CSE4203: Computer Graphics

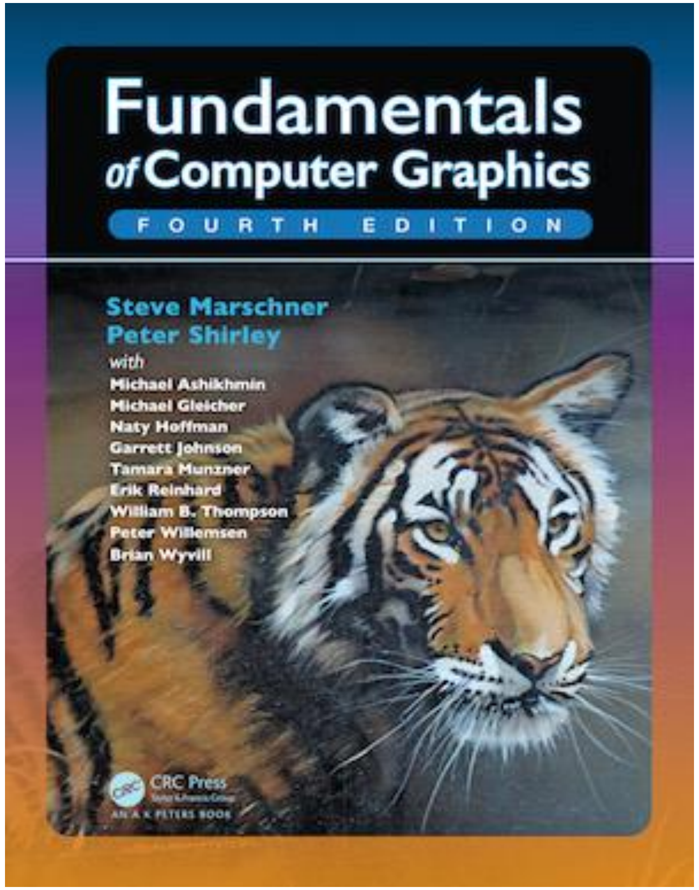
Chapter – 8 (part -D)

Graphics Pipeline

Outline

– Clipping
– Operations before and after rasterization

Credit



CS4620: Introduction to Computer Graphics

Cornell University

Instructor: Steve Marschner

http://www.cs.cornell.edu/courses/cs4620/2019fa/

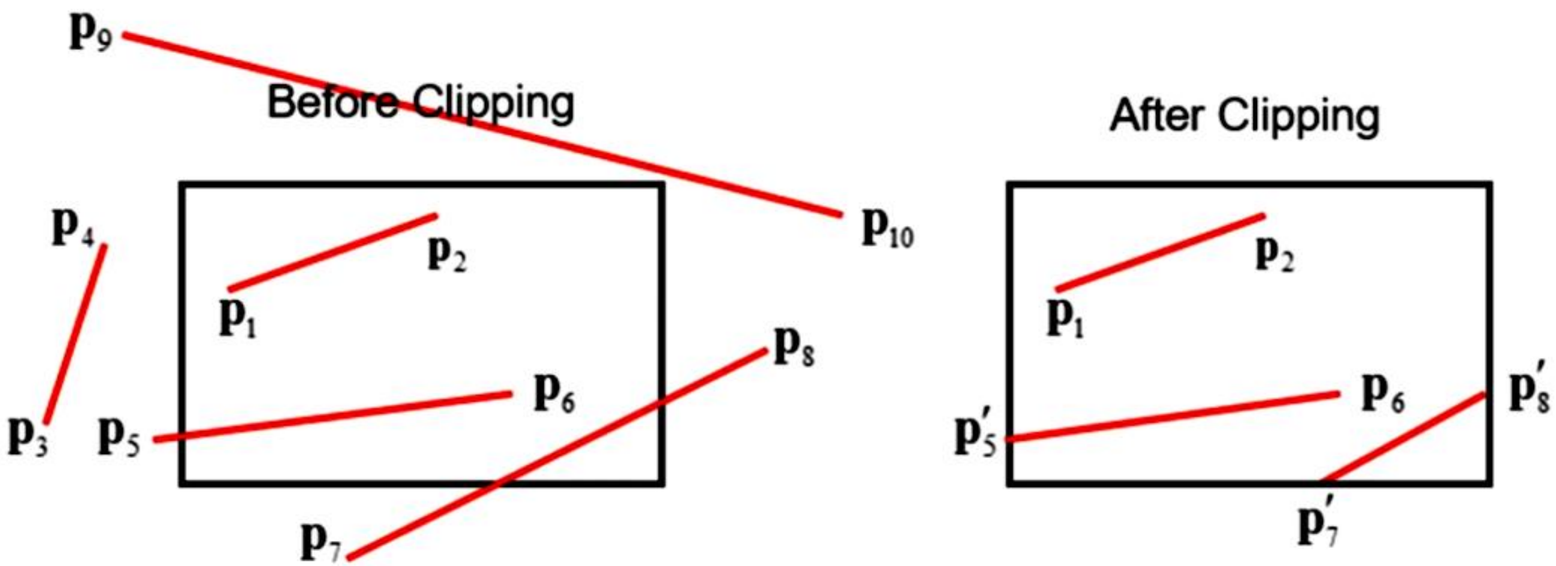
Clipping (1/2)

- ***Clipping*** is a method to selectively enable or disable rendering operations within a defined *region of interest*.
  - The primary use of clipping is to remove objects, lines, or line segments that are *outside the viewing pane*.

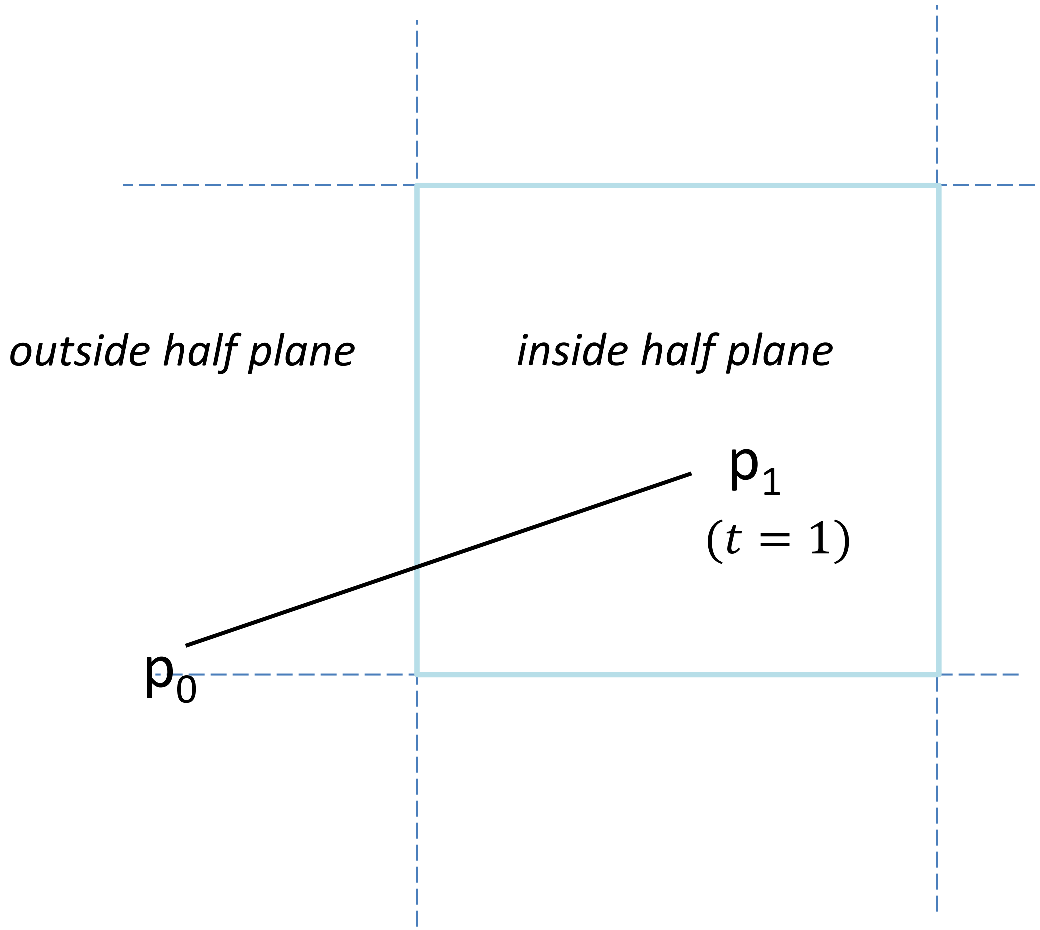
Line Clipping (2/2)

We must clip against a plane.

- ***Cyrus-Beck Parametric Line Clipping Algorithm***

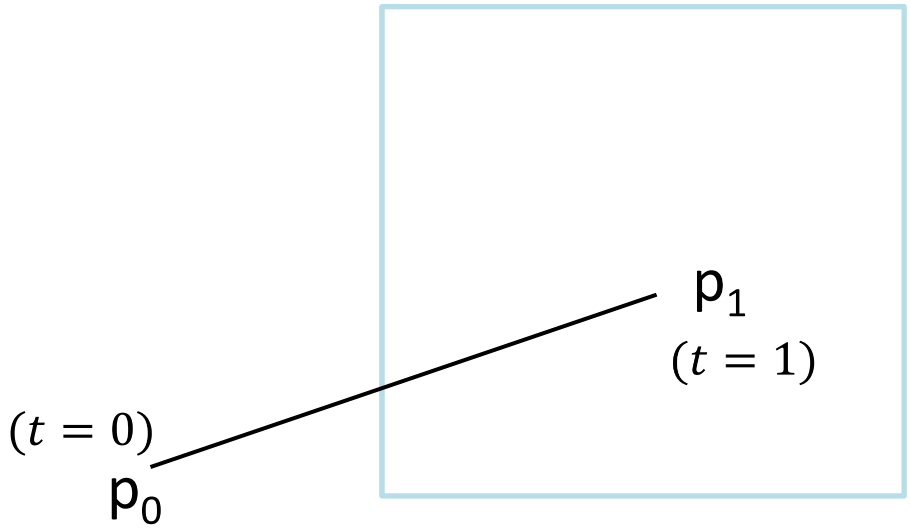


Inside/ outside of Half Plane (1/1)



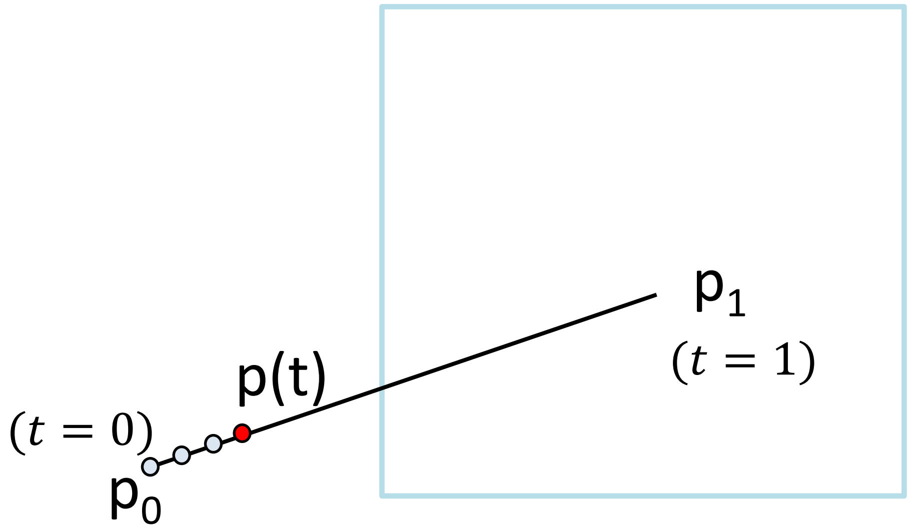
Parametric Eq. of a line (1/2)

$p(t) = p_0 + t(p_1 - p_0)$



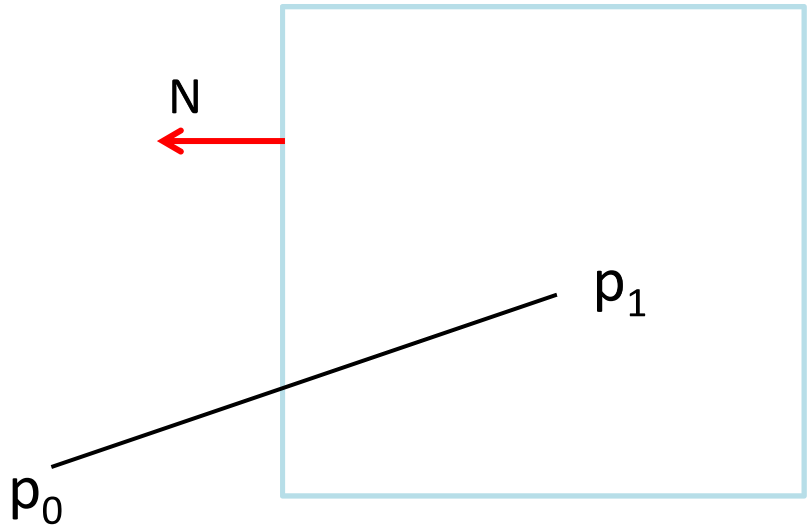
Parametric Eq. of a line (2/2)

$p(t) = p_0 + t(p_1 - p_0)$



Edge-line Intersection (1/7)

N = outward normal to the edge E

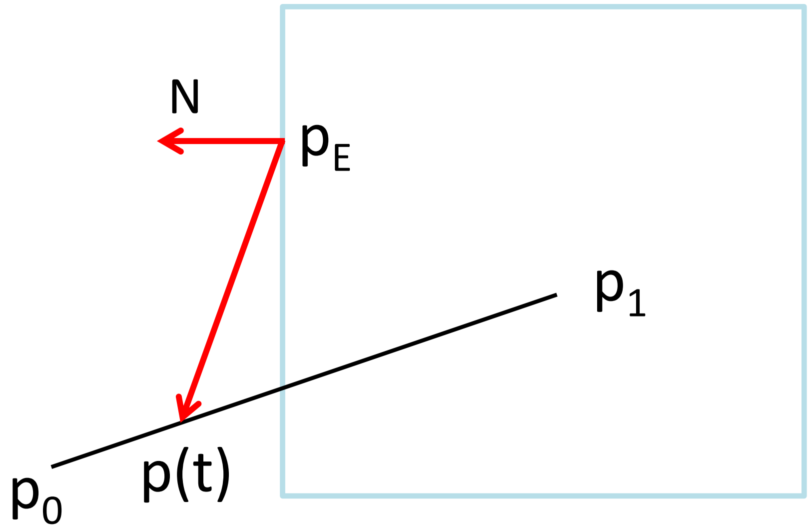


Edge-line Intersection (2/7)

N = outward normal to the edge E

p_E = any point to the edge E

$[\,p(t) - p_E]$ = vector from p_E to $p(t)$



Edge-line Intersection (3/7)

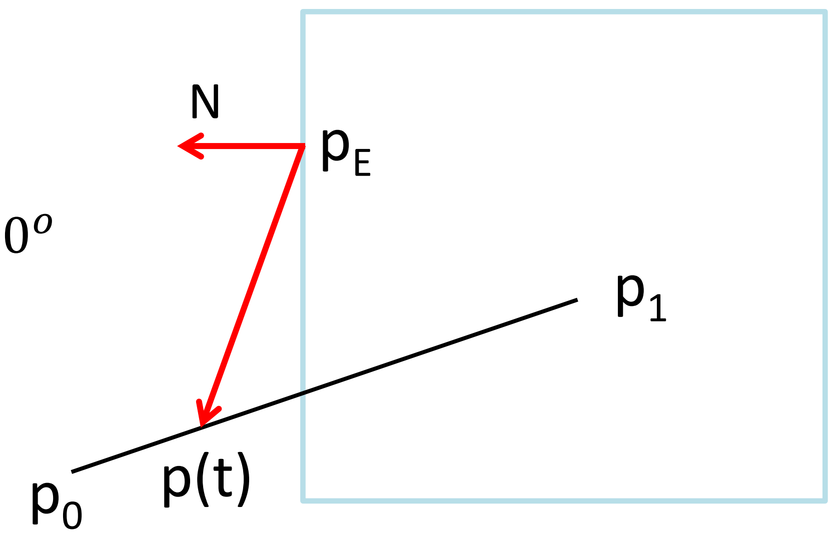
N = outward normal to the edge E

p_E = any point to the edge E

$[\,p(t) - p_E]$ = vector from p_E to $p(t)$

$N.[\,p(t) - p_E] > 0$

- Angel between$N$ and $[p(t) - p_E]$<90^o



Edge-line Intersection (4/7)

N = outward normal to the edge E

p_E = any point to the edge E

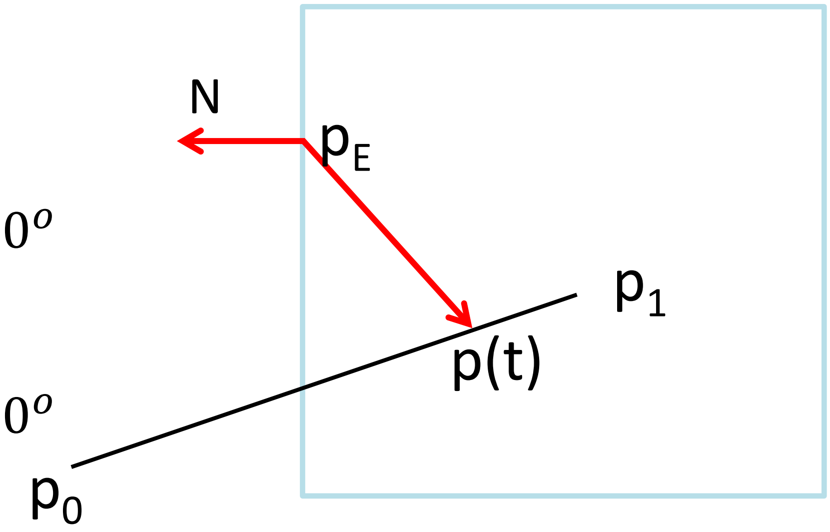
$[\, p(t) - p_E]$ = vector from p_E to $p(t)$

