CS380: Computer Graphics
Clipping and Culling

Sung-Eui Yoon
(윤성의)

Course URL:
http://sglab.kaist.ac.kr/~sungeui/CG/

Class Objectives

- **Understand clipping and culling**
- **Understand view-frustum, back-face culling, and hierarchical culling methods**
- **Know various possibilities to perform culling and clipping in the rendering pipeline**

- **Related chapter:**
  - **Ch. 6: Clipping and Culling**

Culling and Clipping

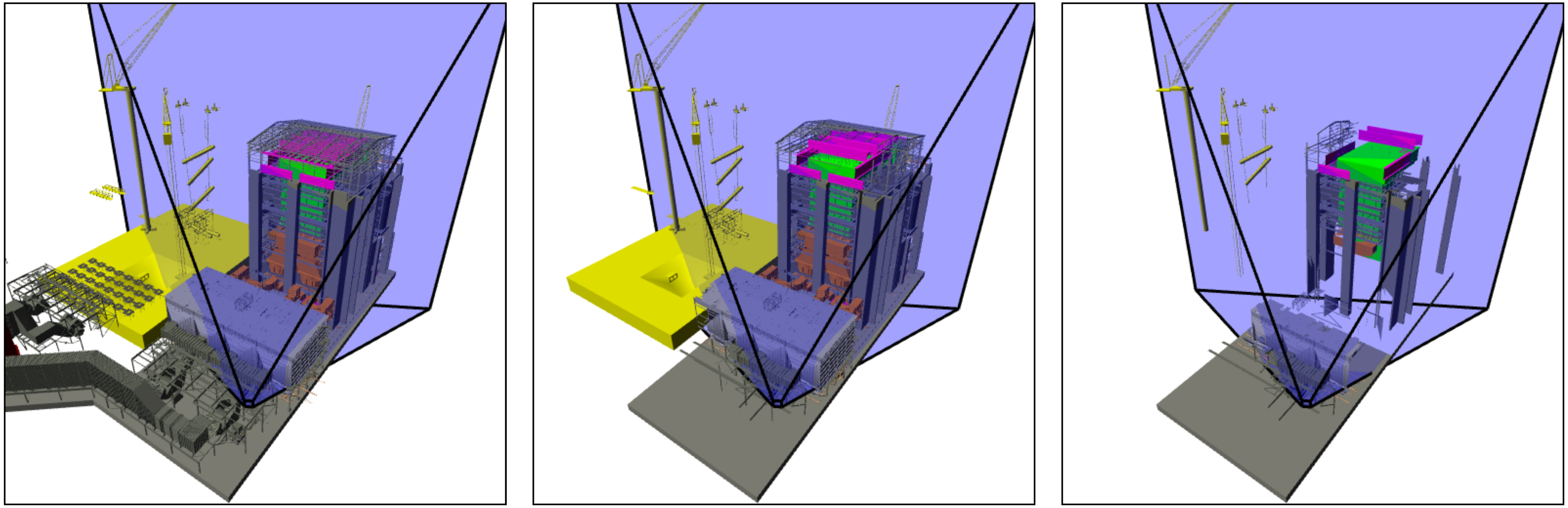
- Culling
  - Throws away entire objects and primitives that cannot possibly be visible
  - An important rendering optimization (esp. for large models)
- Clipping
  - "Clips off" the visible portion of a primitive
  - Simplifies rasterization
  - Also, used to create "cut-away" views of a model

Culling Example

Power plant model

(12 million triangles)

