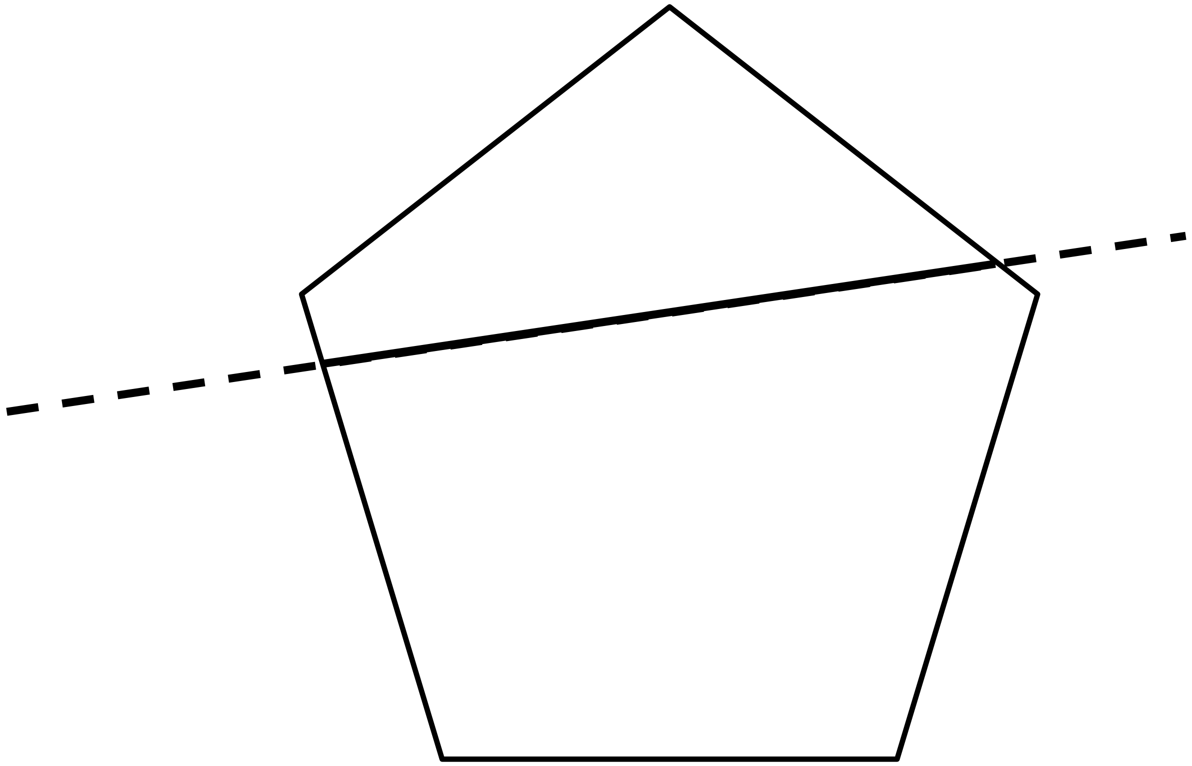
- Assuming a rectangle with bounds left=-1, right=1, bottom=-1, and top=1, clip the line from P=(-1.5,-2) to Q=(0,0).

Case needing 4 Clips



Cyrus Beck

- Clipping a line to a convex polygon.



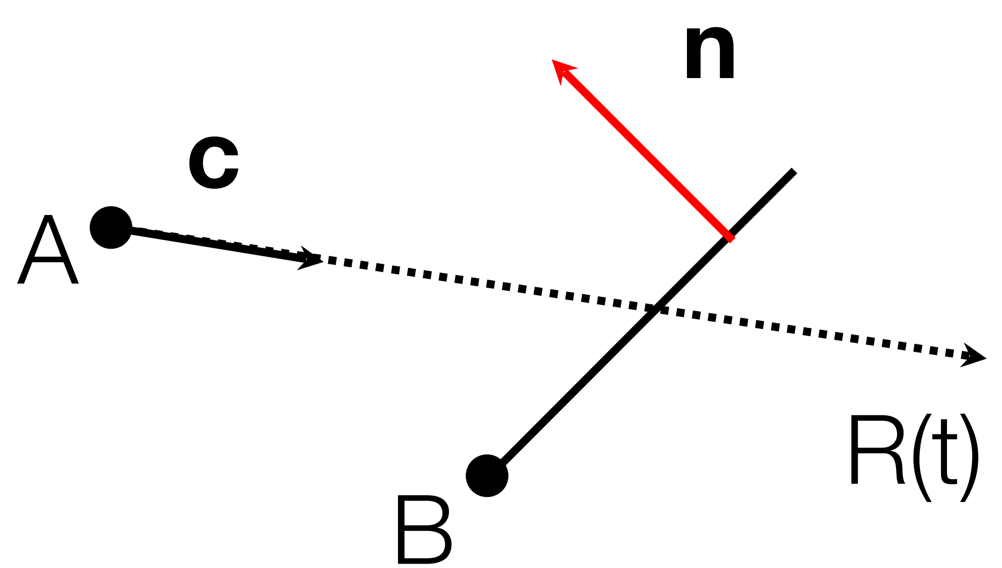
Ray colliding with segment

- 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

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

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

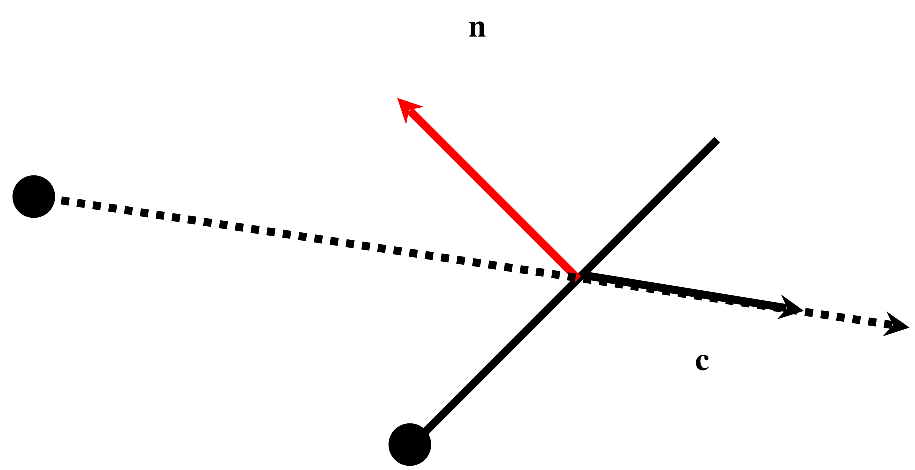
$$\mathbf{n} \cdot (A - B) + \mathbf{n} \cdot \mathbf{c}t_{hit} = 0$$