$N.[\, p(t) - p_E] > 0$

- Angel between$N$ and $[p(t) - p_E]$<90^o

$N.[\, p(t) - p_E] < 0$

- Angel between$N$ and $[p(t) - p_E]$>90^o



Edge-line Intersection (5/7)

N = outward normal to the edge E

p_E = any point to the edge E

$[\,p(t) - p_E]$ = vector from p_E to $p(t)$

$N.[\,p(t) - p_E] > 0$

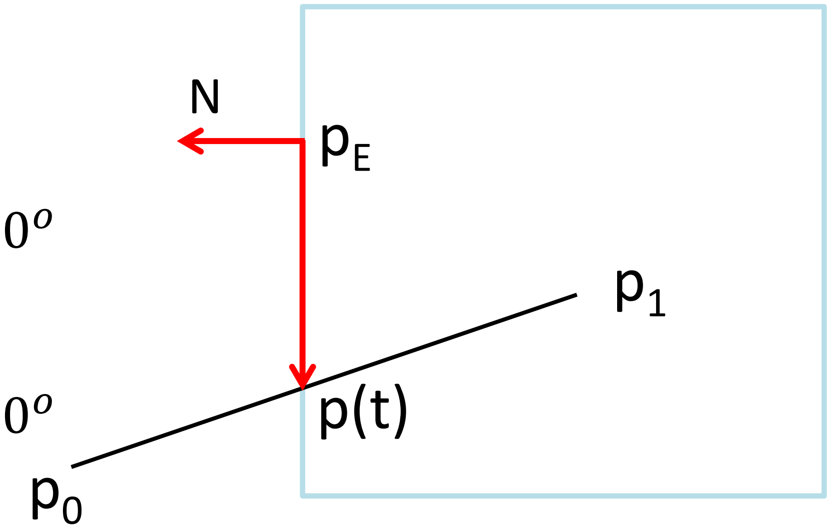
- Angel between$N$ and $[p(t) - p_E]$<90^o

$N.[\,p(t) - p_E] < 0$

- Angel between$N$ and $[p(t) - p_E]$>90^o

$N.[\,p(t) - p_E] = 0$

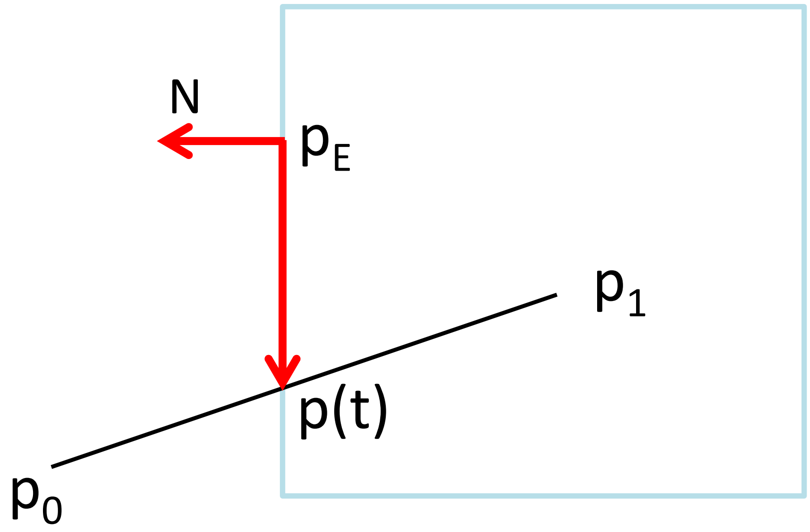
- Angel between$N$ and $[p(t) - p_E]$ = 90^o



Edge-line Intersection (6/7)

For intersection, $\boldsymbol{N}.\,[\,\boldsymbol{p}(\boldsymbol{t}) - \boldsymbol{p_e}] = \boldsymbol{0}$ ……… (1)

we know, $\boldsymbol{p}(\boldsymbol{t}) = \boldsymbol{p_0} + \boldsymbol{t}(\boldsymbol{p_1} \;-\; \boldsymbol{p_0})$



<u>Putting into Eq.(1):</u>

$$N.\,[\,p_0 + t(p_1 \;-\; p_0) - p_E\,] = 0$$

$$t = \frac{N.\,[p_0 - p_E]}{-N.\,[p_1 - p_0]}$$

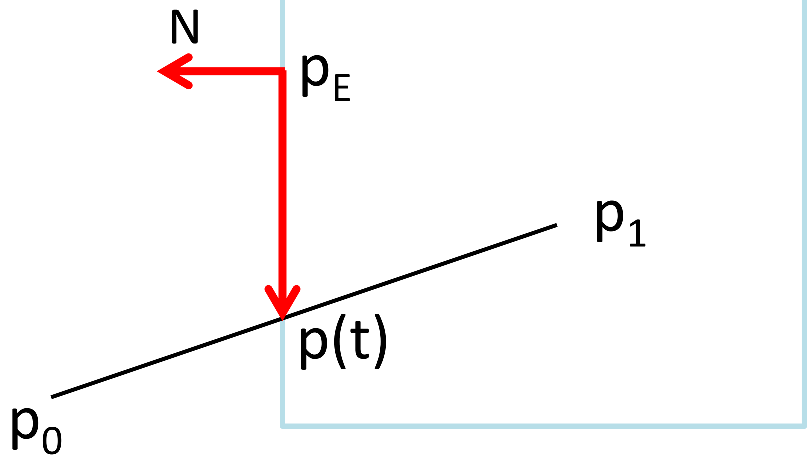
$$t = \frac{N.\,[p_0 - p_E]}{-N.\,D}$$

where, $D = p_1 - p_0$

Edge-line Intersection (7/7)

Therefore, *edge* and *line* are intersected at –

$$\boldsymbol{t} = \frac{\boldsymbol{N}.\,[\boldsymbol{p_0} - \boldsymbol{p_E}]}{-\boldsymbol{N}.\,\boldsymbol{D}} \quad \text{where, } D = p_1 - p_0$$



Check for Nonzero (1/2)

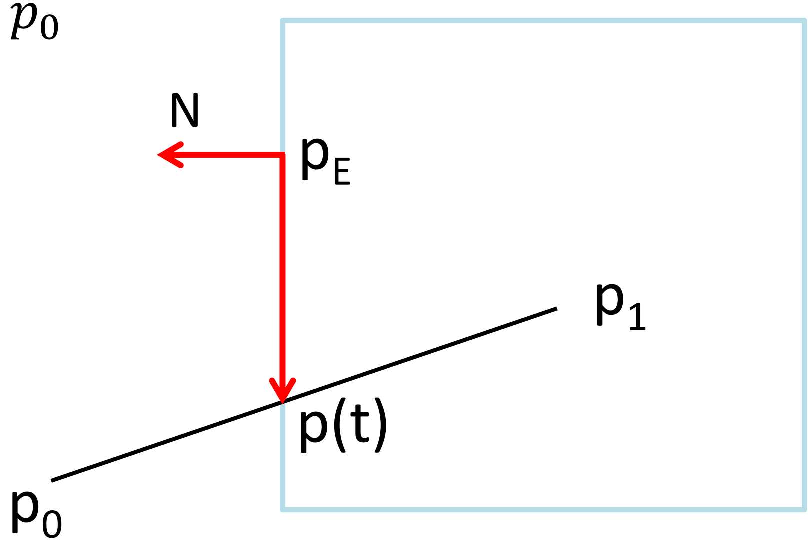
Therefore, *edge* and *line* are intersected at –

$$\boldsymbol{t} = \frac{\boldsymbol{N}.[\boldsymbol{p_0} - \boldsymbol{p_E}]}{-\boldsymbol{N}.\boldsymbol{D}} \quad \text{where, } D = p_1 - p_0$$

However, $N.D$ can not be zero.

We need to check –

- $\boldsymbol{N} \neq \boldsymbol{0}$ (by mistake, normal should not be 0)
- $\boldsymbol{D} \neq \boldsymbol{0}$ (means what?)
- $\boldsymbol{N}.\boldsymbol{D} \neq \boldsymbol{0}$ (means what?)



Check for Nonzero (2/2)

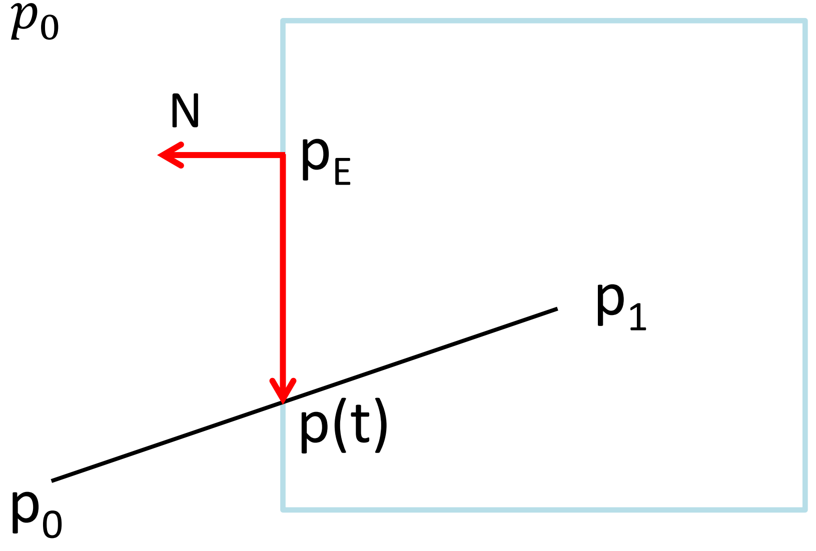
Therefore, *edge* and *line* are intersected at –

$$\boldsymbol{t} = \frac{\boldsymbol{N}.[\boldsymbol{p_0} - \boldsymbol{p_E}]}{-\boldsymbol{N}.\boldsymbol{D}} \quad \text{where, } D = p_1 - p_0$$

However, $N.D$ can not be zero.

We need to check –

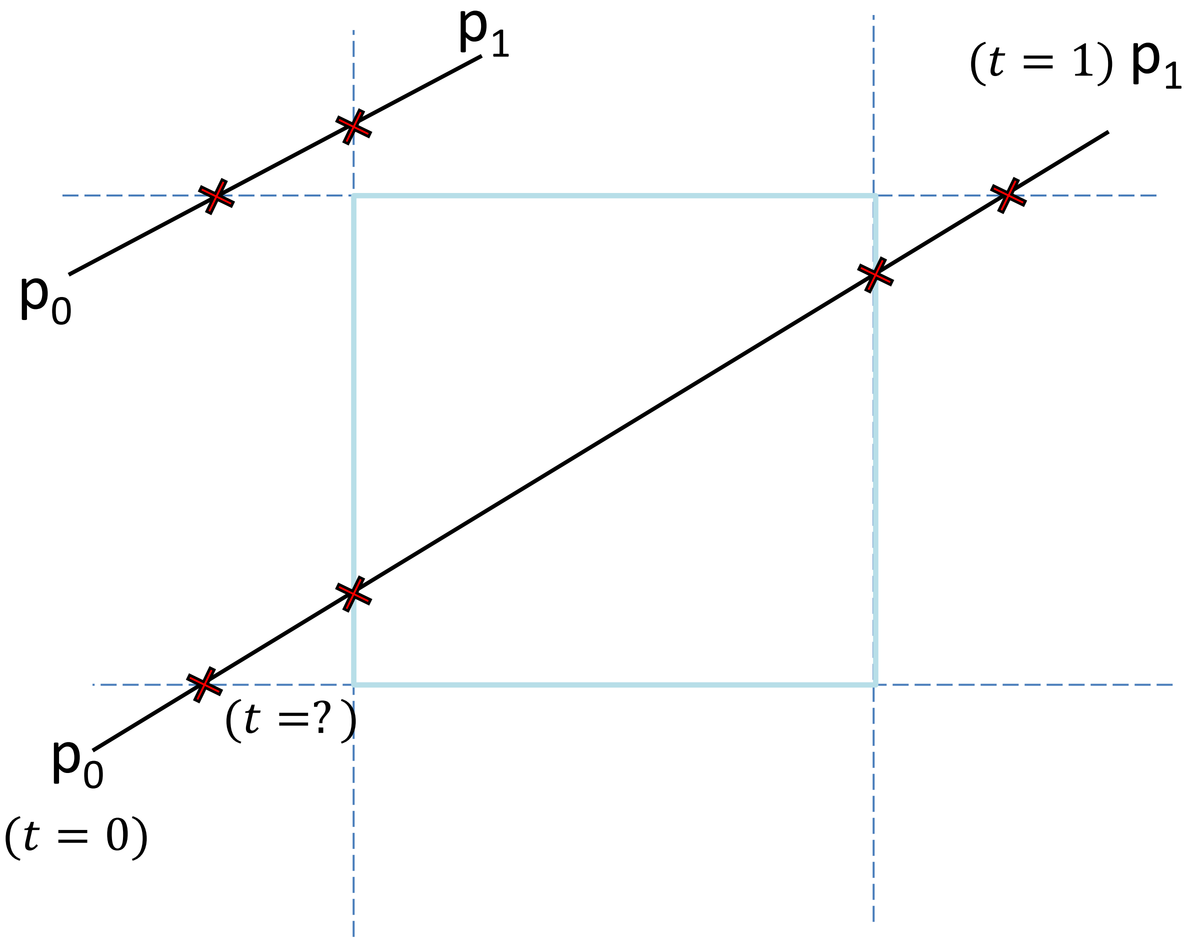
- $\boldsymbol{N} \neq \boldsymbol{0}$ (by mistake, normal should not be 0)
- $\boldsymbol{D} \neq \boldsymbol{0}$ (that is $p_1 \neq p_0$ for a line)
- $\boldsymbol{N}.\boldsymbol{D} \neq \boldsymbol{0}$ (line and the normal are not perpendicular; *line and edge are parallel*)



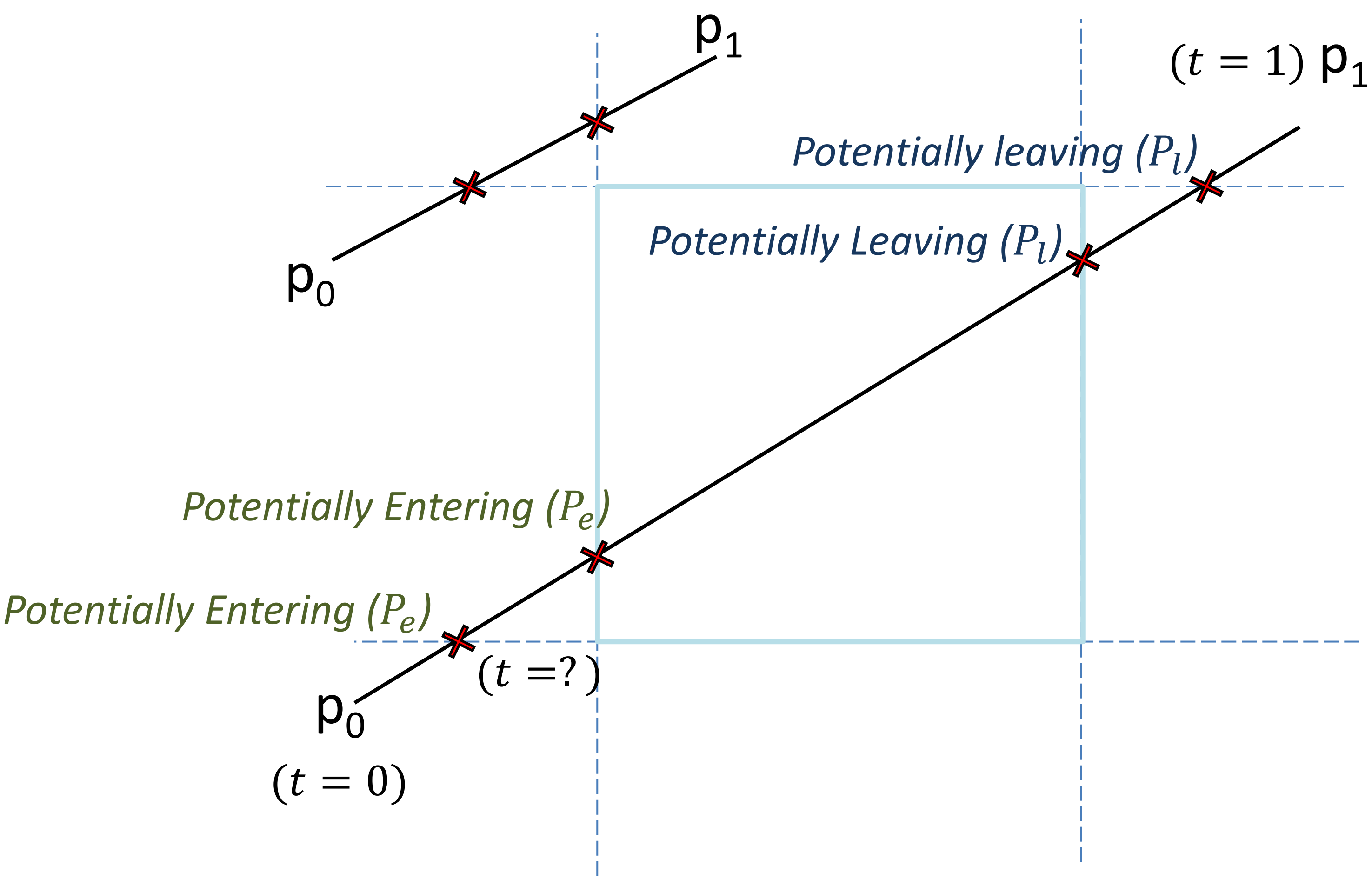
Inside/ outside Half Plane (1/1)