Culling Example



Full model 12 Mtris | View frustum culling 10 Mtris | Occulsion culling 1 Mtris

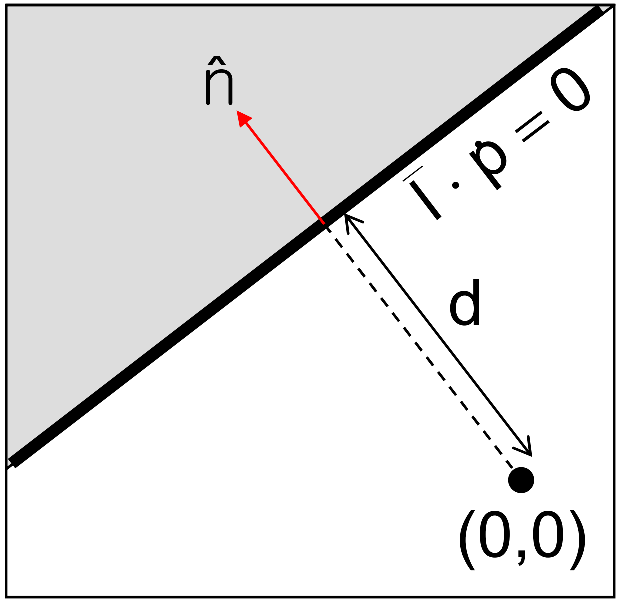
KAIST

Lines and Planes

- **Implicit equation for line (plane):**

$$n_x x + n_y y - d = 0$$

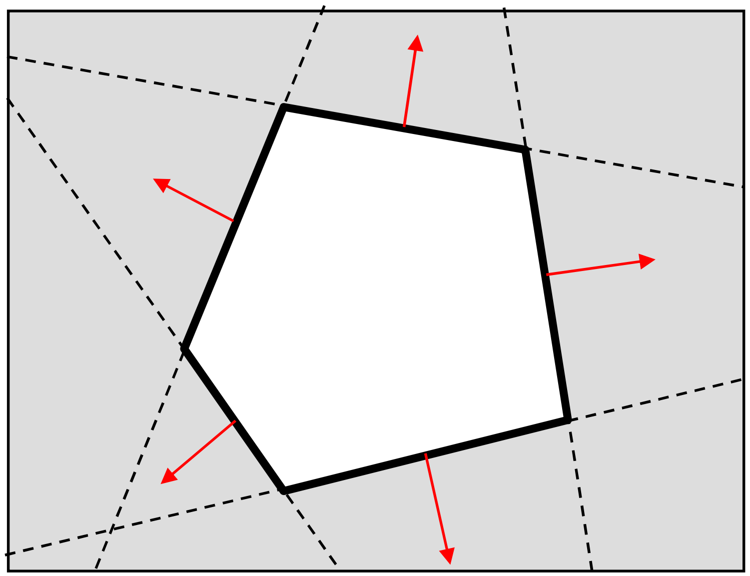
$$\begin{bmatrix} n_x & n_y & -d \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = 0 \Rightarrow T \cdot p = 0$$

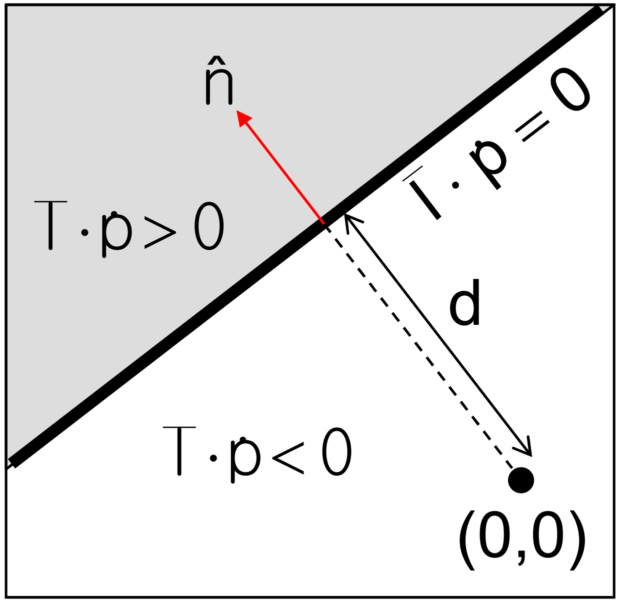


- **If n is normalized then d gives the distance of the line (plane) from the origin along n**