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

$$P_{hit} = A + \mathbf{c}t_{hit}$$

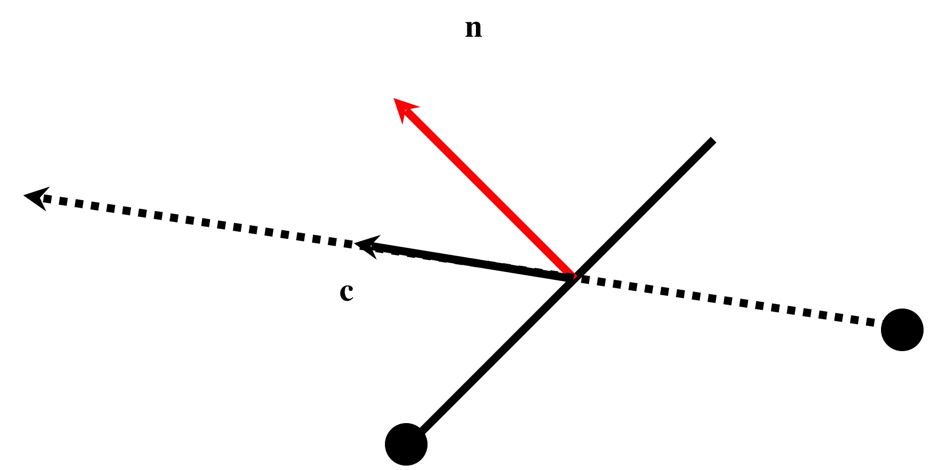
Entering / exiting

- Assuming all normals point out of the polygon:



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

entering



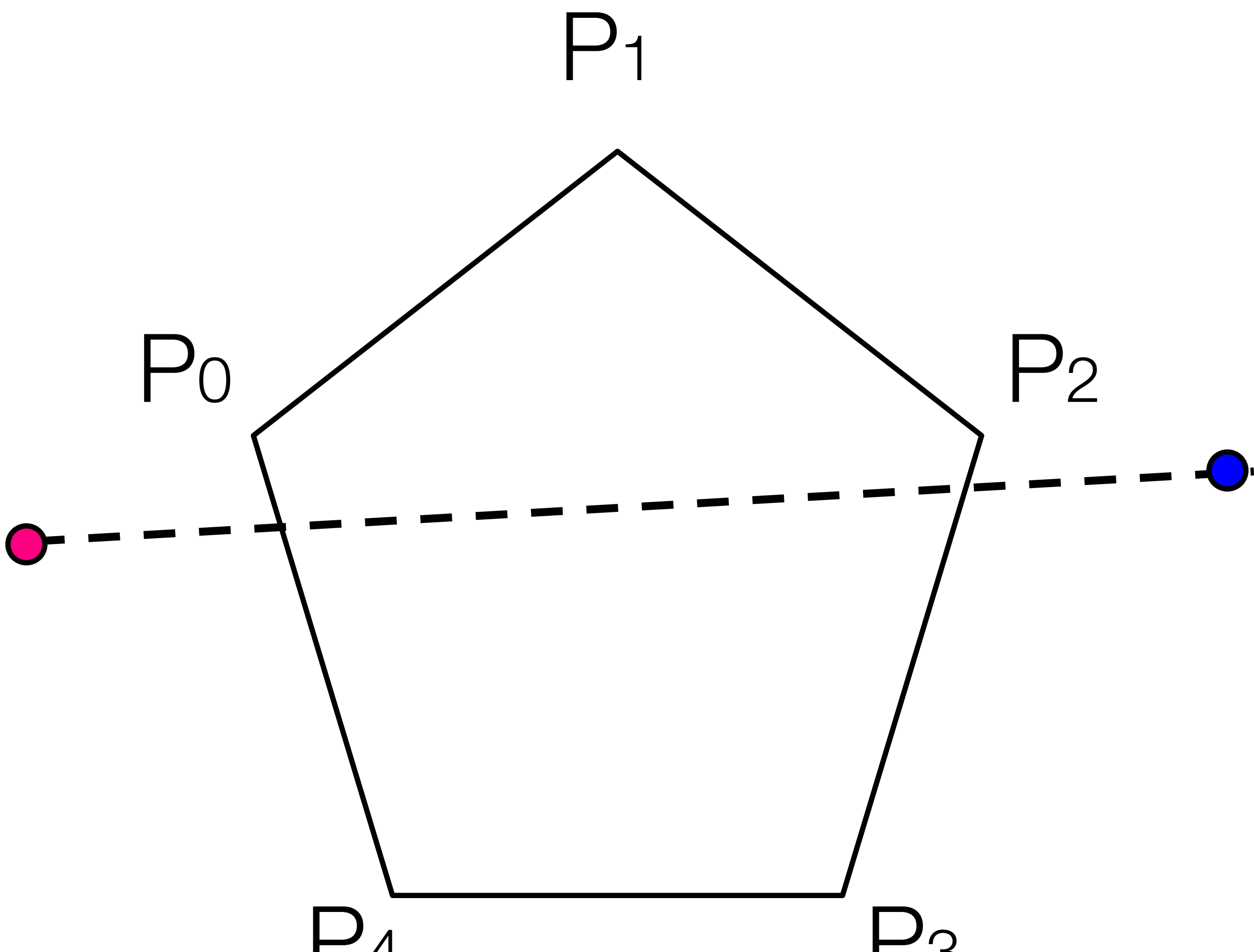
$$\mathbf{n} \cdot \mathbf{c} > 0$$

exiting

Cyrus-Beck

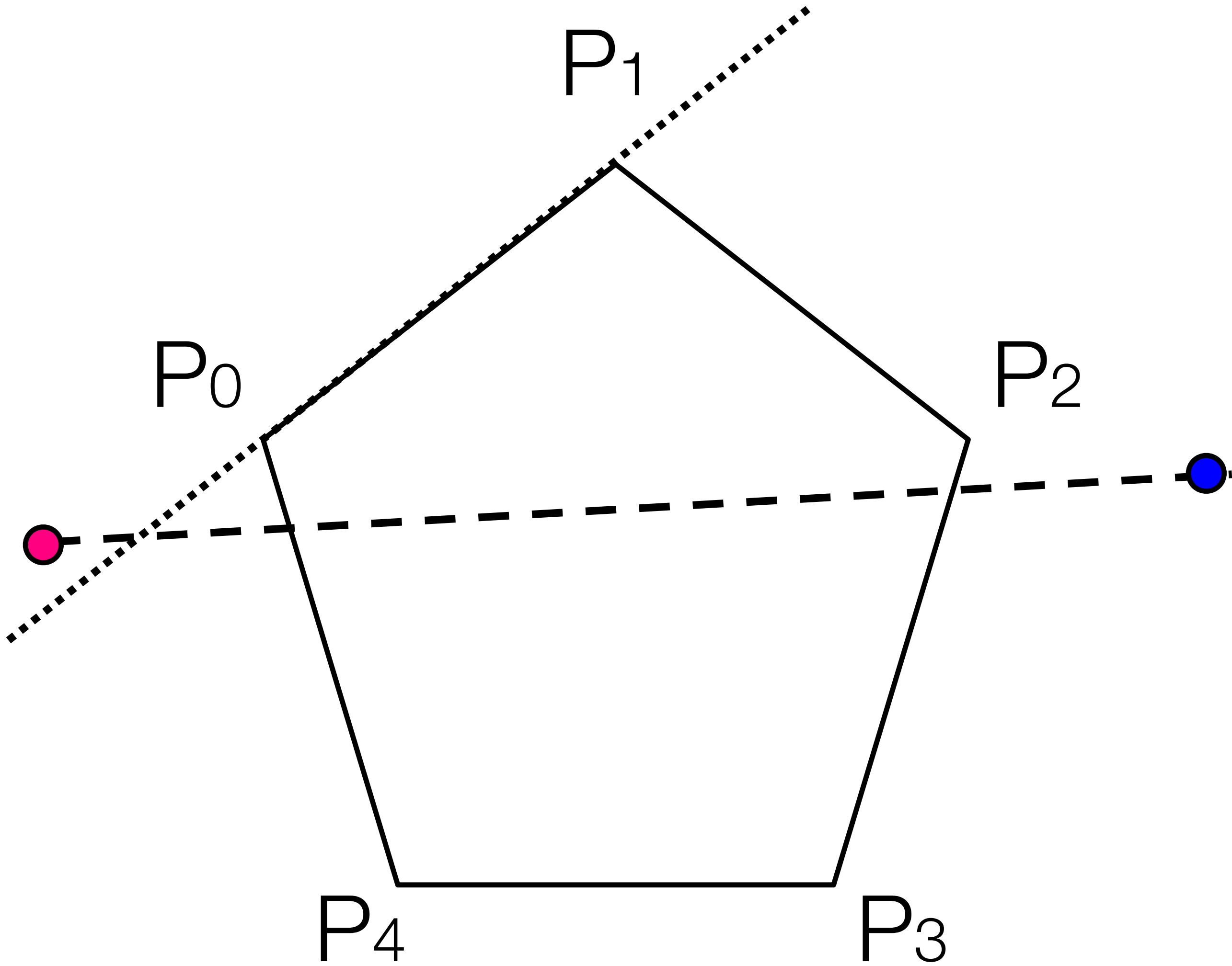
- Initialise t_{in} to 0 and t_{out} to 1
- Compare the ray to each edge of the (convex) polygon.
- Compute t_{hit} for each edge.
- Keep track of maximum t_{in}
- Keep track of minimum t_{out}.