$$t = \frac{N.[p_0 - p_E]}{-N.D}$$

Only this formula is not enough! **Why?**

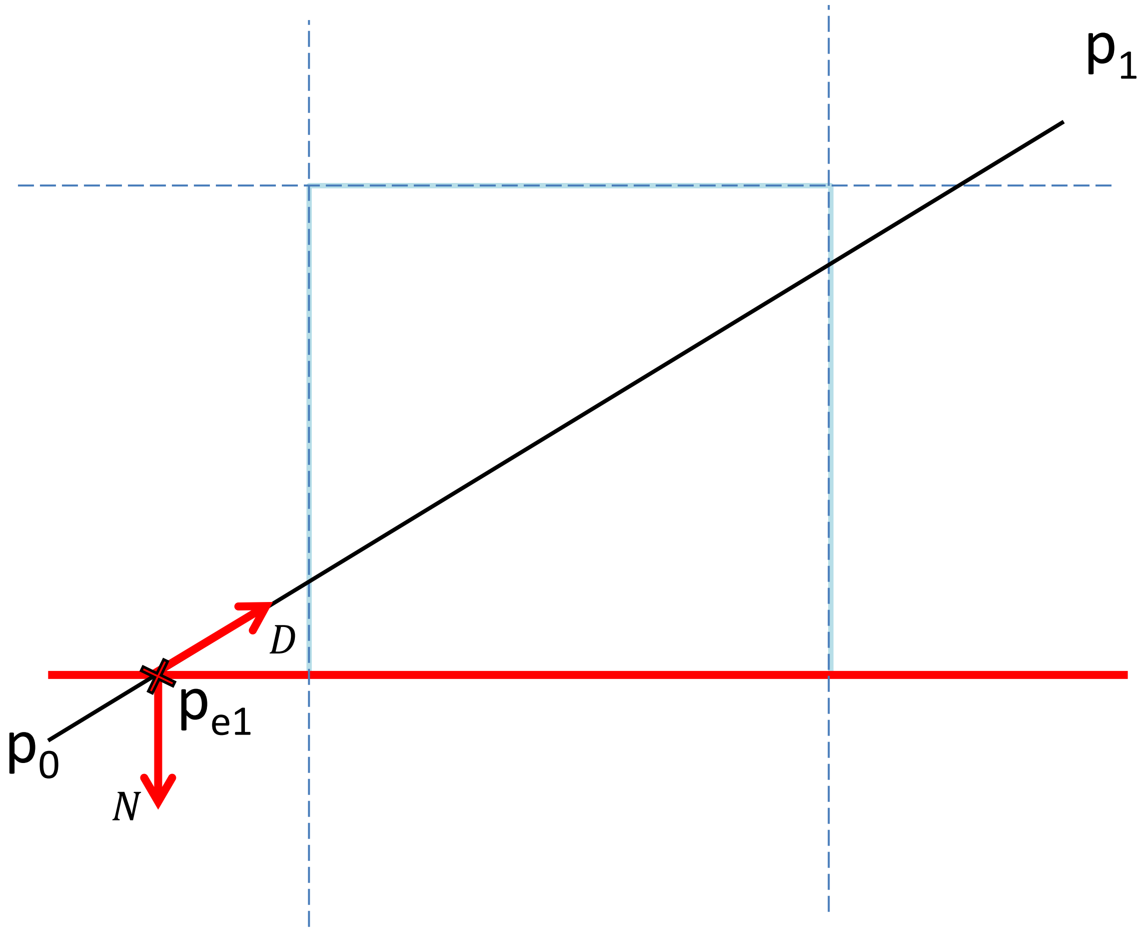


Potentially Entering/ Leaving (1/1)



True Clipping Intersection (1/12)

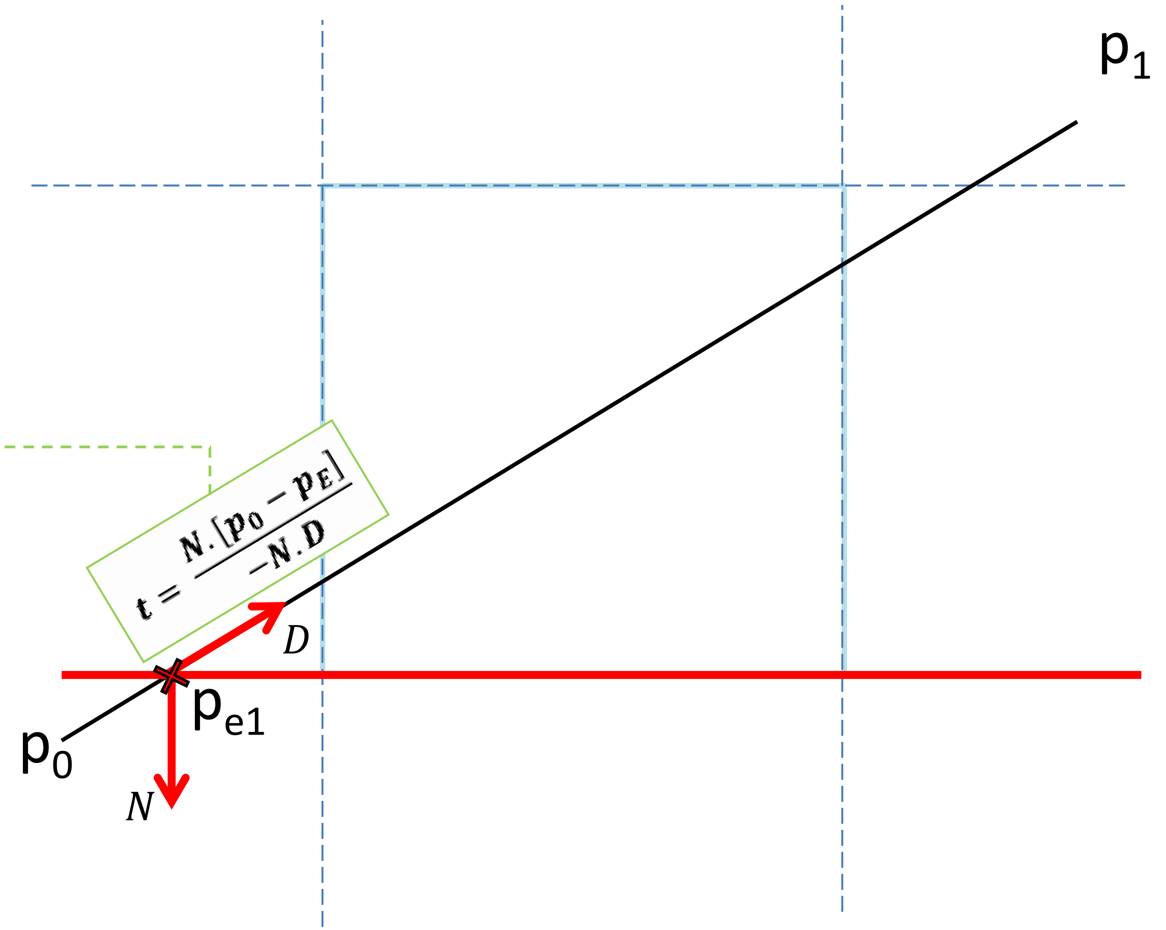
- $N.D < 0 \rightarrow P_e$
- $N.D > 0 \rightarrow P_l$



True Clipping Intersection (2/12)

- $N.D < 0 \rightarrow P_e$
- $N.D > 0 \rightarrow P_l$

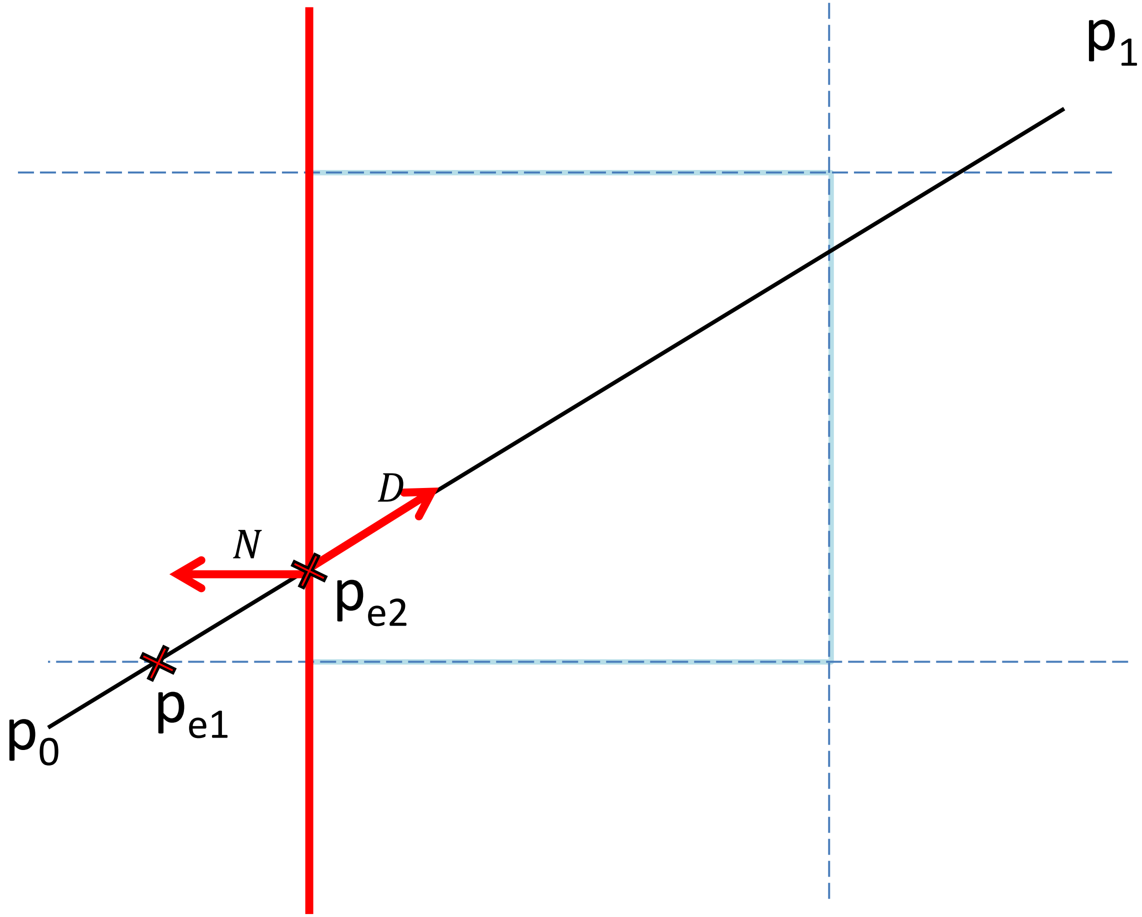
$\boldsymbol{P_e}$		$\boldsymbol{P_l}$	
t_{e1}			



True Clipping Intersection (3/12)

- $N.D < 0 \rightarrow P_e$
- $N.D > 0 \rightarrow P_l$

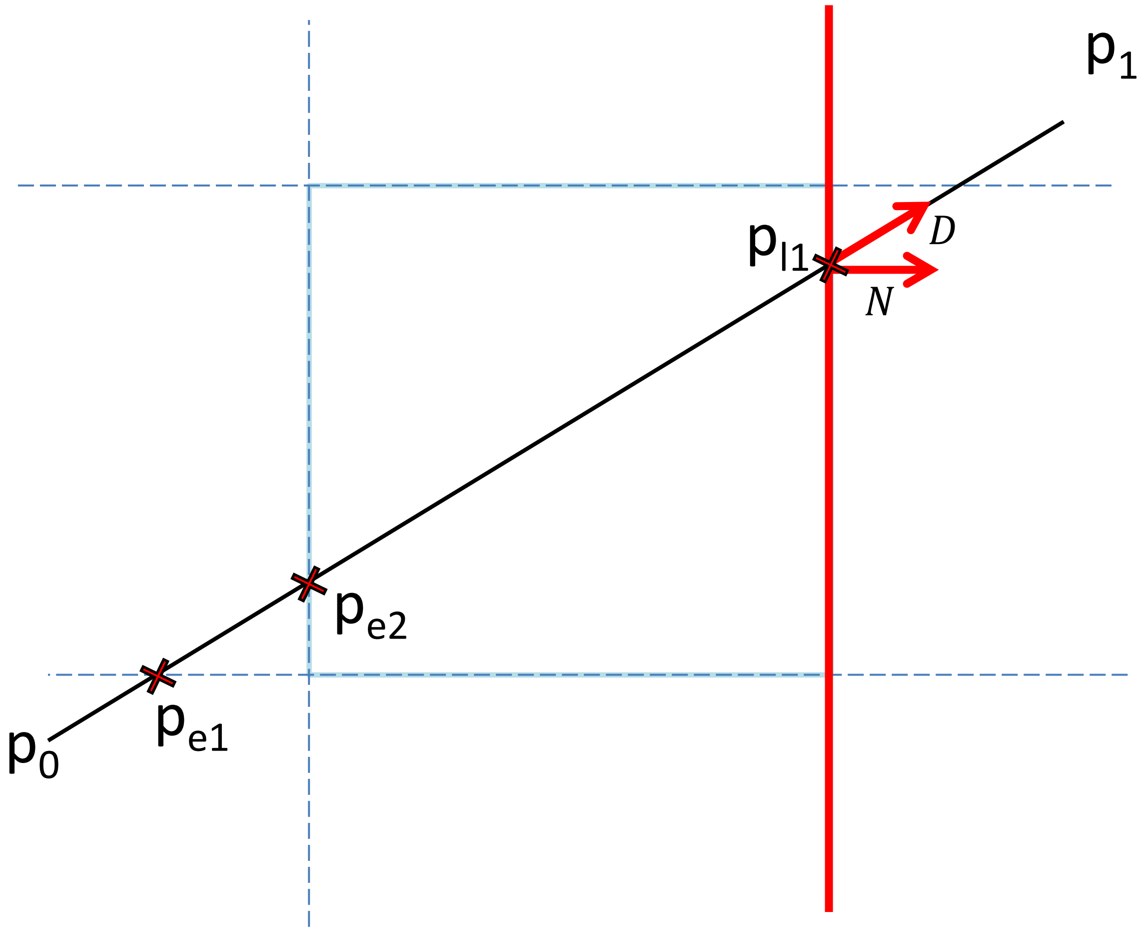
$\boldsymbol{P_e}$		$\boldsymbol{P_l}$	
t_{e1}	t_{e2}		



True Clipping Intersection (4/12)

- $N.D < 0 \rightarrow P_e$
- $N.D > 0 \rightarrow P_l$

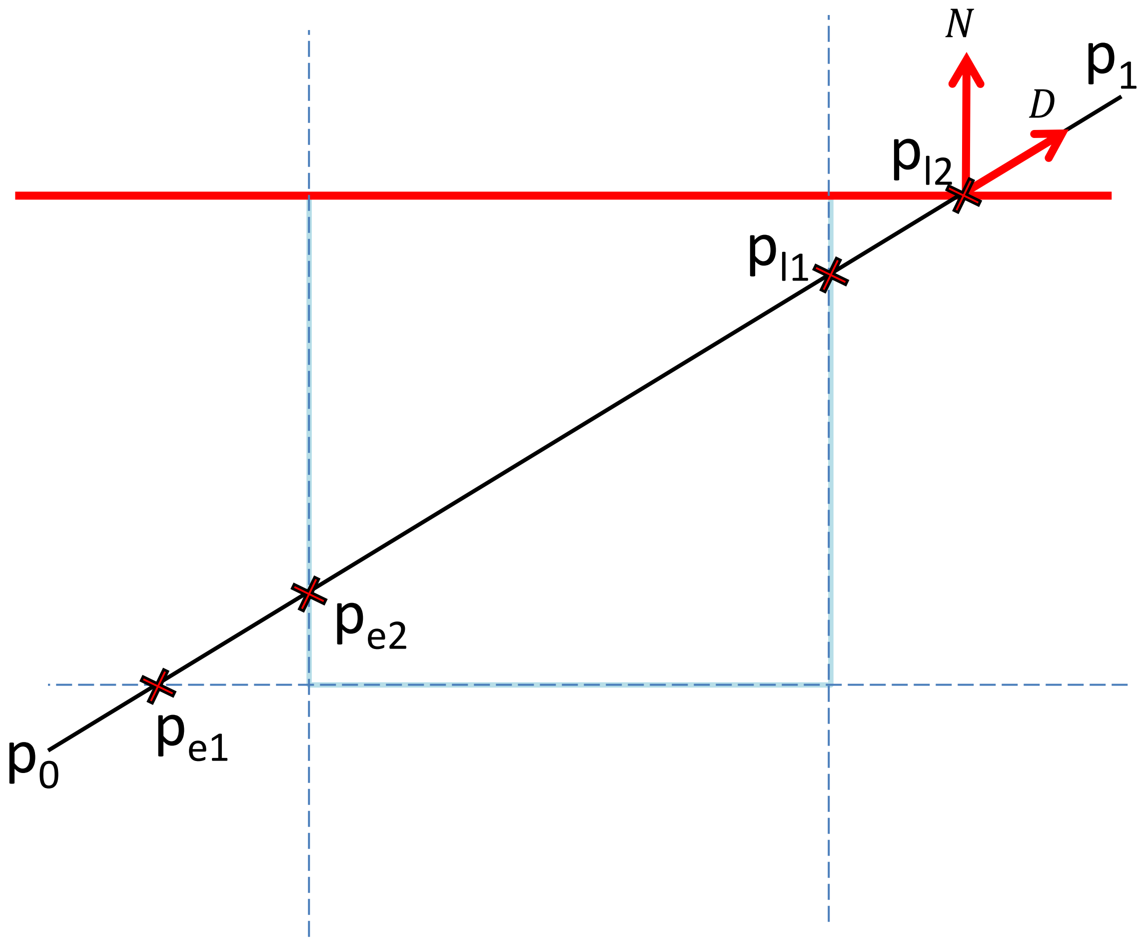
$\boldsymbol{P_e}$		$\boldsymbol{P_l}$	
t_{e1}	t_{e2}	t_{l1}	



True Clipping Intersection (5/12)