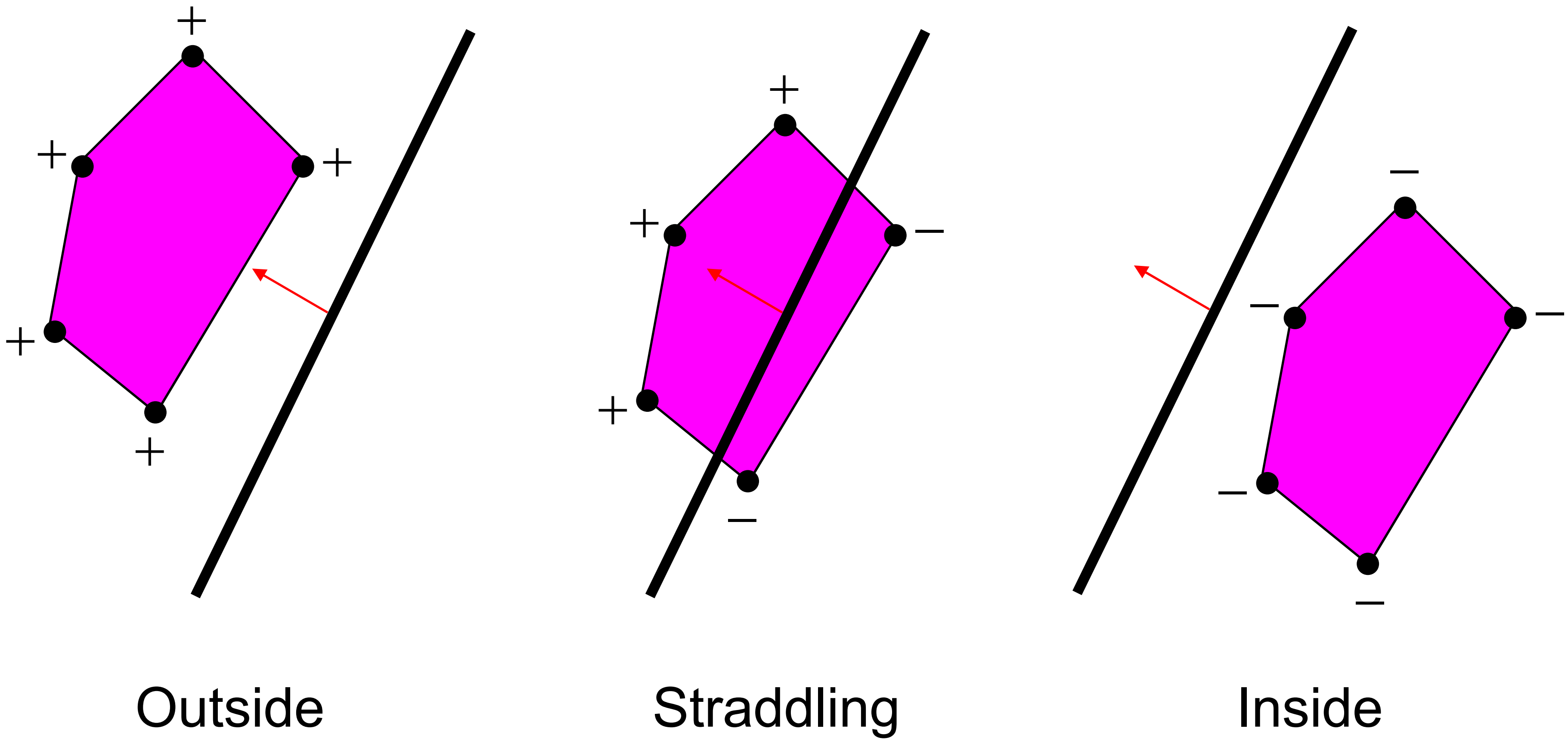
Lines and Planes

- Lines (planes) partition 2D (3D) space:
  - Positive and negative *half-spaces*
- The intersection of negative half-spaces defines a convex region