Example



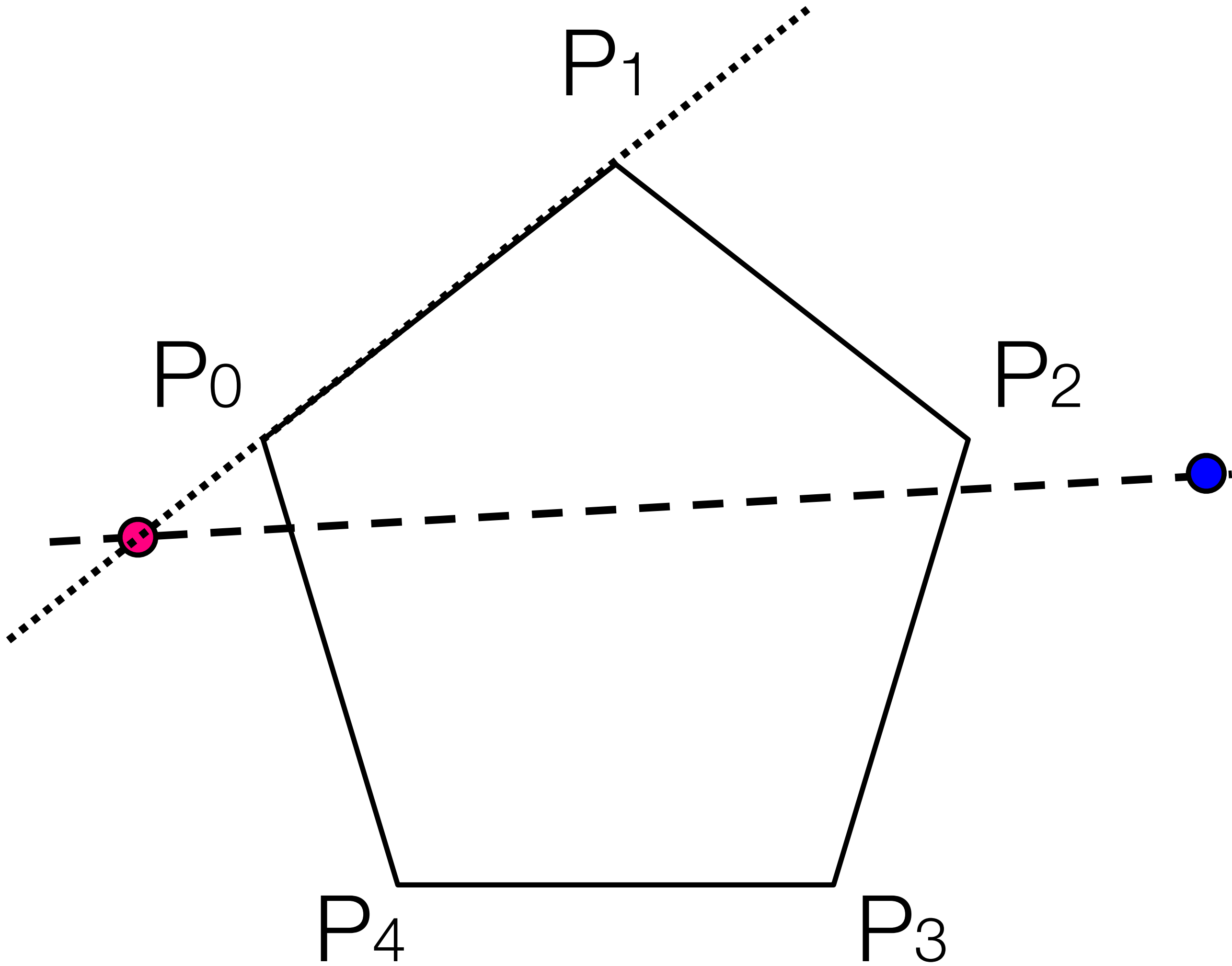
t_{in}	t_{out}
0	1

Example



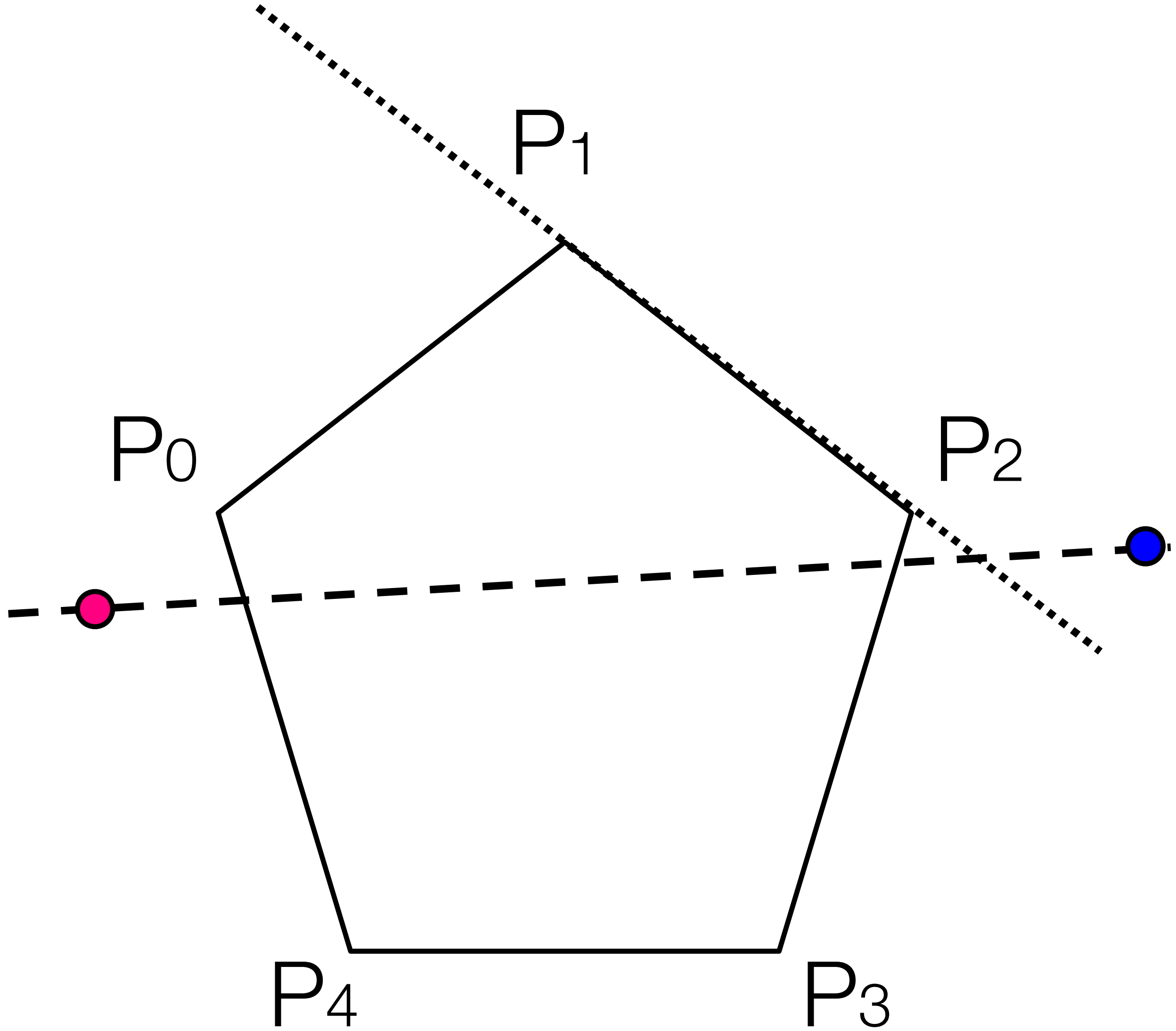
t_{in}	t_{out}
0	1