- $N.D < 0 \rightarrow P_e$
- $N.D > 0 \rightarrow P_l$

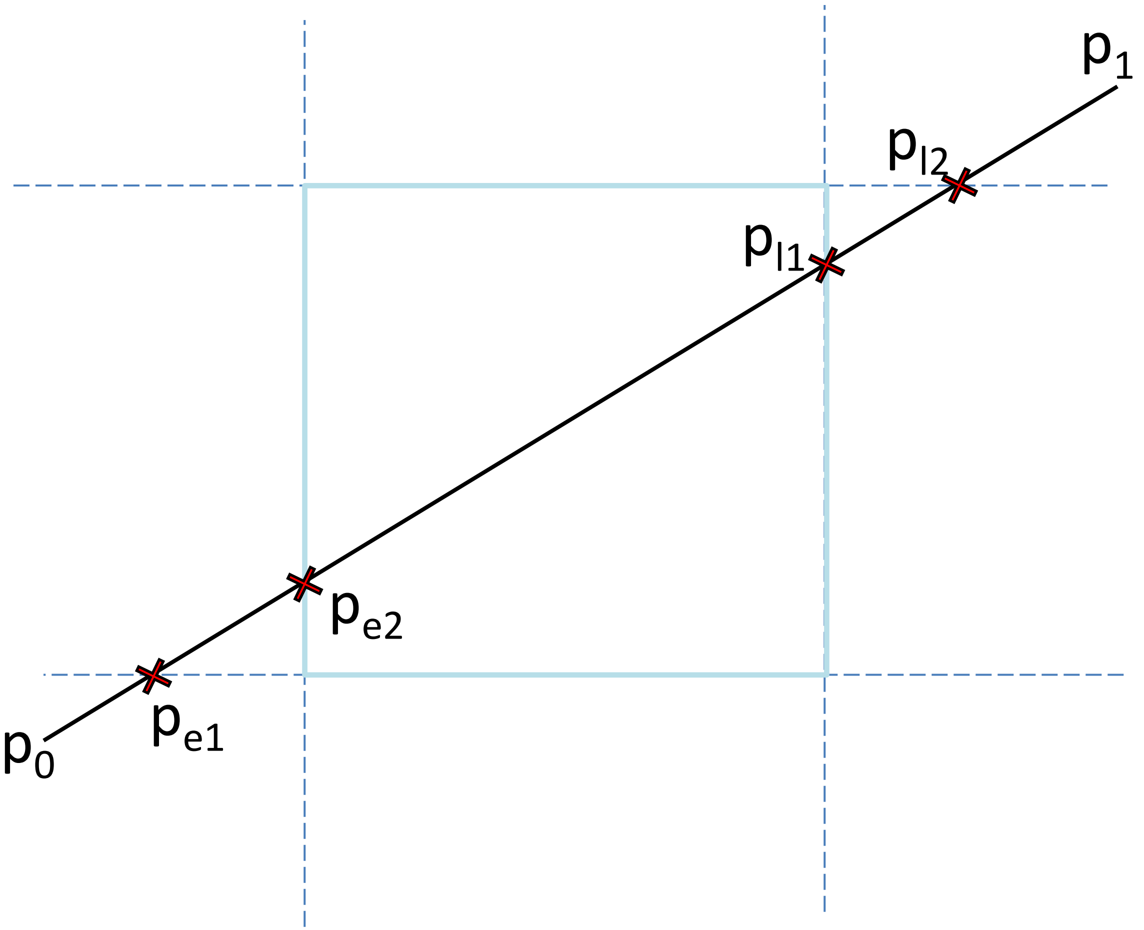
$\boldsymbol{P_e}$		$\boldsymbol{P_l}$	
t_{e1}	t_{e2}	t_{l1}	t_{l2}



True Clipping Intersection (6/12)

P_e		P_l	
t_{e1}	t_{e2}	t_{l1}	t_{l2}

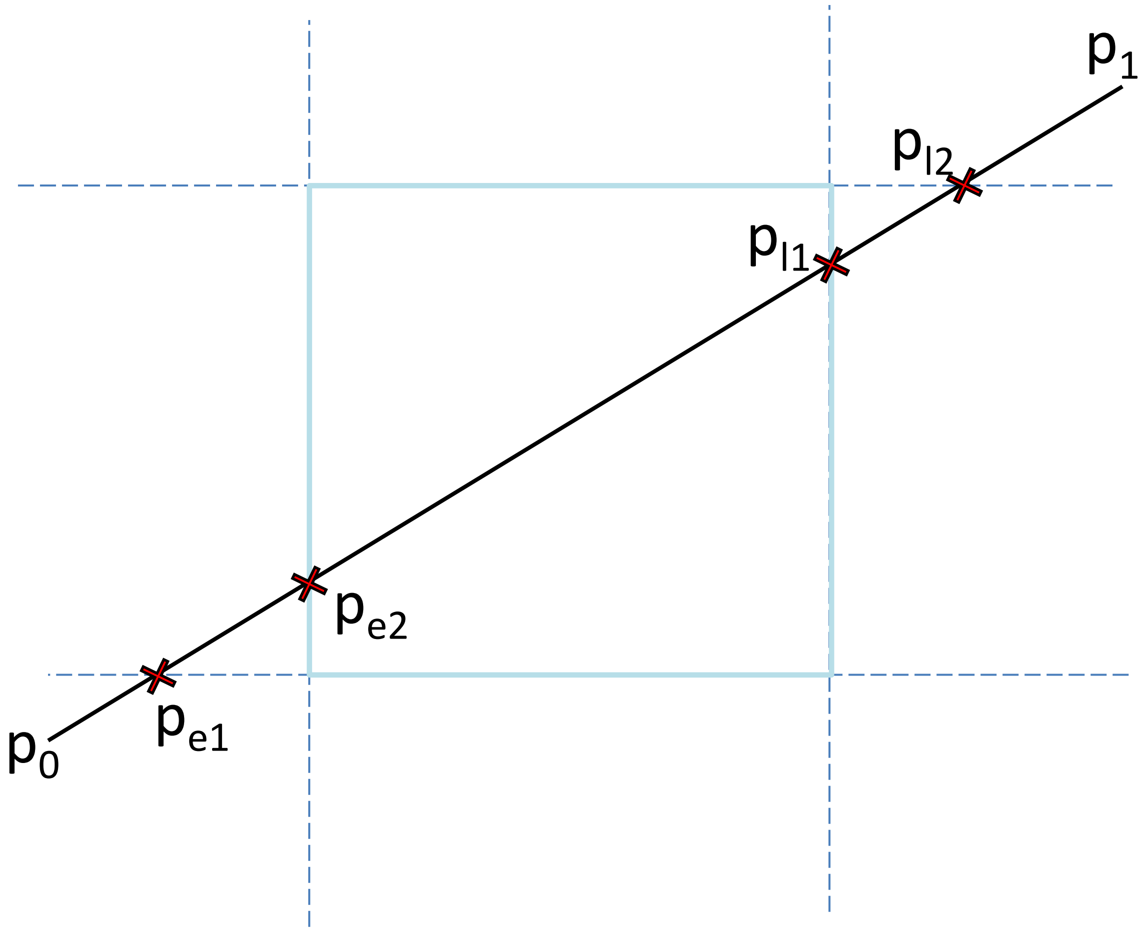
Are they in order?
Ascending or descending?



True Clipping Intersection (7/12)

P_e		P_l	
t_{e1}	t_{e2}	t_{l1}	t_{l2}

$0 < t_{e1} < t_{e2} < t_{l1} < t_{l2} < 1$



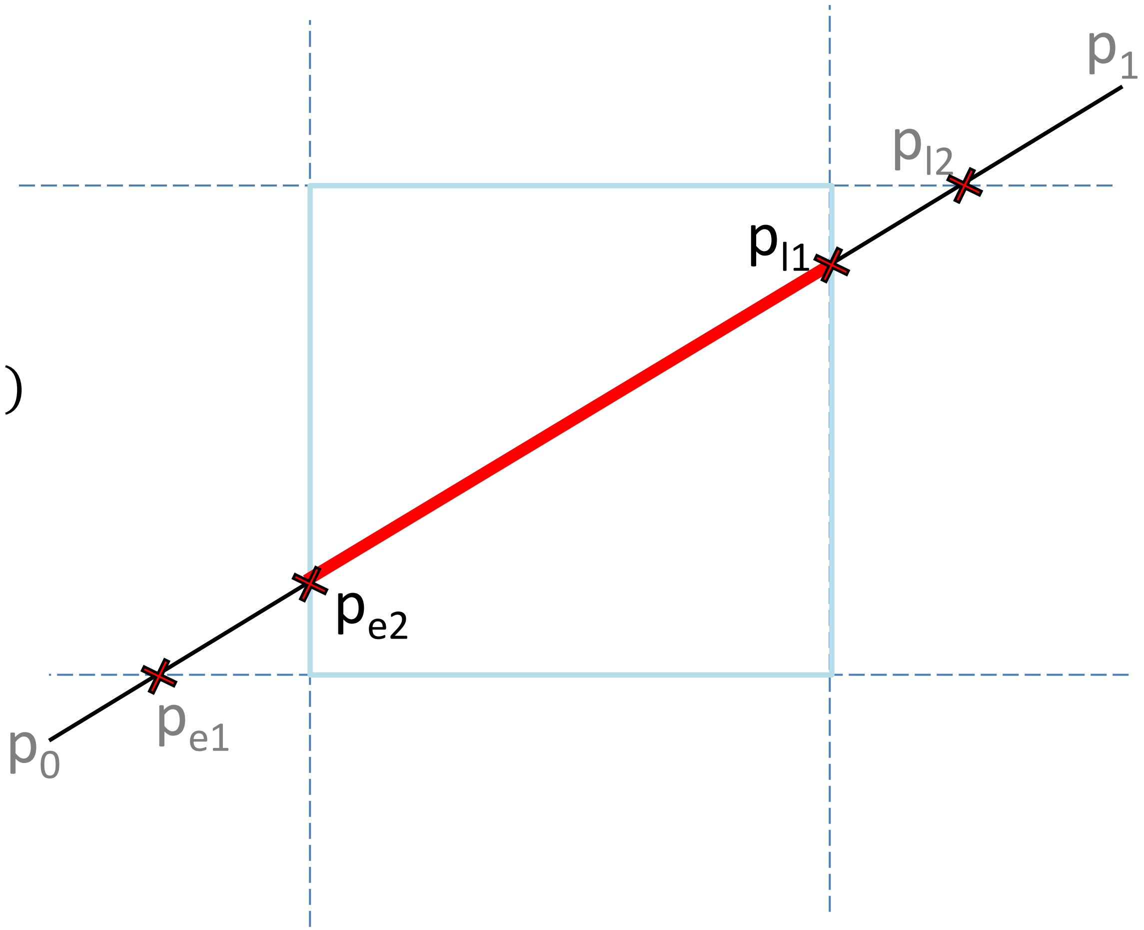
True Clipping Intersection (8/12)

- $t_E = \max\ (\boldsymbol{P_e})$
- $t_L = \min\ (\boldsymbol{P_l})$

$\boldsymbol{t_E < t_L}$ **:**

- *clip from* $p(t_E)$ *to* $p(t_L)$

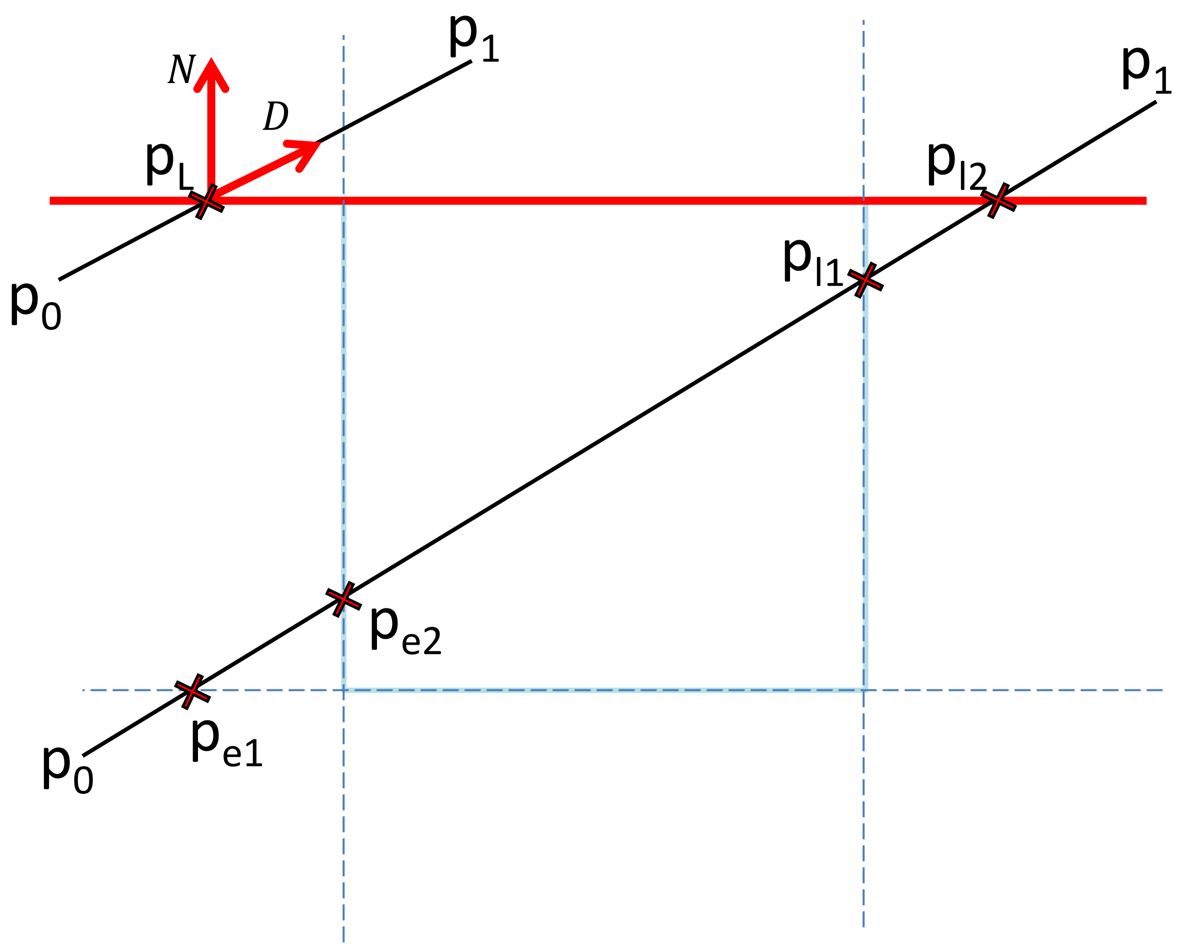
$\boldsymbol{P_e}$		$\boldsymbol{P_l}$	
t_{e1}	$\boldsymbol{t_{e2}}$	$\boldsymbol{t_{l1}}$	t_{l2}



True Clipping Intersection (9/12)

- $N.D < 0 \rightarrow P_e$
- $N.D > 0 \rightarrow P_l$

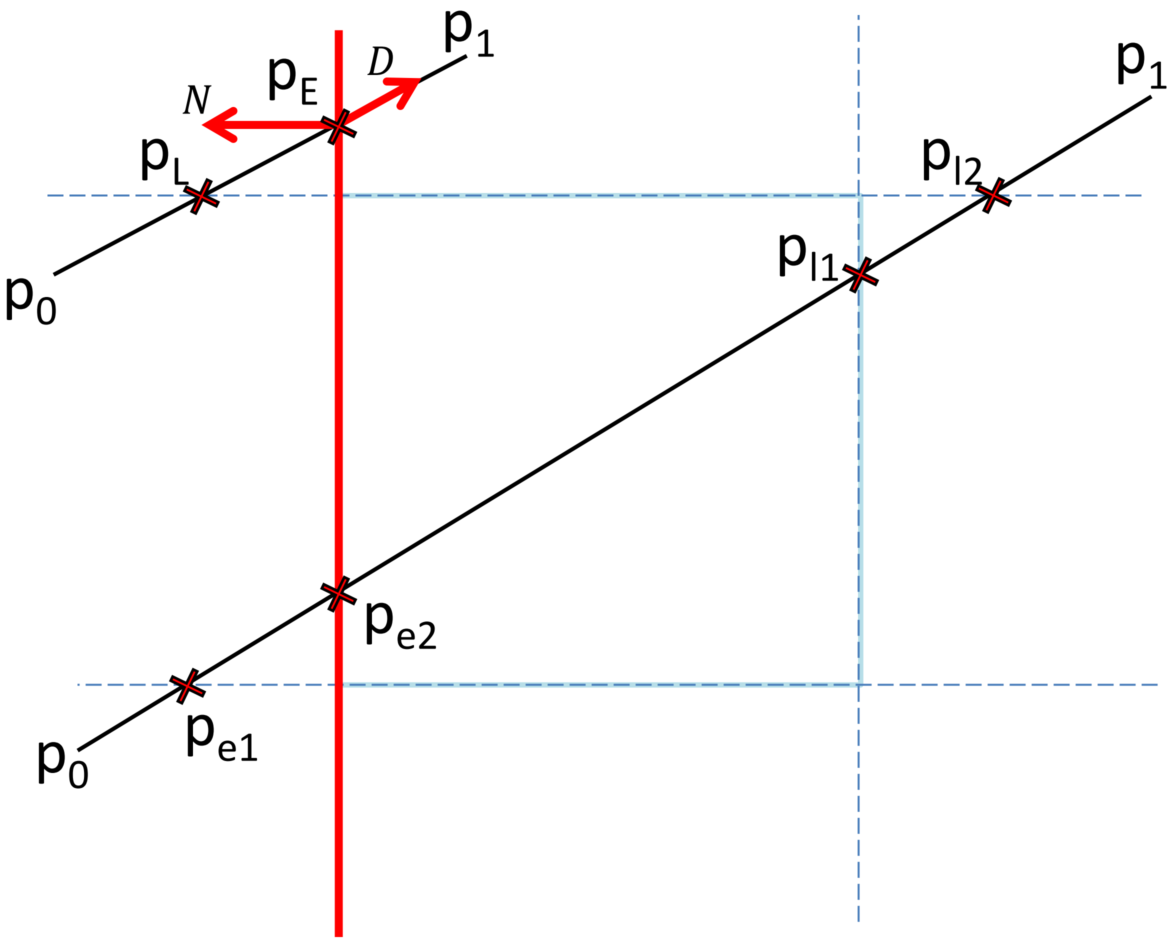
$\boldsymbol{P_e}$	$\boldsymbol{P_l}$
	t_l