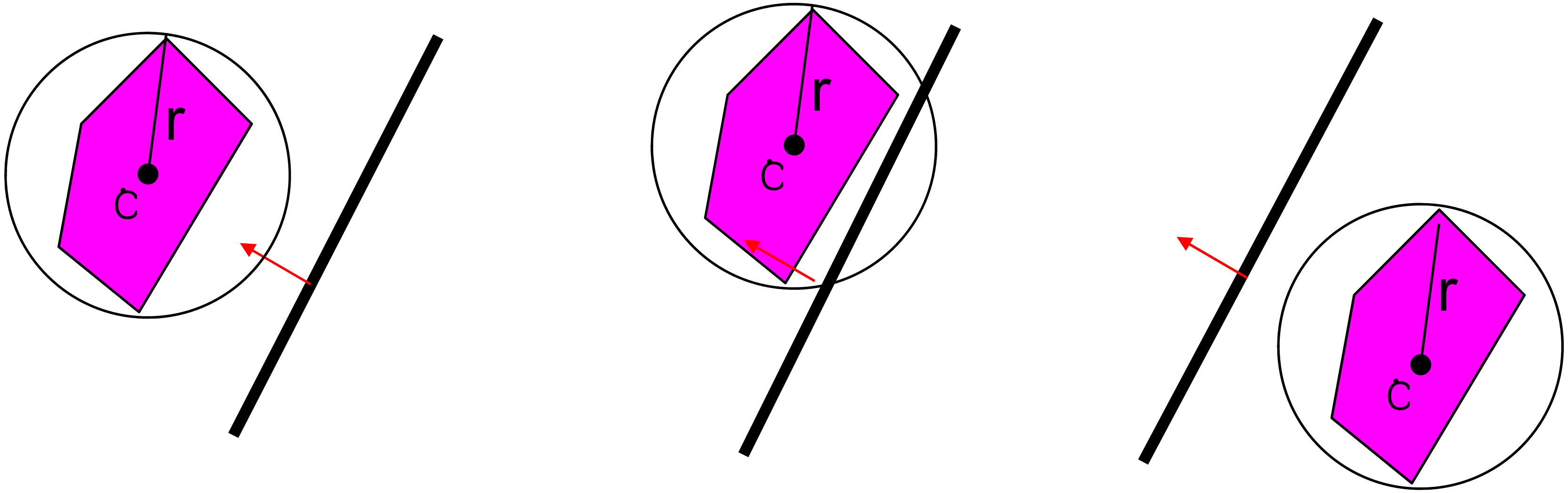




Testing Objects for Containment



Conservative Testing



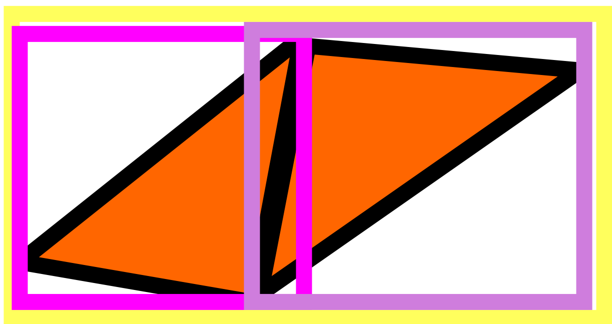
Outside	Indeterminate	Inside
$T \cdot \vec{c} > r$	$-r < T \cdot \vec{c} < r$	$T \cdot \vec{c} < -r$

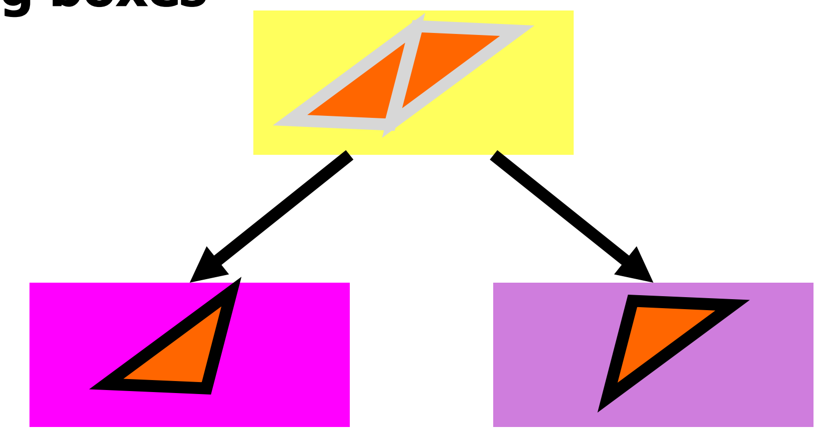
- **Use cheap, conservative bounds for trivial cases**
- **Can use more accurate, more expensive tests for ambiguous cases if needed**

Hierarchical Culling

- Bounding volume hierarchies accelerate culling by rejecting/accepting entire sub-trees at a time

- Bounding volume hierarchies (BVHs)
  - Object partitioning hierarchies
  - Uses axis-aligned bounding boxes