Example



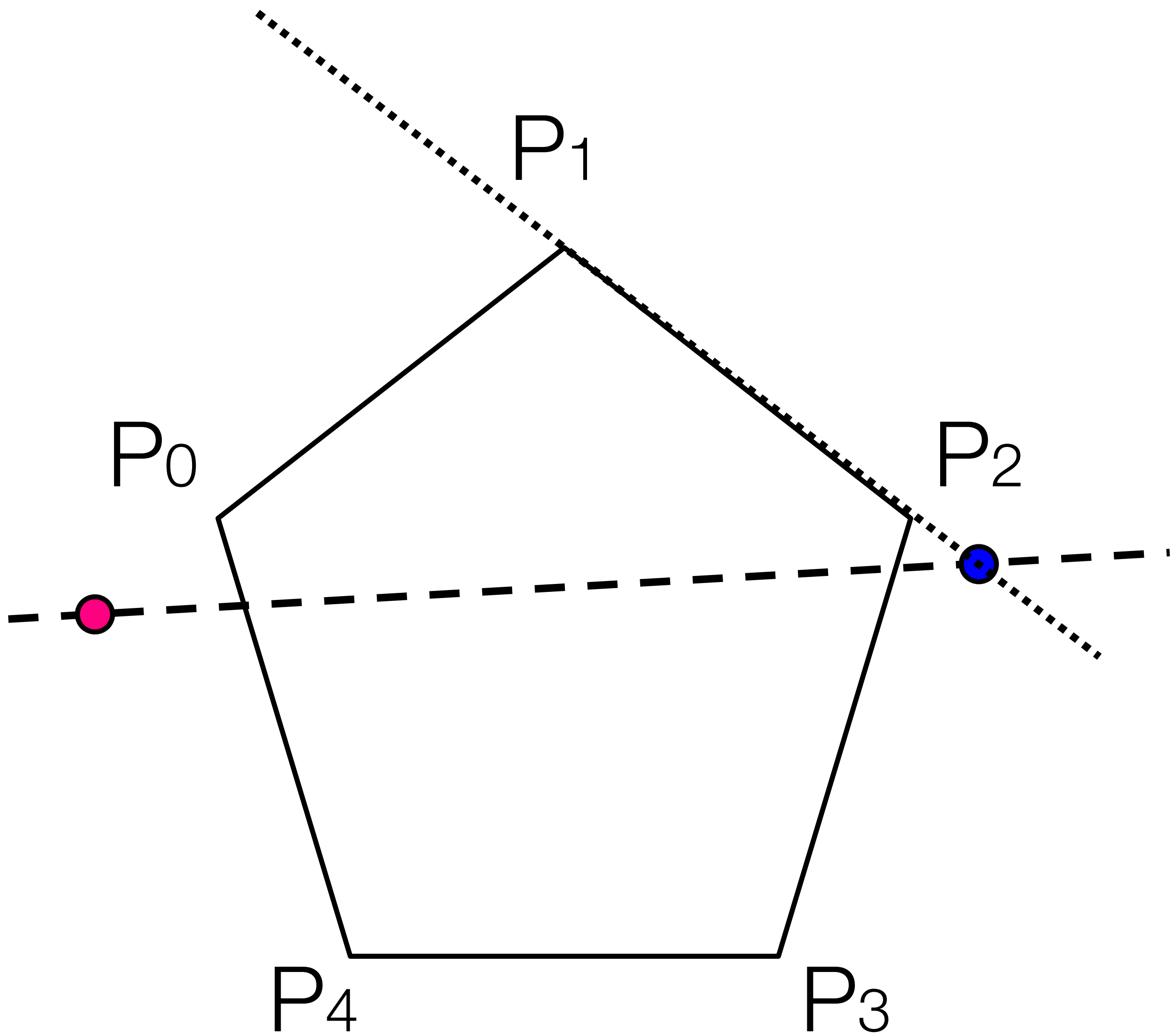
t_{in}	t_{out}
0	1
0.1	1

Example



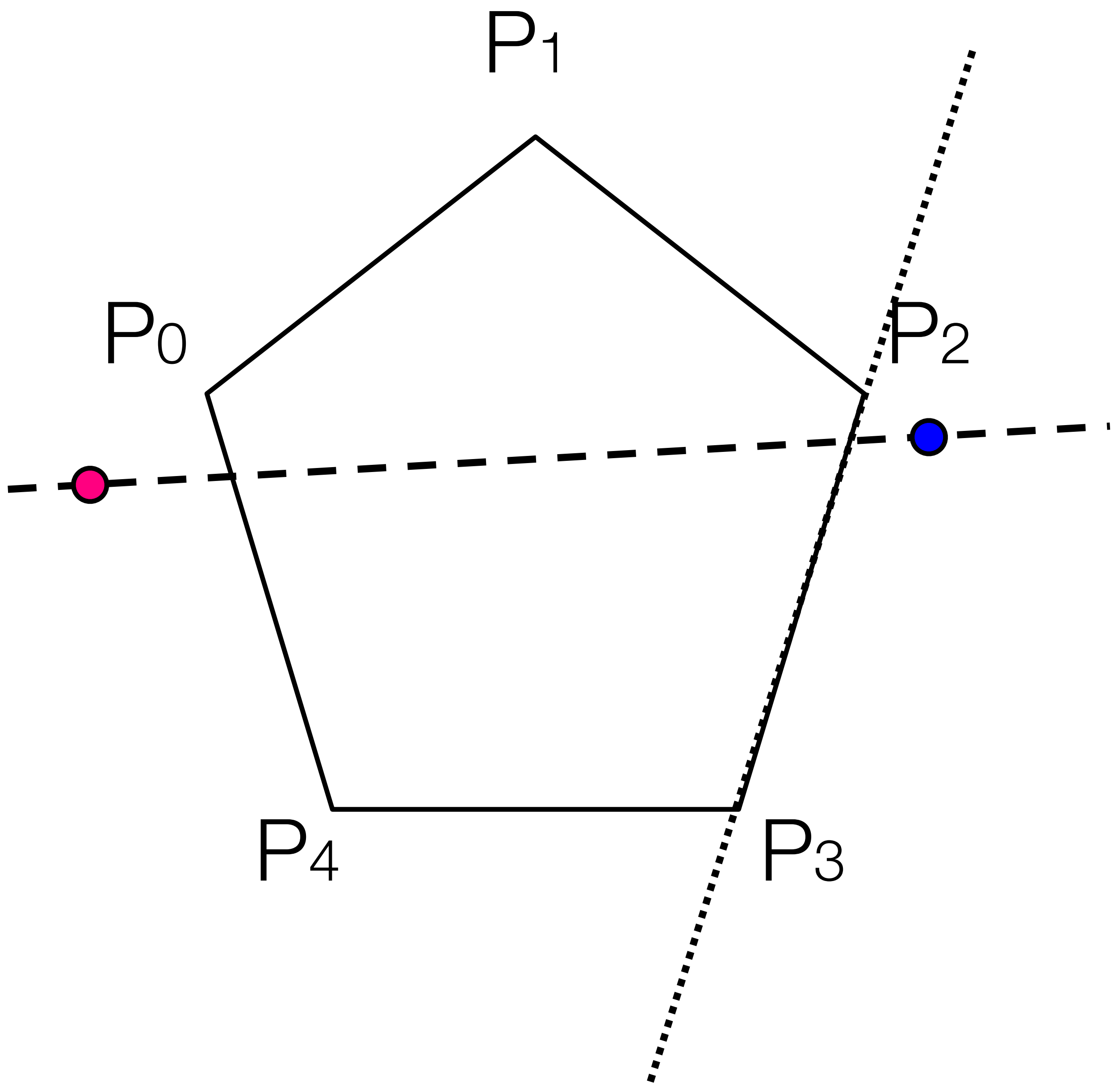
t_{in}	t_{out}