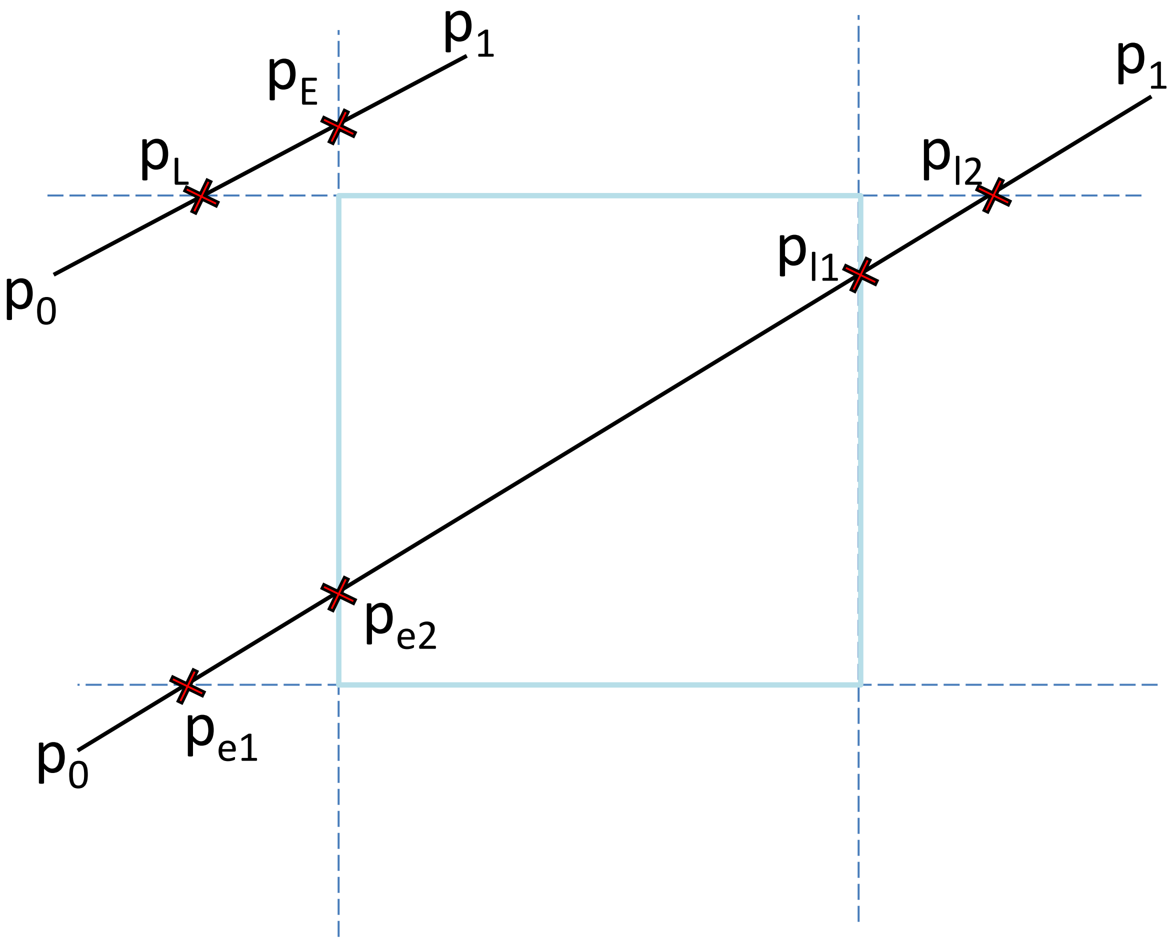
True Clipping Intersection (10/12)

- $N.D < 0 \rightarrow P_e$
- $N.D > 0 \rightarrow P_l$

$\boldsymbol{P_e}$	$\boldsymbol{P_l}$
t_e	t_l



True Clipping Intersection (11/12)



$\boldsymbol{P_e}$	$\boldsymbol{P_l}$
t_e	t_l

$$1 > t_e > t_l > 0$$

True Clipping Intersection (12/12)

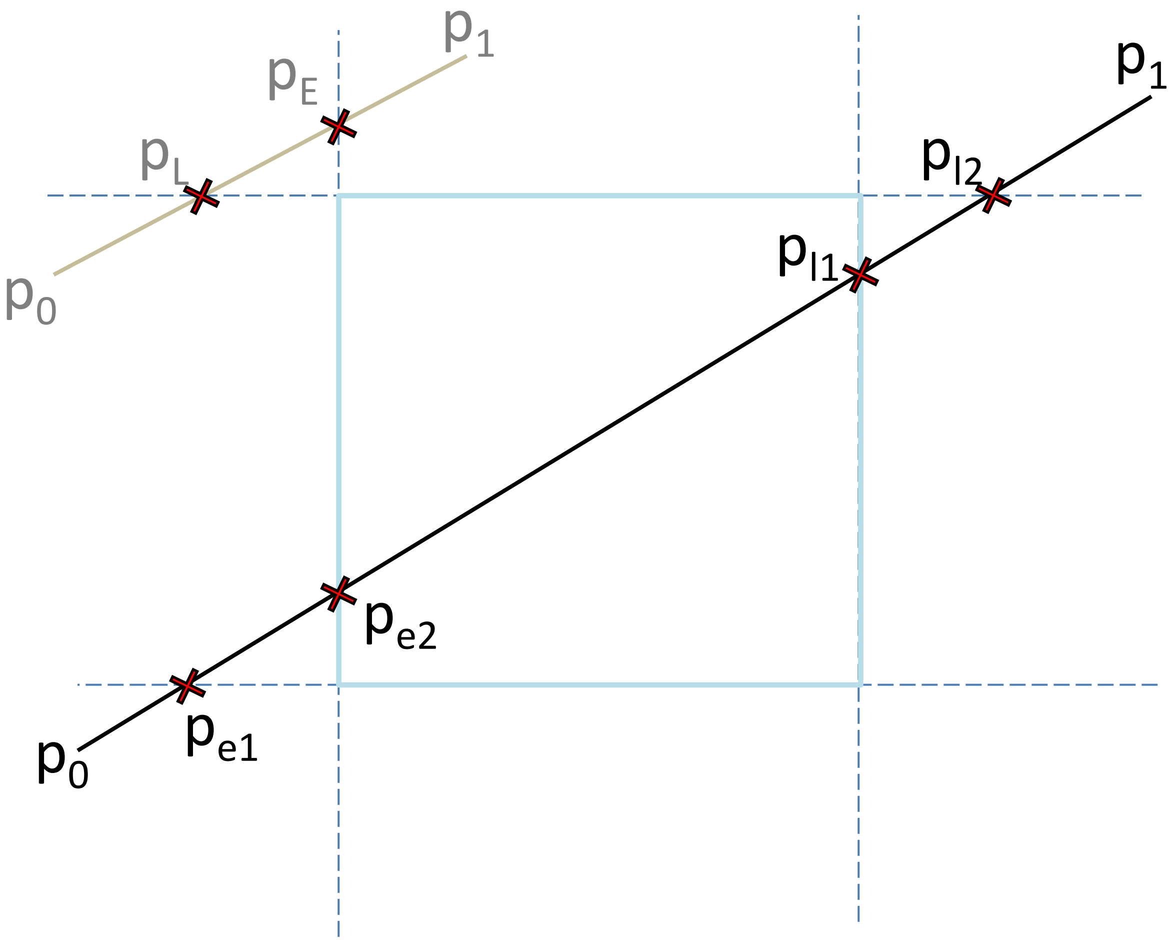
- $t_E = \max(\boldsymbol{P_e})$
- $t_L = \min(\boldsymbol{P_l})$

But this time,

$$\boldsymbol{t_E} > \boldsymbol{t_L} :$$

- *Reject the line*

$\boldsymbol{P_e}$	$\boldsymbol{P_l}$
t_e	t_l

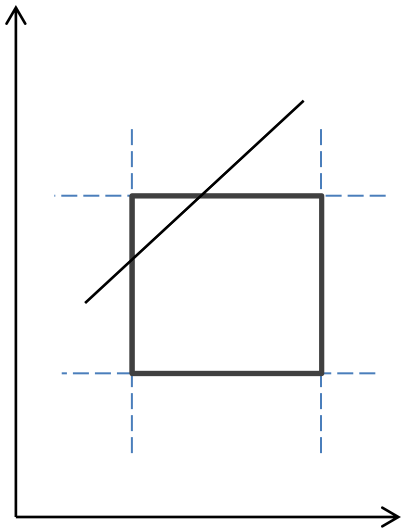


Cyrus-Beck Algorithm (1/1)

```
precalculate Ni and select a PEi for each edge;
for each line segment to be clipped
    if P1 = P0 then
            line is degenerate so clip as a point;
    else
            begin
                tE = 0; tL = 1;
                for each clip edge
                    if Ni •D ≠ 0 then {Ignore edges parallel to line}
                        begin
                            calculate t; {of line ∩ clip edge}
                            use sign of Ni • D to categorize as PE or PL;
                            if PE then tE = max (tE, t);
                            if PL then tL = min (tL, t)
                        end
                if tE > tL then
                    return nil
                else
                    return P (tE) and P (tL) as true clip intersections
            end {else}
```

Known Cases (1/1)

- $D = P_1 - P_0 = (x_1 - x_0, y_1 - y_0)$
- P_{Ei} as an arbitrary point on the clip edge; it's a free variable and drops out



Calculations for Parametric Line Clipping Algorithm

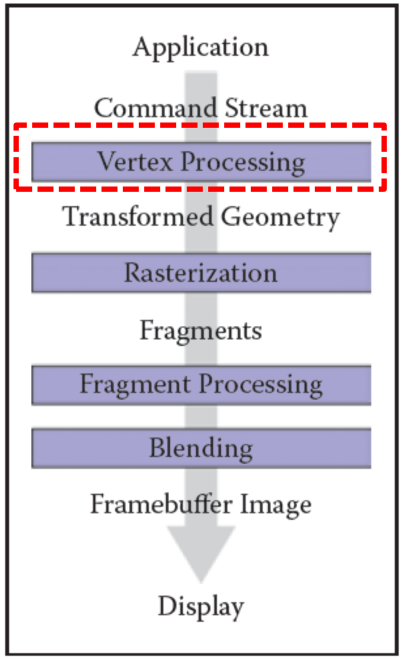
Clip Edge$_i$	Normal N_i	P_{E_i}	$P_o - P_{E_i}$	$t = \frac{N_i \cdot (P_0 - P_{E_i})}{-N_i \cdot D}$
left: $x = x_{min}$	$(-1, 0)$	(x_{min}, y)	$(x_0 - x_{min}, y_0 - y)$	$\frac{-(x_o - x_{min})}{(x_1 - x_o)}$
right: $x = x_{max}$	$(1, 0)$	(x_{max}, y)	$(x_0 - x_{max}, y_0 - y)$	$\frac{(x_0 - x_{max})}{-(x_1 - x_0)}$
bottom: $y = y_{min}$	$(0, -1)$	(x, y_{min})	$(x_0 - x, y_0 - y_{min})$	$\frac{-(y_0 - y_{min})}{(y_1 - y_0)}$
top: $y = y_{max}$	$(0, 1)$	(x, y_{max})	$(x_0 - x, y_0 - y_{max})$	$\frac{(y_0 - y_{max})}{-(y_1 - y_0)}$

Operations Before and After Rasterization

Before Rasterization (1/1)

Before a primitive can be rasterized:

- *The vertices* must be in screen:
  - **M**odeling
  - **V**iewing
  - **P**rojection transformations
  - Original coordinates → screen space
- *Attributes* that are supposed to be interpolated must be known.
  - colors, surface normals, or texture coordinates, is transformed as needed.
- Done *in Vertex Processing stage*

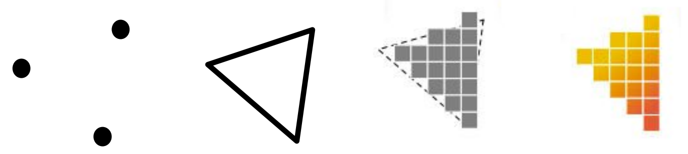


Credit: Fundamentals of Computer Graphics 3rd Edition by Peter Shirley, Steve Marschner | http://www.cs.cornell.edu/courses/cs4620/2019fa/

After Rasterization (1/1)

After a primitive can be rasterized:

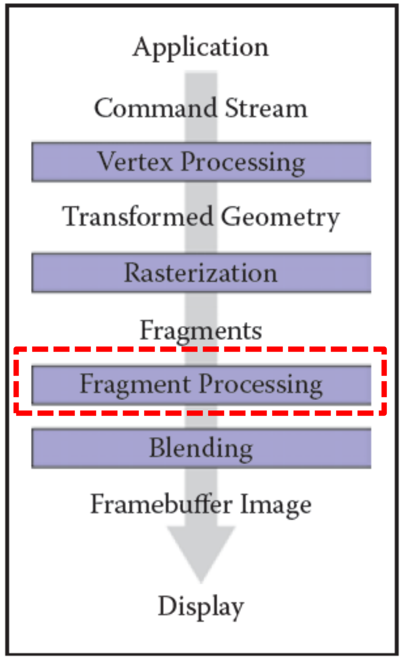
– Computing *a color and depth* for each fragment (i.e. Shading).



– Performing *blending phase*.

- combines the fragments that overlapped.
- compute the final color.

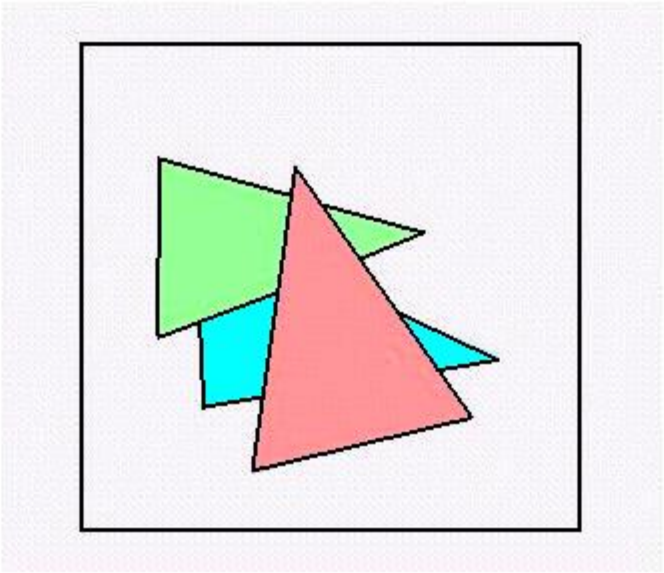
– Done in *Fragment Processing stage*



Credit: Fundamentals of Computer Graphics 3rd Edition by Peter Shirley, Steve Marschner | http://www.cs.cornell.edu/courses/cs4620/2019fa/

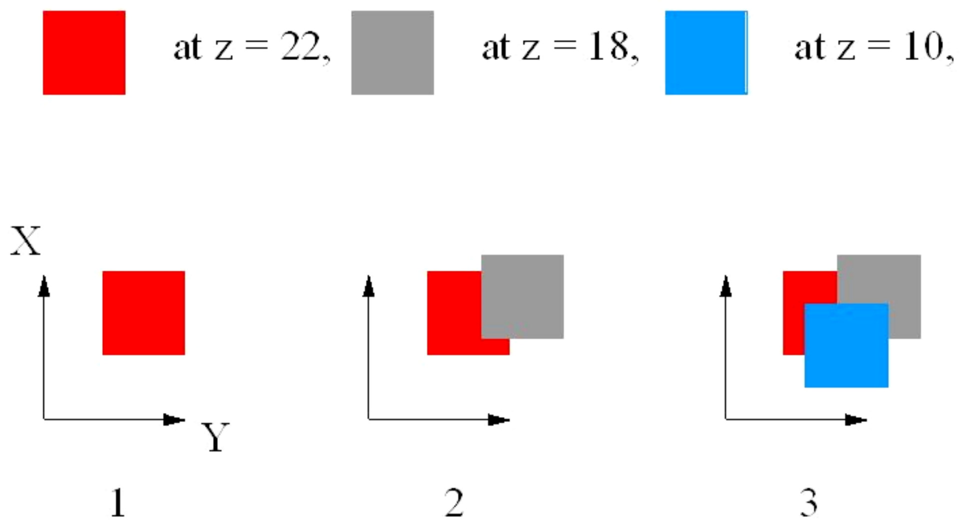
A Minimal 3D Pipeline (2/16)

- Main challenge is – ***occlusion***.



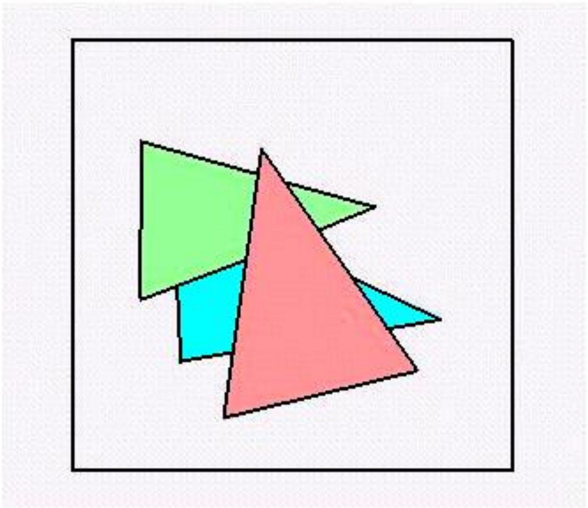
A Minimal 3D Pipeline (3/16)

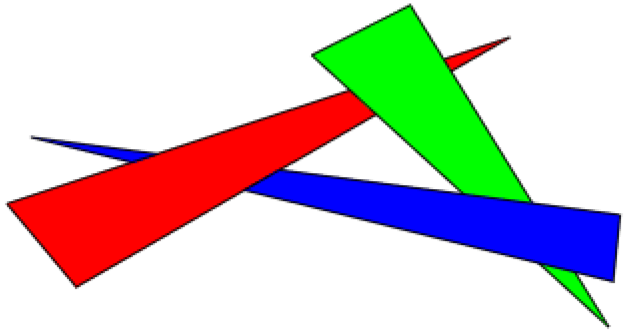
- ***Painter's Algorithm***
  - Sort surfaces/ polygons by their depth (z values)
  - Draw objects in order (farthest to closest)



A Minimal 3D Pipeline (4/16)

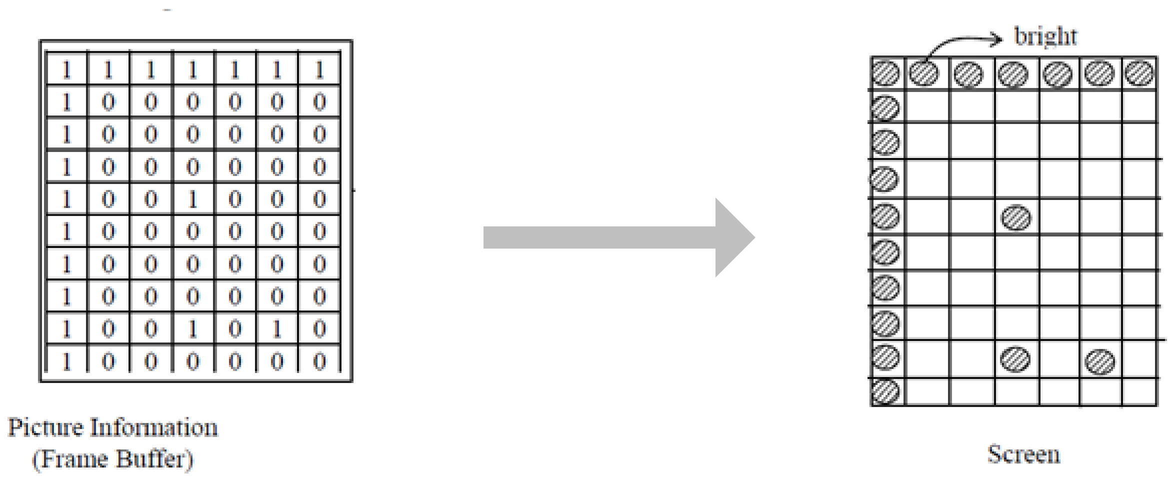
- ***Painter's Algorithm***
  - Disadvantage:
    - Sometimes it is difficult to sort





A Minimal 3D Pipeline (6/16)

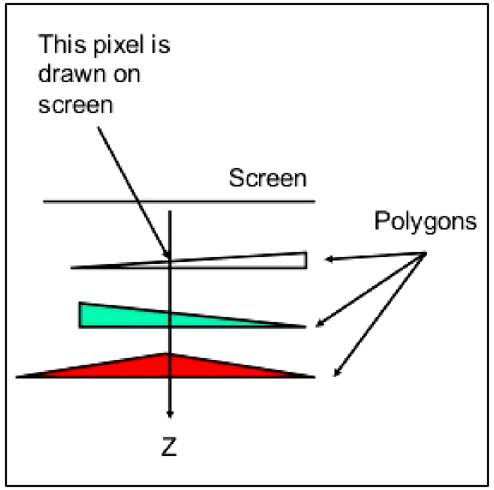
- A **frame buffer** is a portion of memory (RAM) containing a bitmap that drives a video display.
  - It is a memory buffer containing a complete frame of data

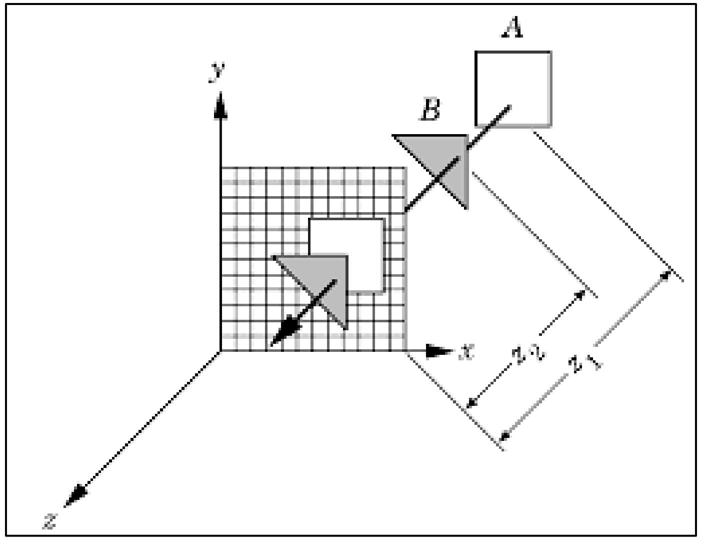


A Minimal 3D Pipeline (7/16)

Z-buffer Algorithm:

- *At each pixel* we keep track of *the distance to the closest surface* that has been drawn so far
  - we *throw* away fragments that are farther away than that distance.





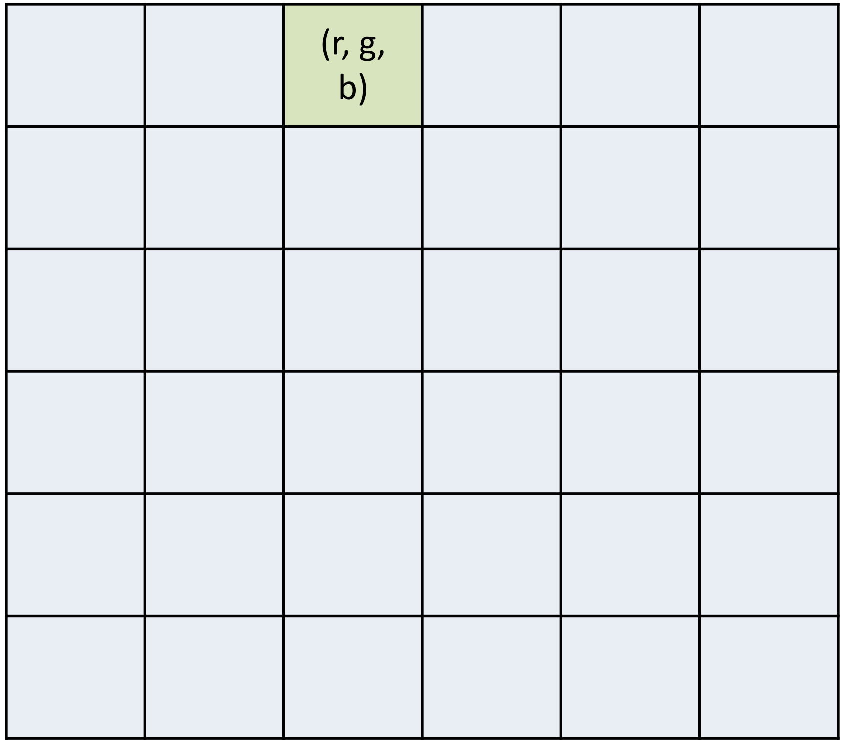
A Minimal 3D Pipeline (8/16)

Z-buffer Algorithm:

- Implementation:
  - Red, green, and blue color values *(**frame buffer**)* + depth, or z-value (***z-buffer***).
    - {(r, g ,b) , z}

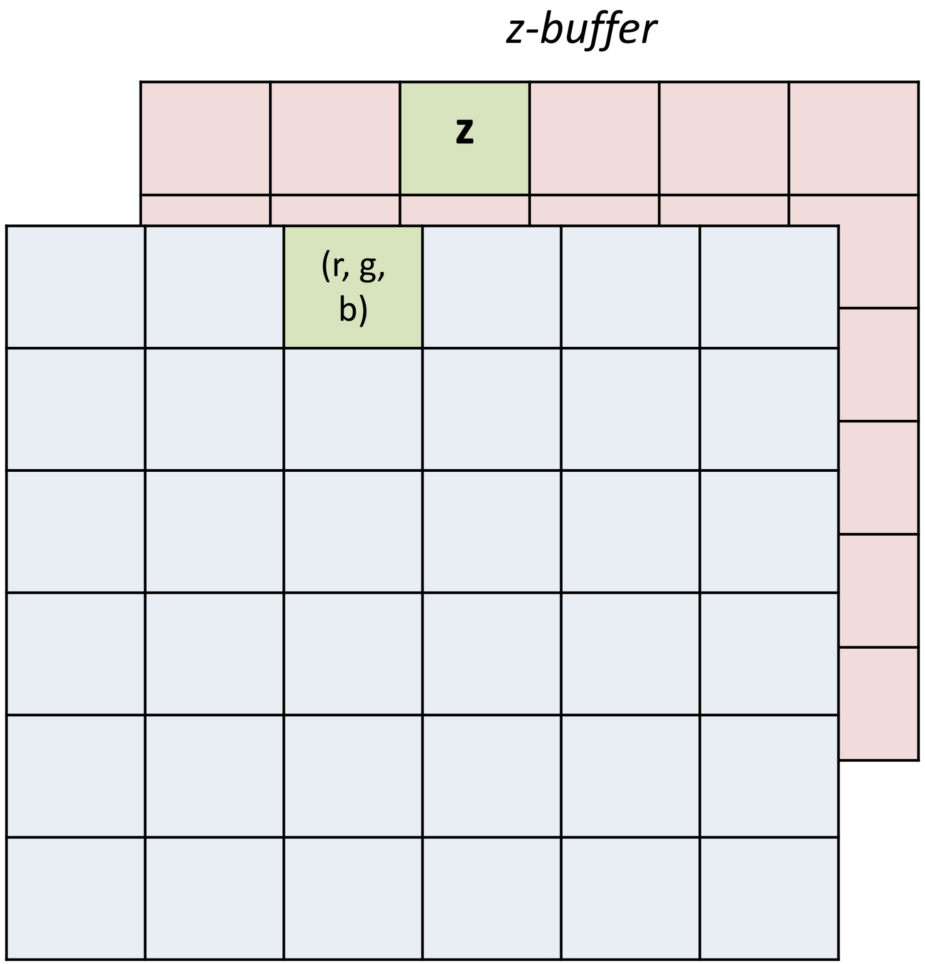
A Minimal 3D Pipeline (9/16)

Z-buffer Algorithm:



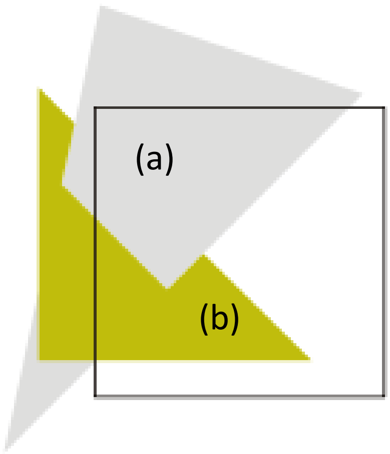
A Minimal 3D Pipeline (10/16)

Z-buffer Algorithm:



A Minimal 3D Pipeline (11/16)

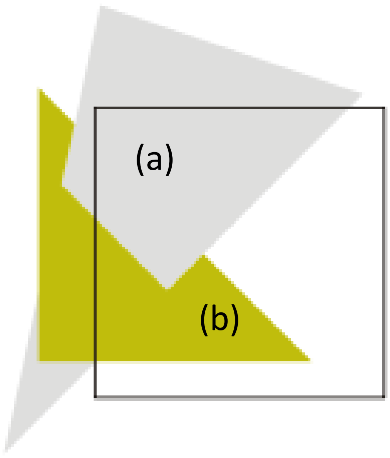
Z-buffer Algorithm:

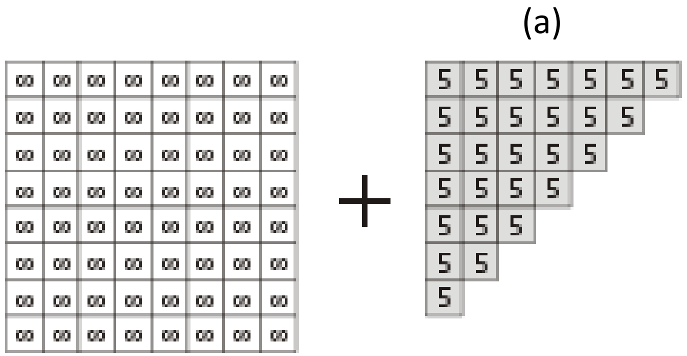


∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞

A Minimal 3D Pipeline (12/16)

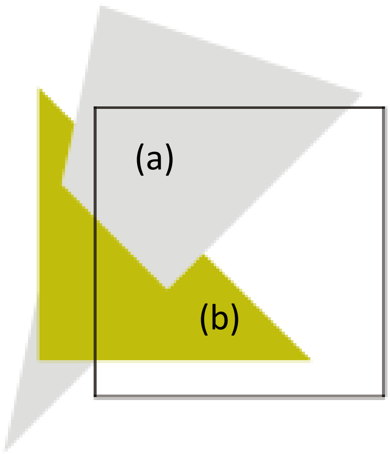
Z-buffer Algorithm:

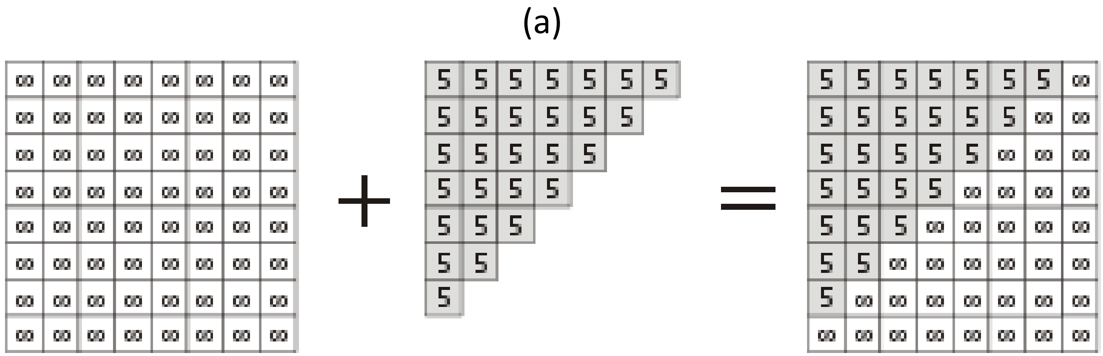




A Minimal 3D Pipeline (13/16)

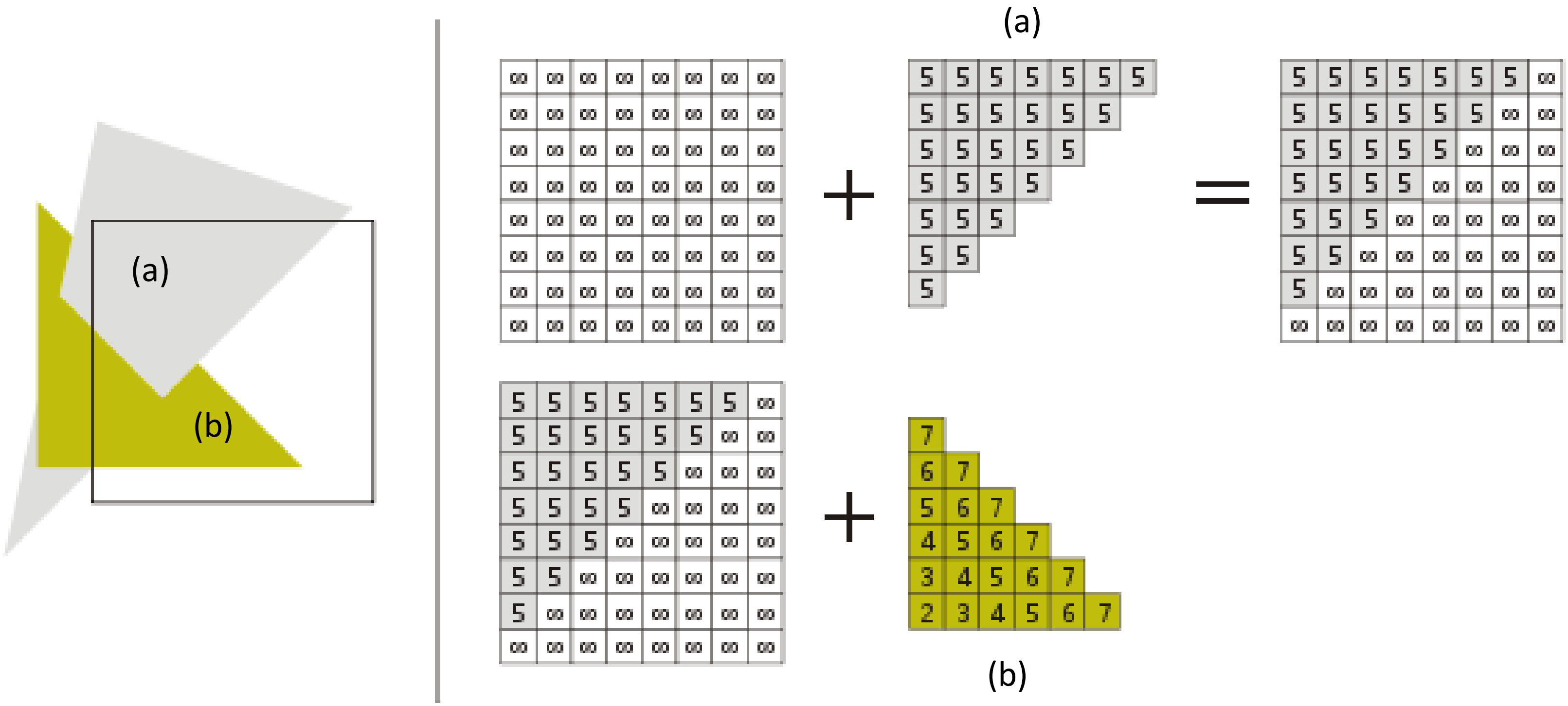
Z-buffer Algorithm:





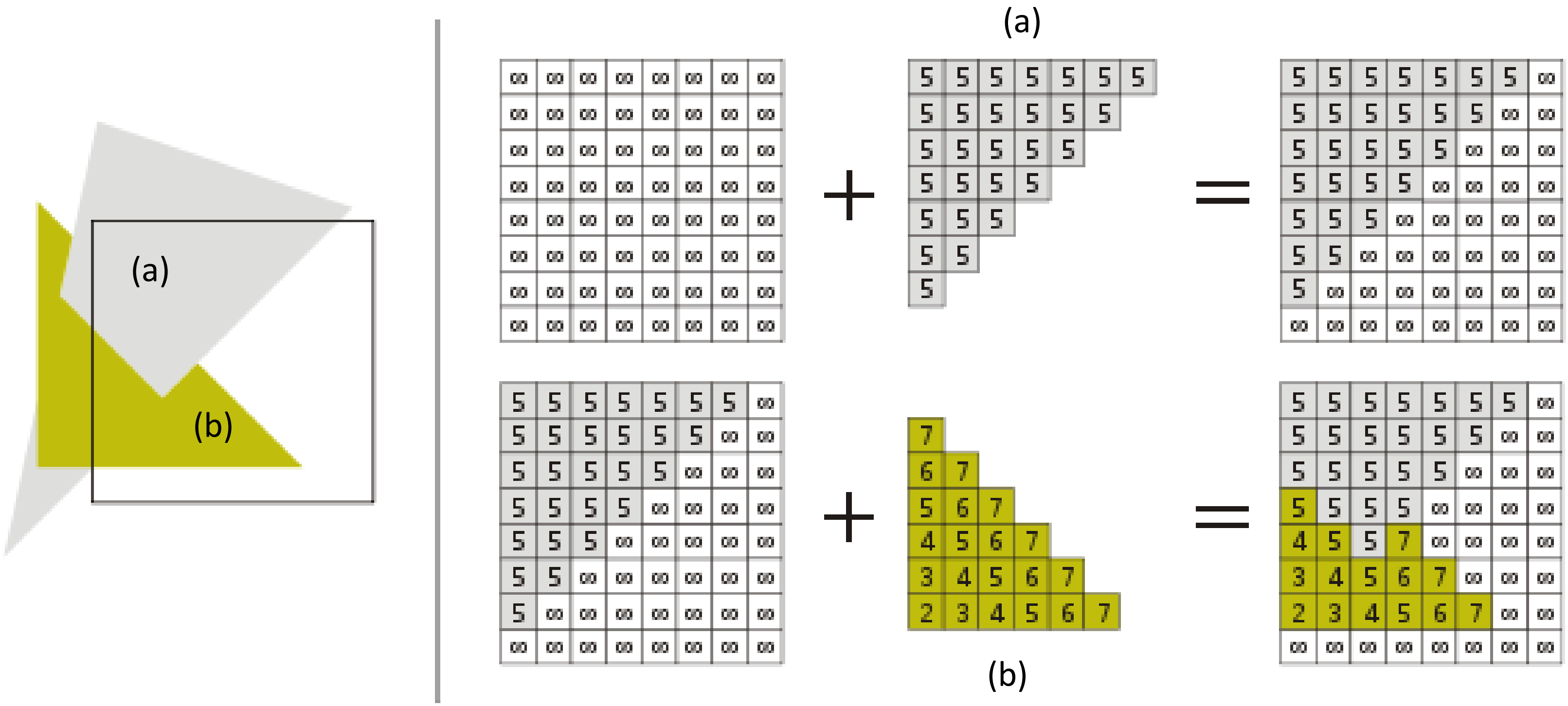
A Minimal 3D Pipeline (14/16)

Z-buffer Algorithm:



A Minimal 3D Pipeline (15/16)

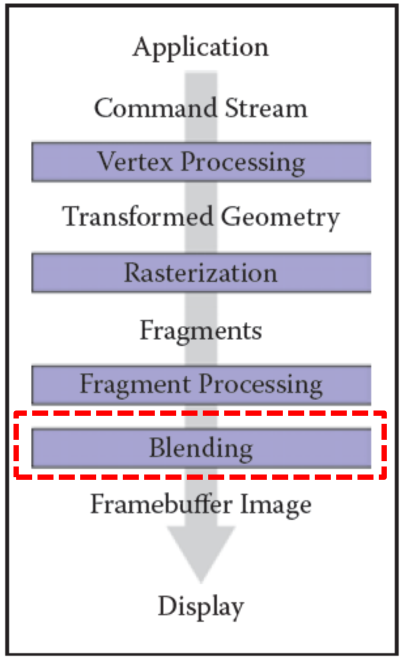
Z-buffer Algorithm:



A Minimal 3D Pipeline (16/16)

Z-buffer Algorithm:

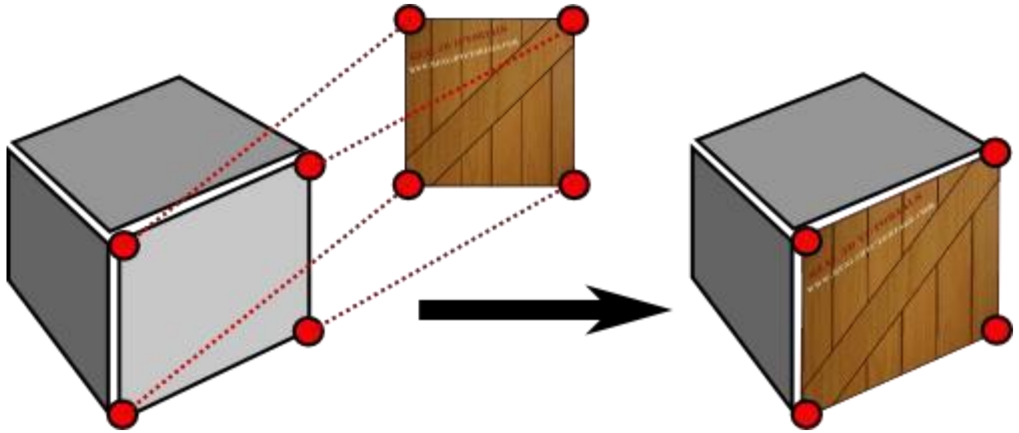
- Done in the *fragment blending phase*.



Credit: Fundamentals of Computer Graphics 3rd Edition by Peter Shirley, Steve Marschner | http://www.cs.cornell.edu/courses/cs4620/2019fa/

Texture Mapping (1/8)

- During shading, we read one of the color values *from a texture.*
  - *instead of using the attribute* values (colors) that are attached to the geometry.



Credit: Fundamentals of Computer Graphics 3rd Edition by Peter Shirley, Steve Marschner | http://www.cs.cornell.edu/courses/cs4620/2019fa/

Texture Mapping (2/8)

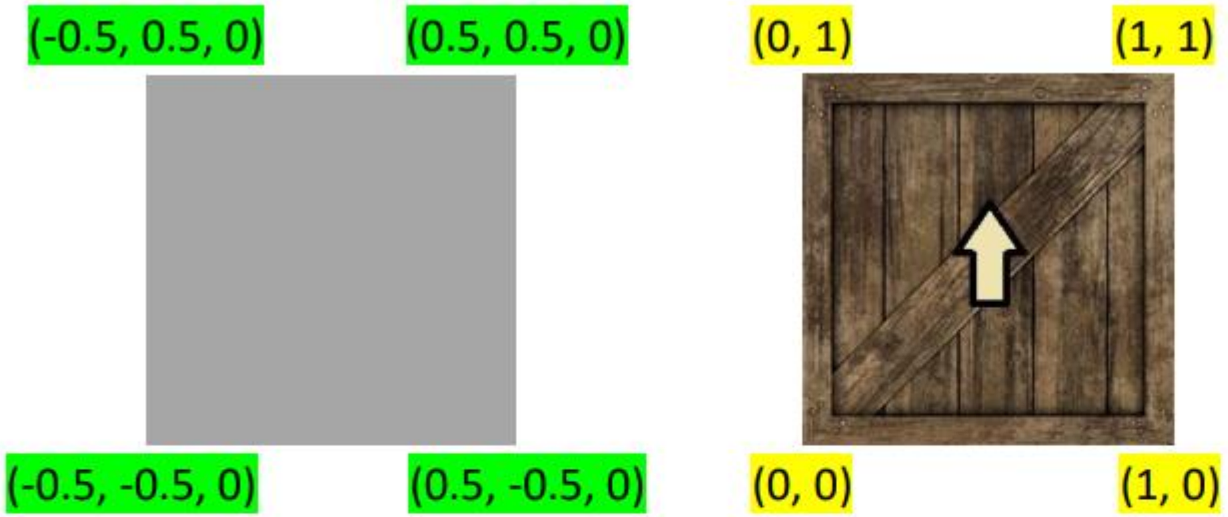
Texture lookup:

- specifies a *texture coordinate*
  - a point in the domain of the texture, and the texture-mapping.

Credit: Fundamentals of Computer Graphics 3rd Edition by Peter Shirley, Steve Marschner | http://www.cs.cornell.edu/courses/cs4620/2019fa/

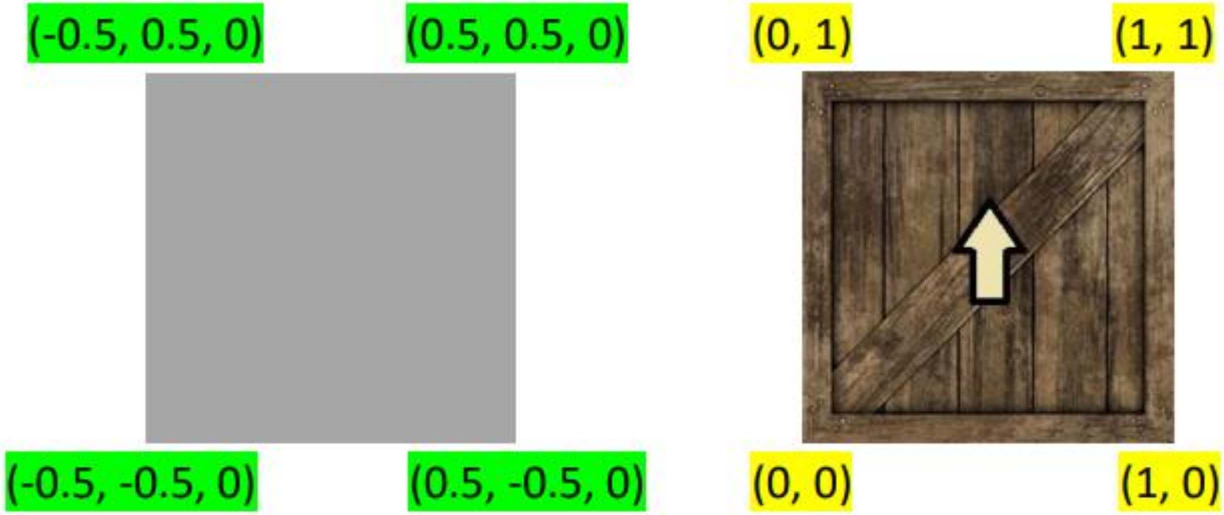
Texture Mapping (3/8)

- XY coordinate ↔ UV coordinate
  - Example: Quad



Texture Mapping (4/8)

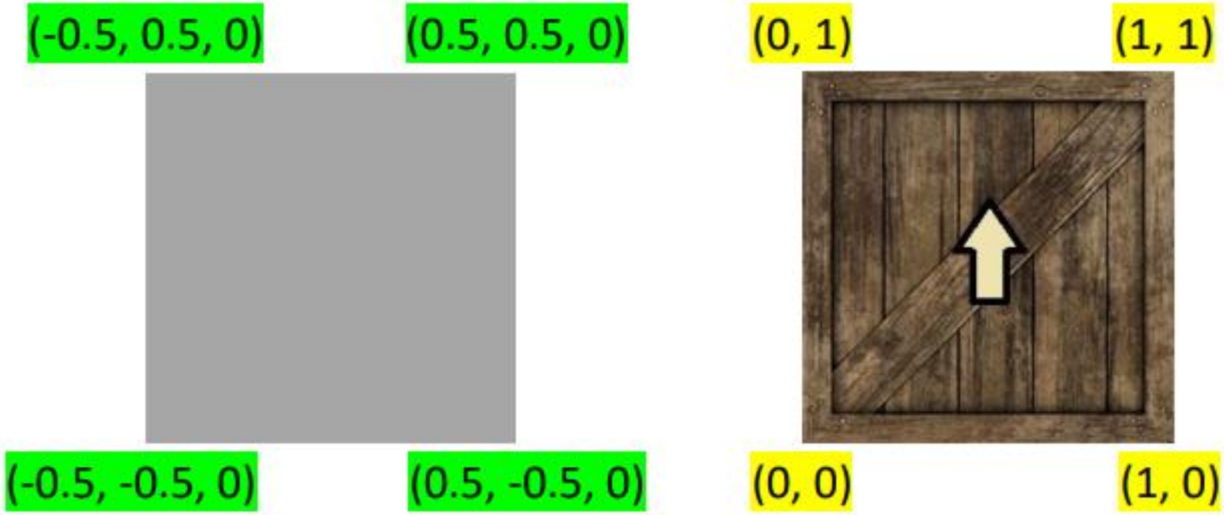
- XY coordinate ↔ UV coordinate



XY	UV
(-0.5, -0.5, 0)	(0, 0)
(0.5, -0.5, 0)	(1, 0)
(0.5, 0.5, 0)	(1, 1)
(-0.5, 0.5, 0)	(0, 1)

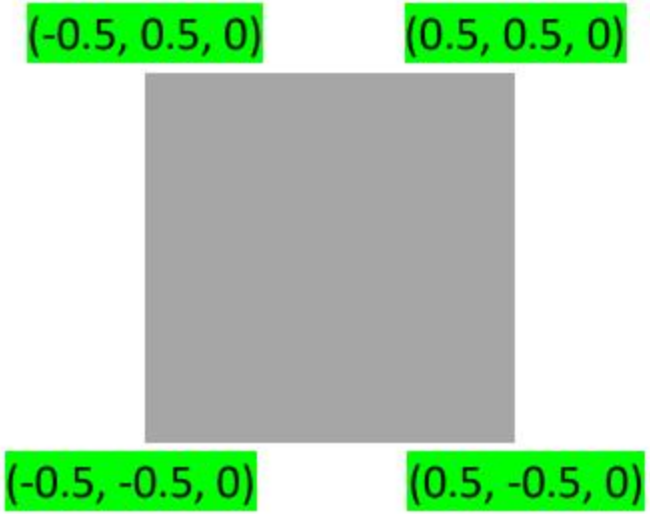
Texture Mapping (5/8)

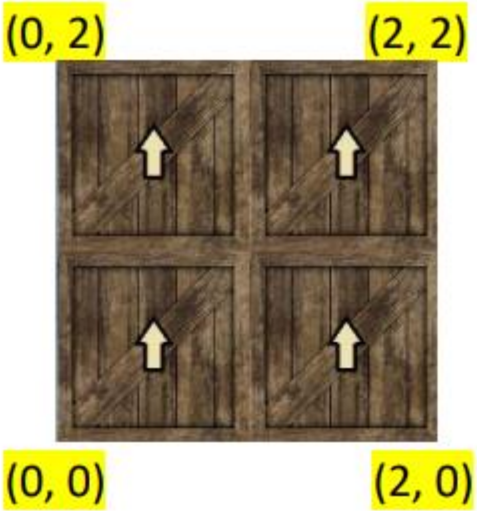
- XY coordinate ↔ UV coordinate



XY	UV
(-0.5, -0.5, 0)	(0, 0)
(0.5, -0.5, 0)	(1, 0)
(0.5, 0.5, 0)	(1, 1)
(-0.5, 0.5, 0)	(0, 1)

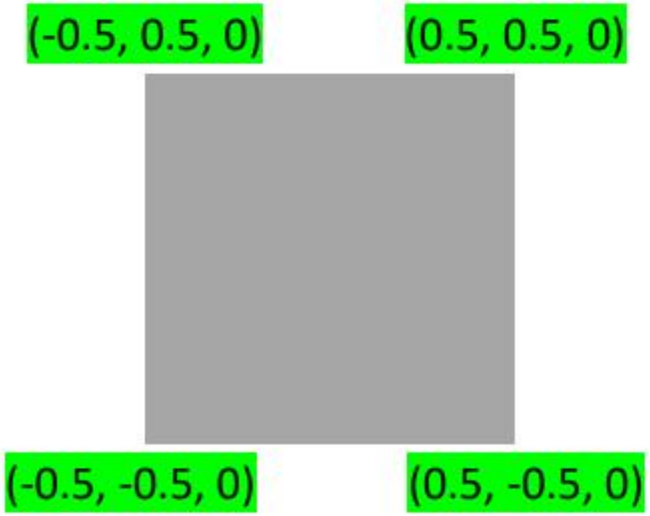
Texture Mapping (6/8)





XY	UV
(-0.5, -0.5, 0)	(0, 0)
(0.5, -0.5, 0)	(2, 0)
(0.5, 0.5, 0)	(2, 2)
(-0.5, 0.5, 0)	(0, 2)