0	1
0.1	1

Example



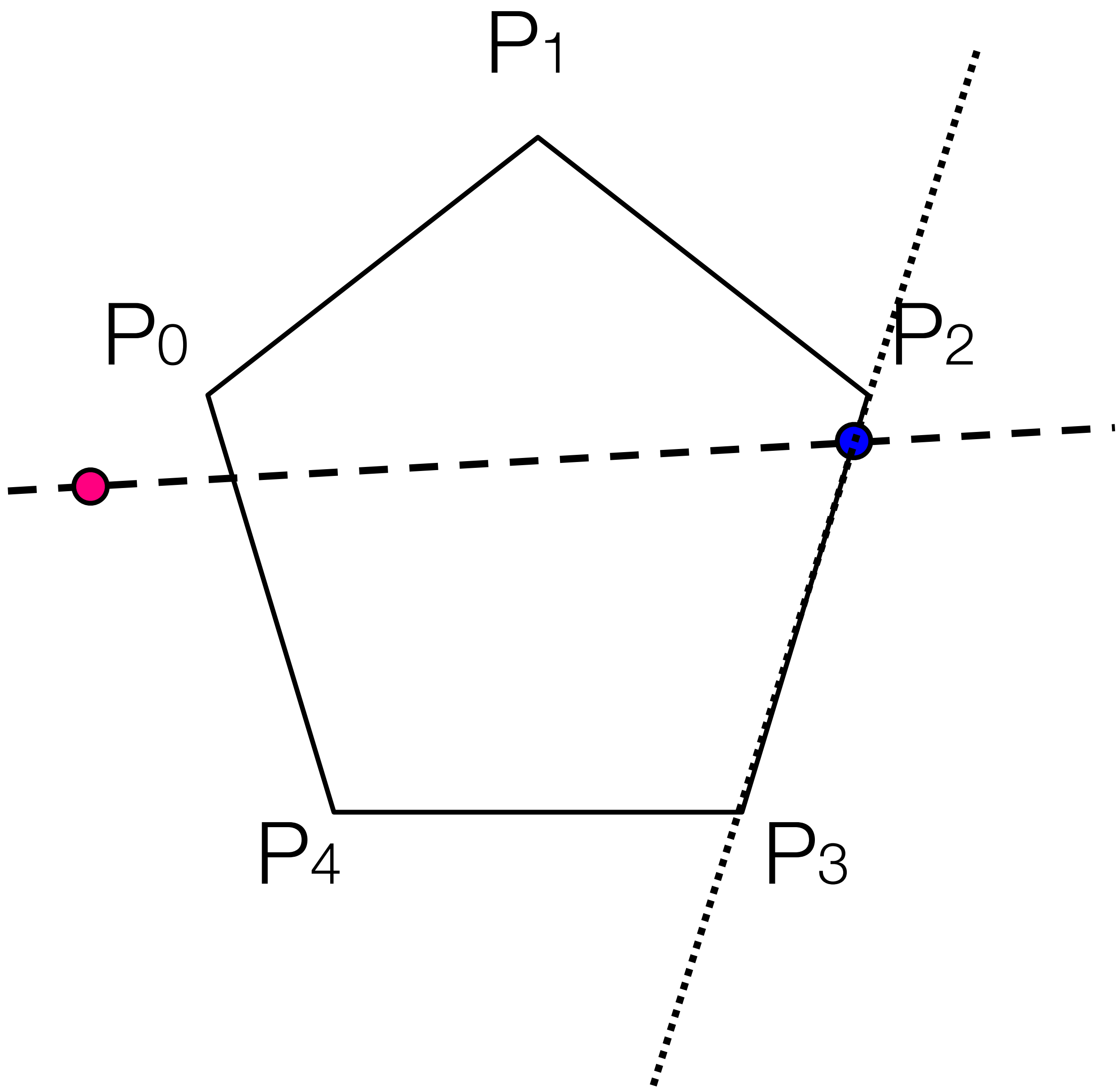
t_{in}	t_{out}
0	1
0.1	1
0.1	0.9

Example



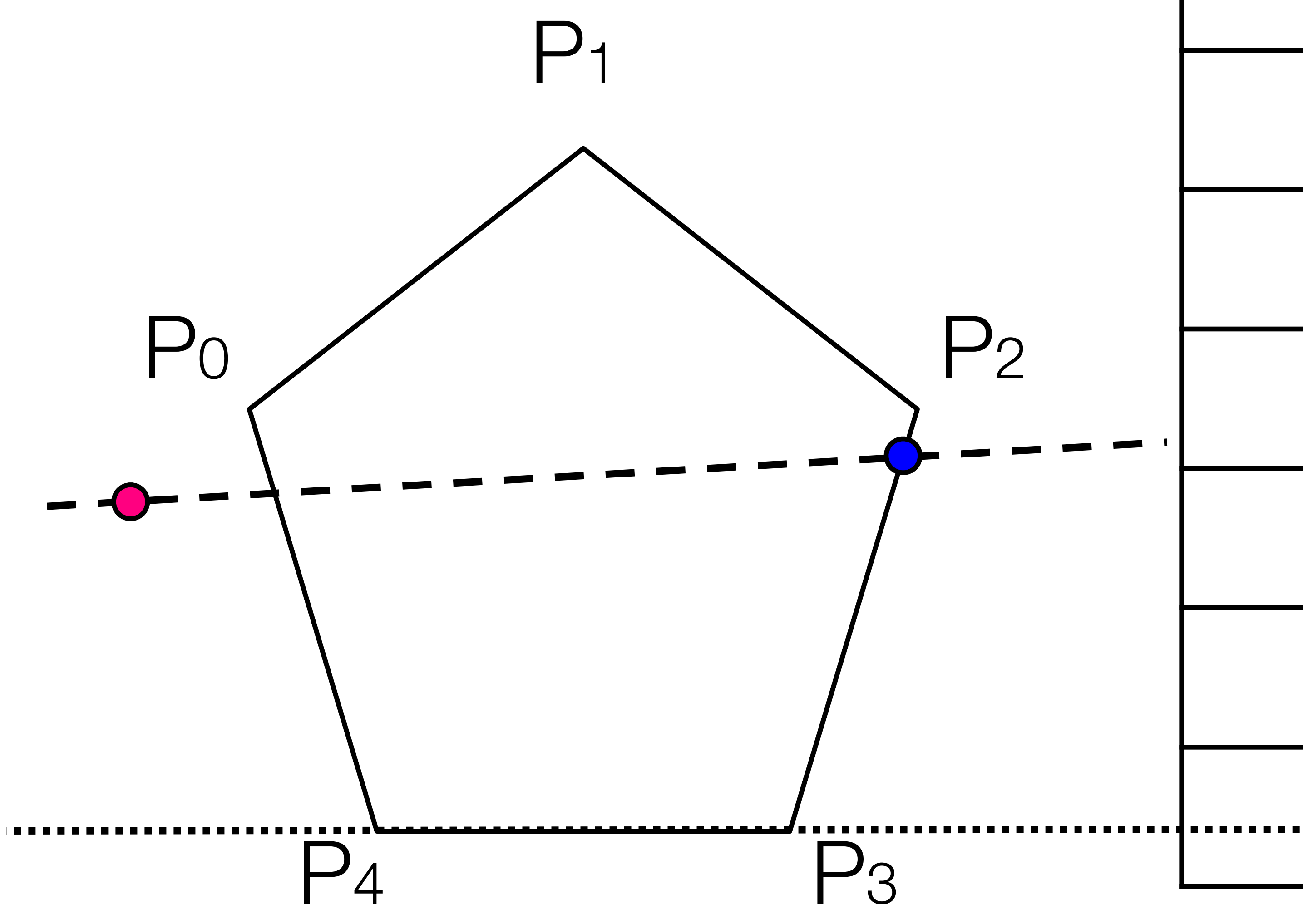