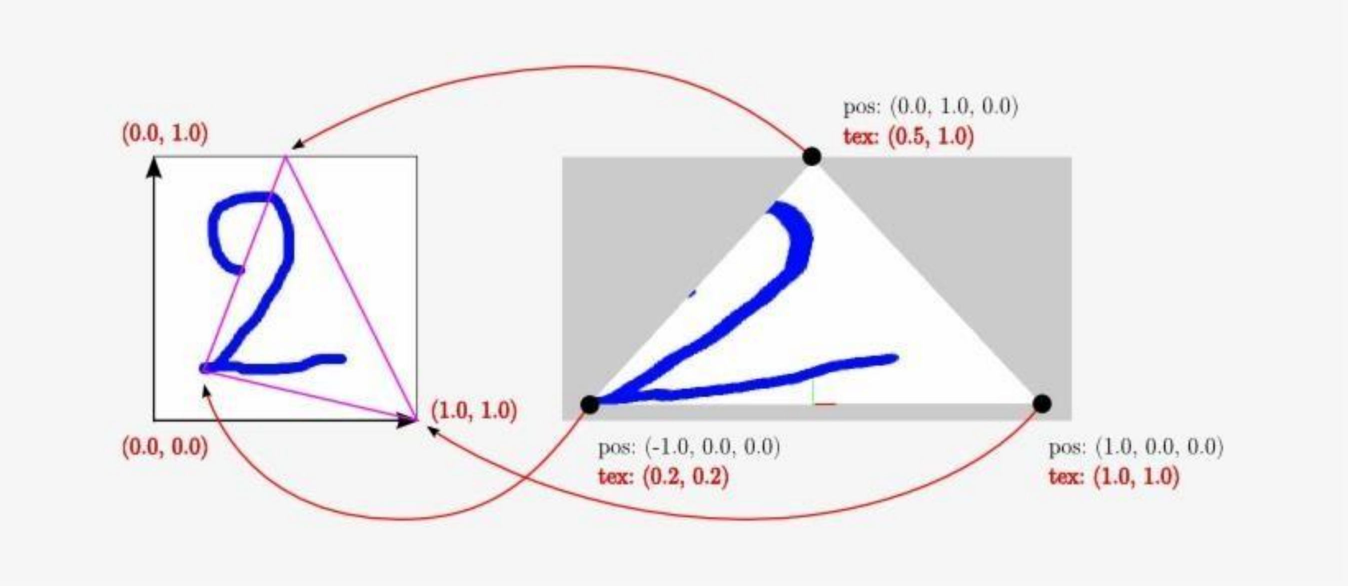
Texture Mapping (7/8)



XY	UV
(-0.5, -0.5, 0)	(0, 0)
(0.5, -0.5, 0)	(,)
(0.5, 0.5, 0)	(,)
(-0.5, 0.5, 0)	(,)

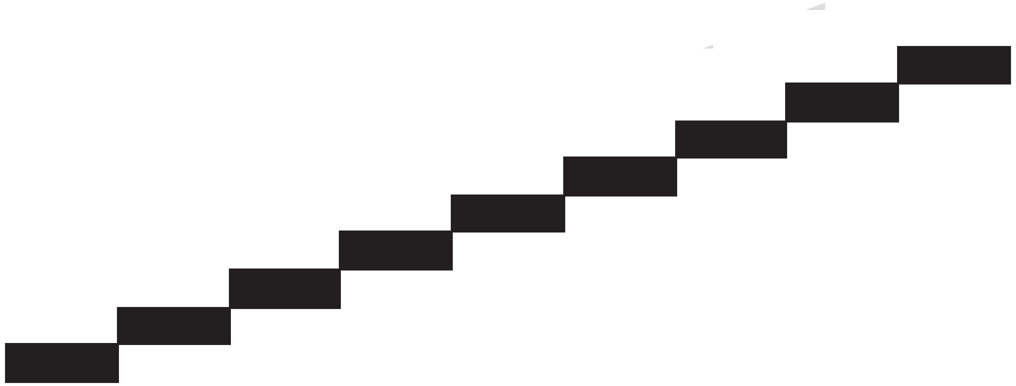
Texture Mapping (8/8)

- XY coordinate ↔ UV coordinate
  - Example: triangle



Anti-aliasing (1/6)

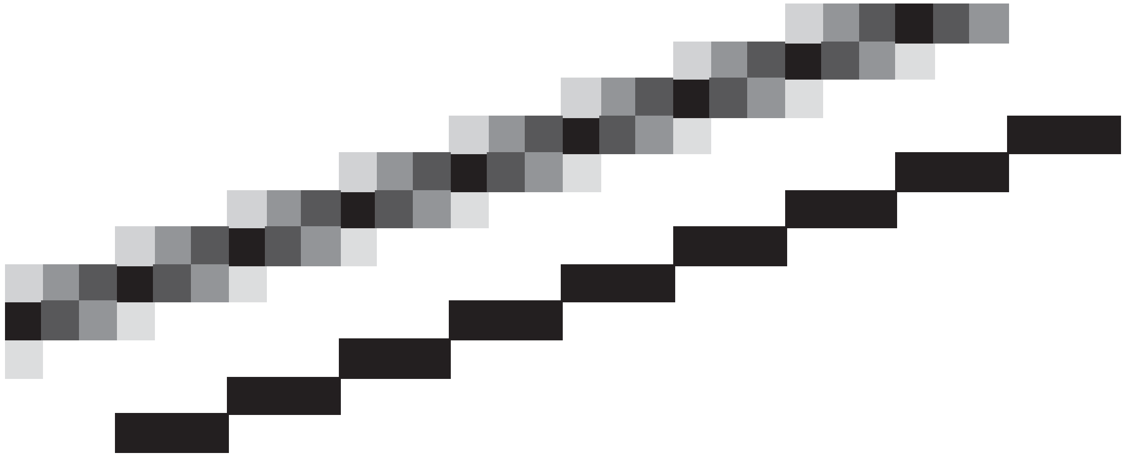
- Aliasing



Credit: Fundamentals of Computer Graphics 3rd Edition by Peter Shirley, Steve Marschner | http://www.cs.cornell.edu/courses/cs4620/2019fa/

Anti-aliasing (2/6)

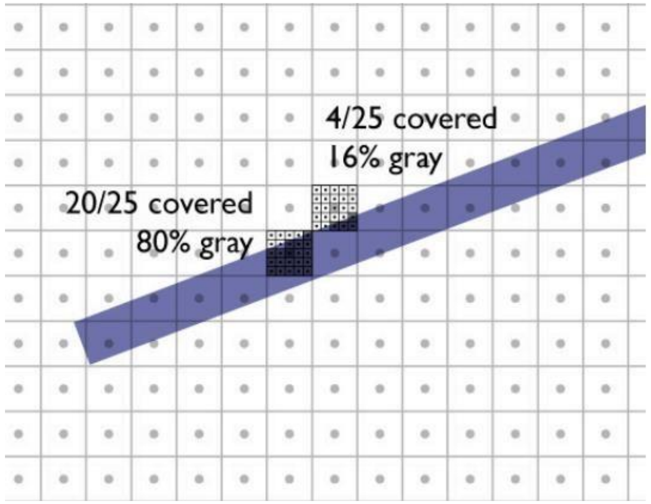
- Anti-aliasing



Credit: Fundamentals of Computer Graphics 3rd Edition by Peter Shirley, Steve Marschner | http://www.cs.cornell.edu/courses/cs4620/2019fa/

Anti-aliasing (3/6)

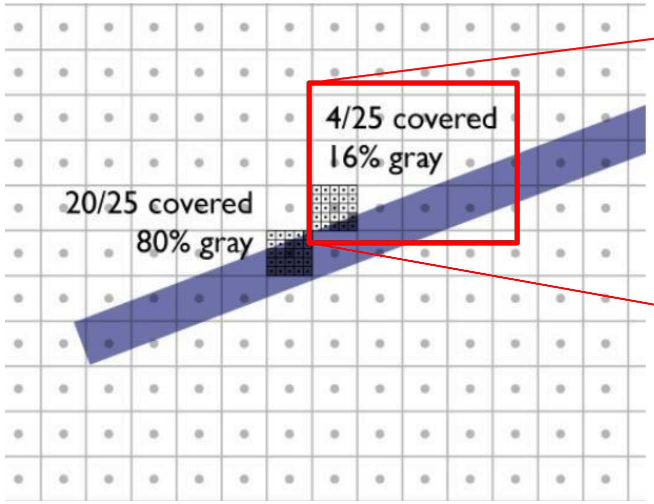
- Anti-aliasing:
  - Box filtering by supersampling

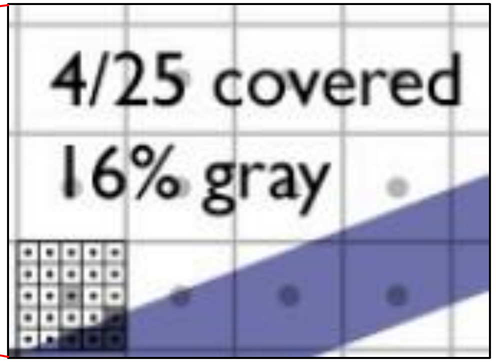


Credit: Fundamentals of Computer Graphics 3rd Edition by Peter Shirley, Steve Marschner | http://www.cs.cornell.edu/courses/cs4620/2019fa/

Anti-aliasing (4/6)

- Anti-aliasing:
  - Box filtering by supersampling

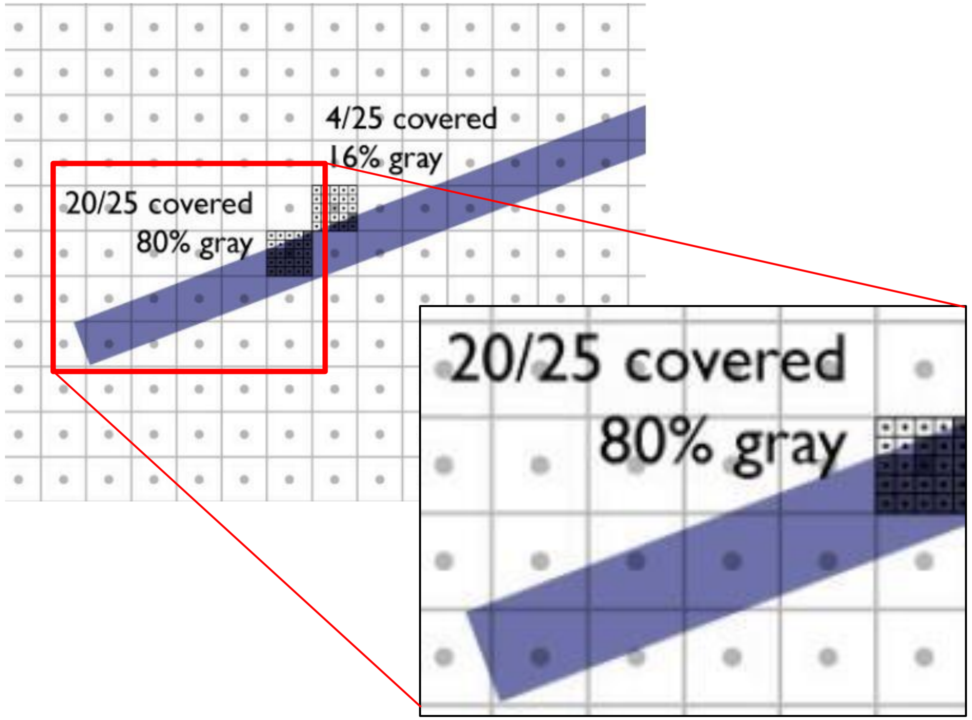




Credit: Fundamentals of Computer Graphics 3rd Edition by Peter Shirley, Steve Marschner | http://www.cs.cornell.edu/courses/cs4620/2019fa/

Anti-aliasing (5/6)

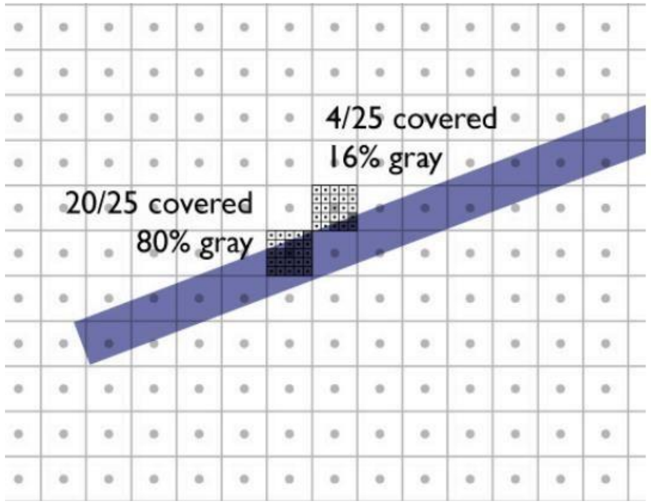
- Anti-aliasing:
  - Box filtering by supersampling

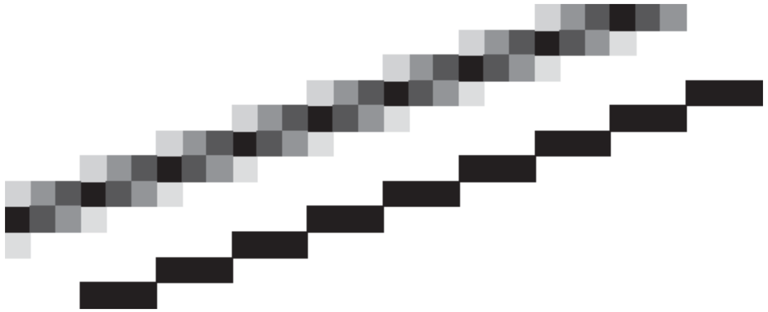


Credit: Fundamentals of Computer Graphics 3rd Edition by Peter Shirley, Steve Marschner | http://www.cs.cornell.edu/courses/cs4620/2019fa/

Anti-aliasing (6/6)

- Anti-aliasing:
  - Box filtering by supersampling

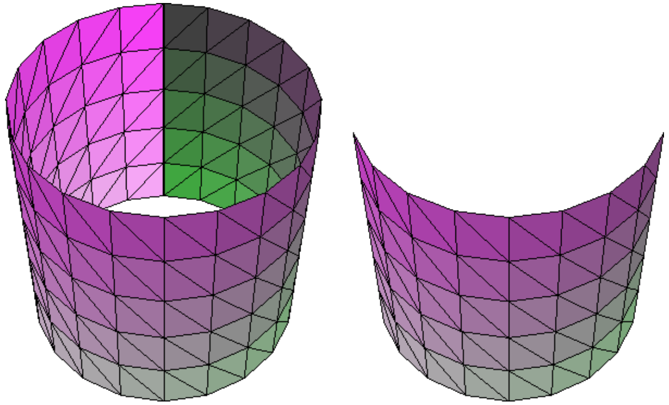




Credit: Fundamentals of Computer Graphics 3rd Edition by Peter Shirley, Steve Marschner | http://www.cs.cornell.edu/courses/cs4620/2019fa/

Backface Culling (1/3)

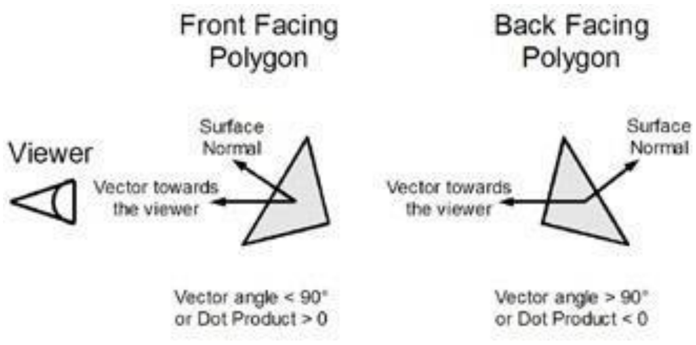
- Removal of primitives facing away from the camera.
  - Polygons that face away from the eye are certain to be overdrawn by polygons that face the eye.
    - Those polygons can be culled before the pipeline even starts.

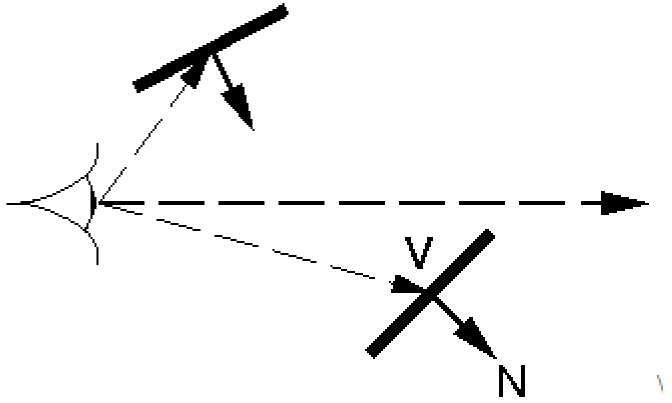


Credit: Fundamentals of Computer Graphics 3rd Edition by Peter Shirley, Steve Marschner | http://www.cs.cornell.edu/courses/cs4620/2019fa/

Backface Culling (2/3)

- If polygon normal is facing away from the viewer then it is "*backfacing*".
  - For solid objects, polygon will not be seen.
- Thus, if *N.V > 0* , then cull polygon.
  - *V* is vector from eye to point on polygon



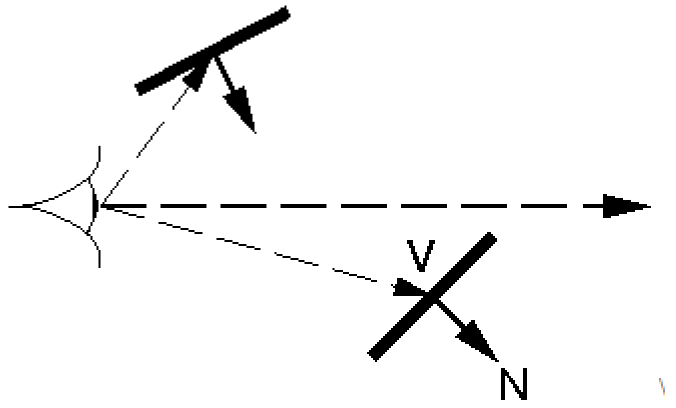


Credit: Fundamentals of Computer Graphics 3rd Edition by Peter Shirley, Steve Marschner | http://www.cs.cornell.edu/courses/cs4620/2019fa/

Backface Culling (3/3)

- If polygon normal is facing away from the viewer then it is "*backfacing*".
  - For solid objects, polygon will not be seen.
- Thus, if $N.V > 0$, then cull polygon.
  - V is vector from eye to point on polygon

Q: Disadvantage?



Credit: Fundamentals of Computer Graphics 3rd Edition by Peter Shirley, Steve Marschner | http://www.cs.cornell.edu/courses/cs4620/2019fa/

Practice Problem

- Verify Cyrus-Beck line clipping algorithm for different condition.