t_{in}	t_{out}
0	1
0.1	1
0.1	0.9

Example



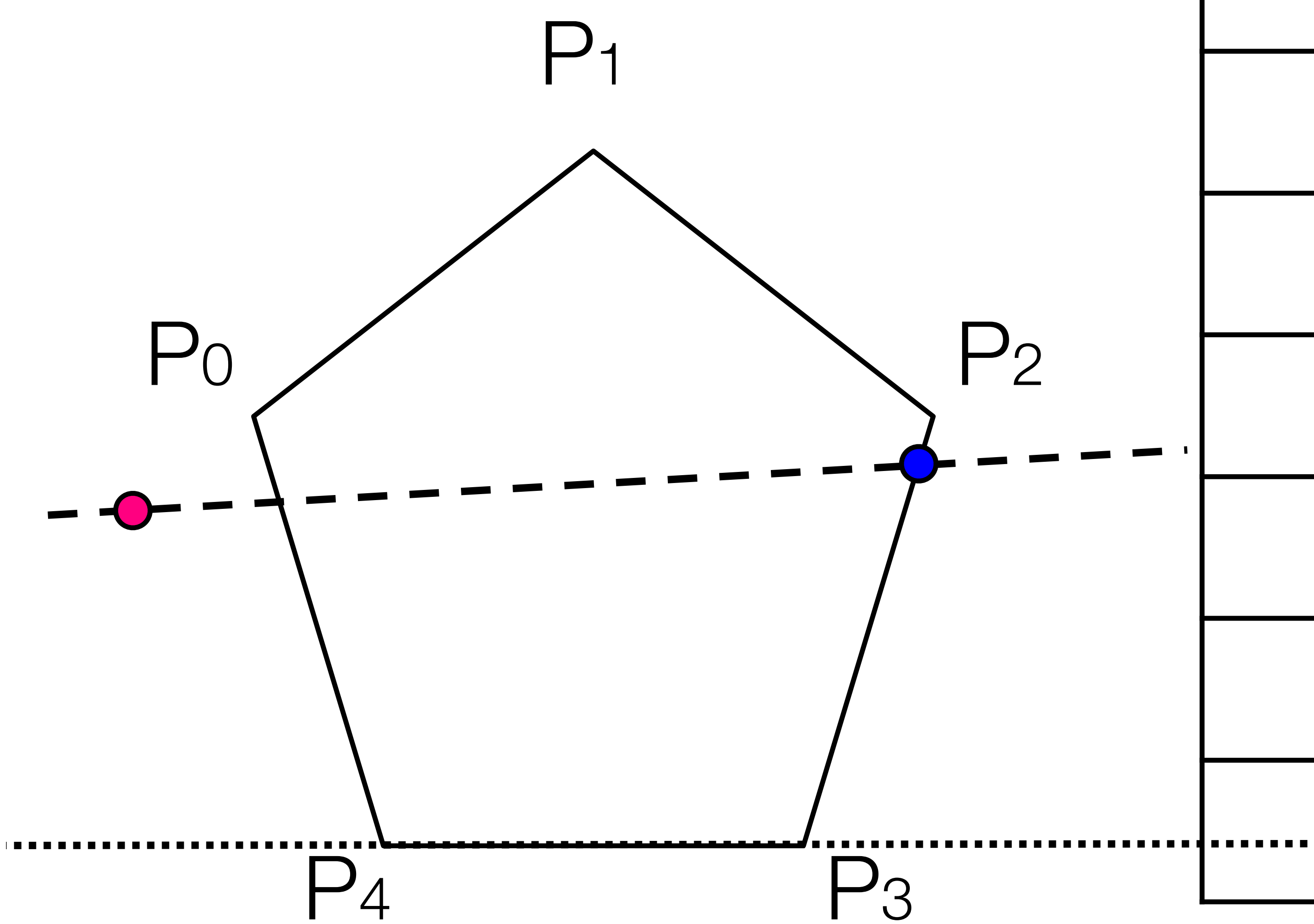
t_{in}	t_{out}
0	1
0.1	1
0.1	0.9
0.1	0.85

Example



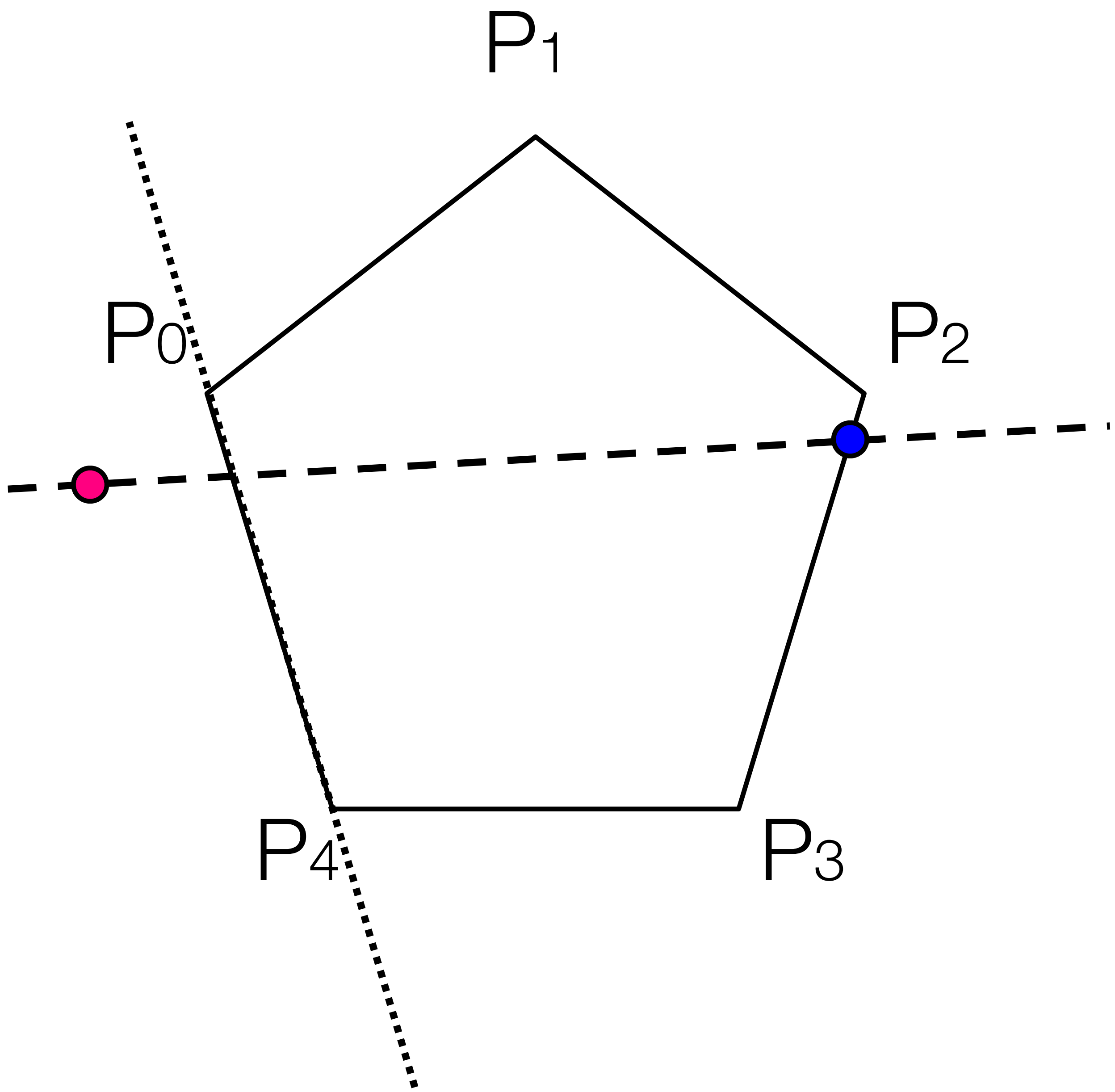
t_{in}	t_{out}
0	1
0.1	1
0.1	0.9
0.1	0.85

Example



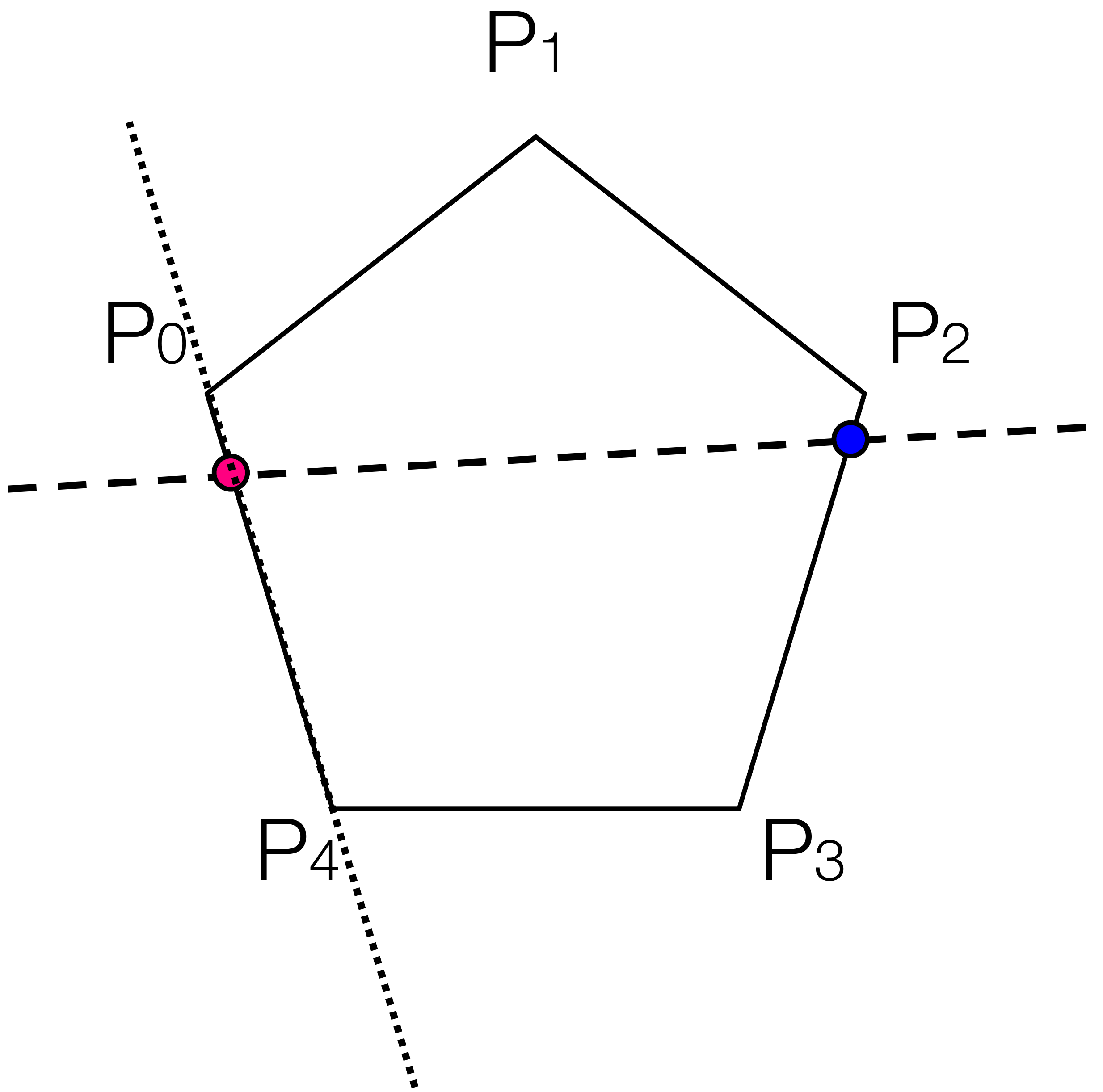
t_{in}	t_{out}
0	1
0.1	1
0.1	0.9
0.1	0.85
0.1	0.85

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