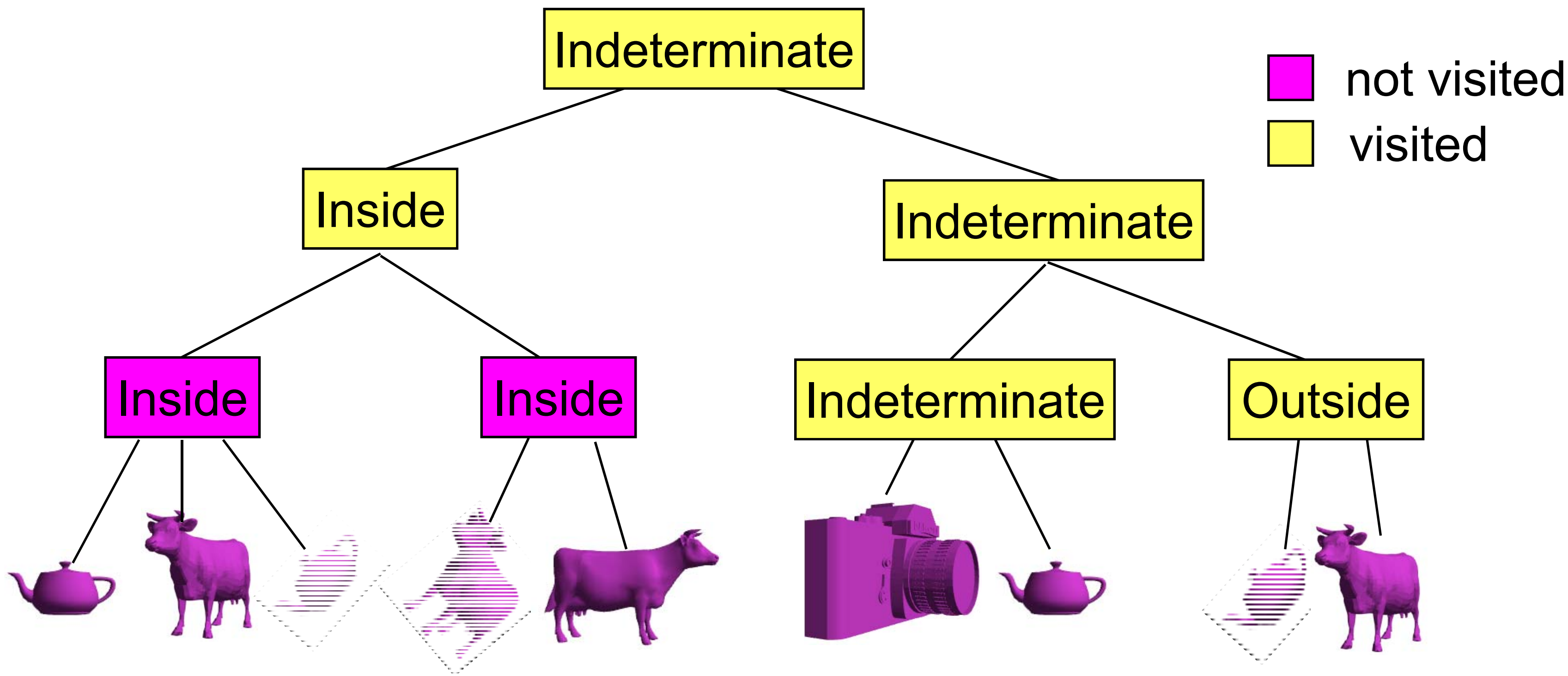




A BVH

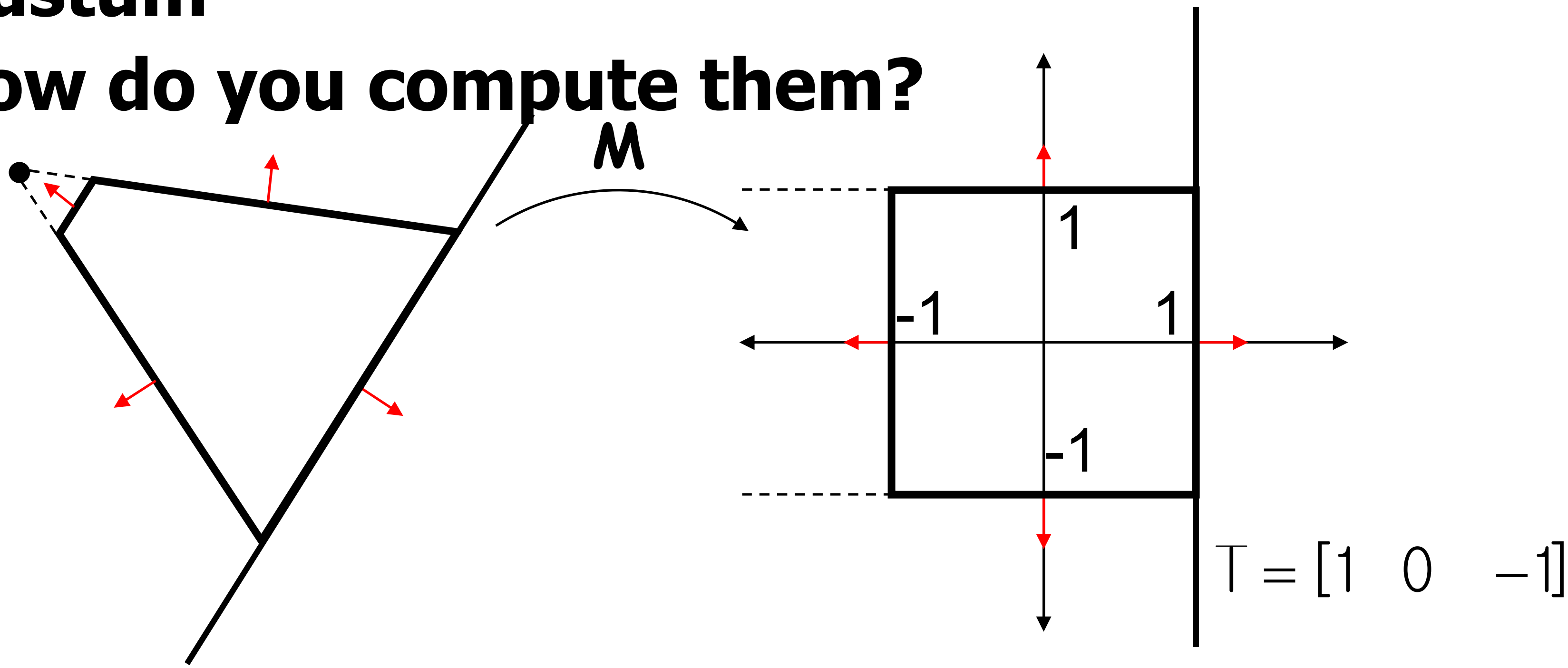
Hierarchical Culling

- **Simple algorithm:**
  while(node is indeterminate) recurse on children



View Frustum Culling

- **Test objects against planes defining view frustum**
- **How do you compute them?**

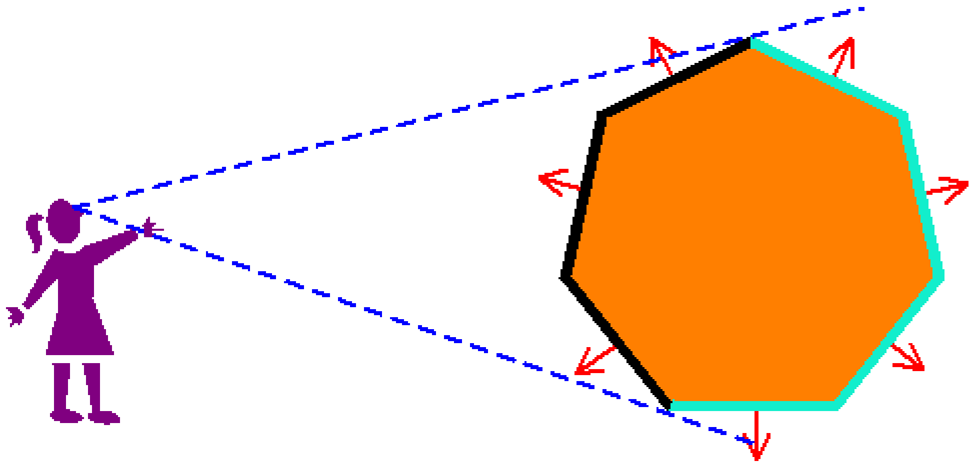


- **Other planes can be computed similarly**

KAIST

Back-Face Culling

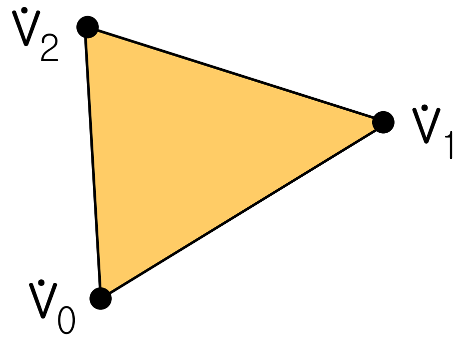
- Special case of occlusion - **convex self-occlusion**
  - For closed objects (has well-defined inside and outside) some parts of the surface must be blocked by other parts of the surface
- Specifically, the backside of the object is not visible



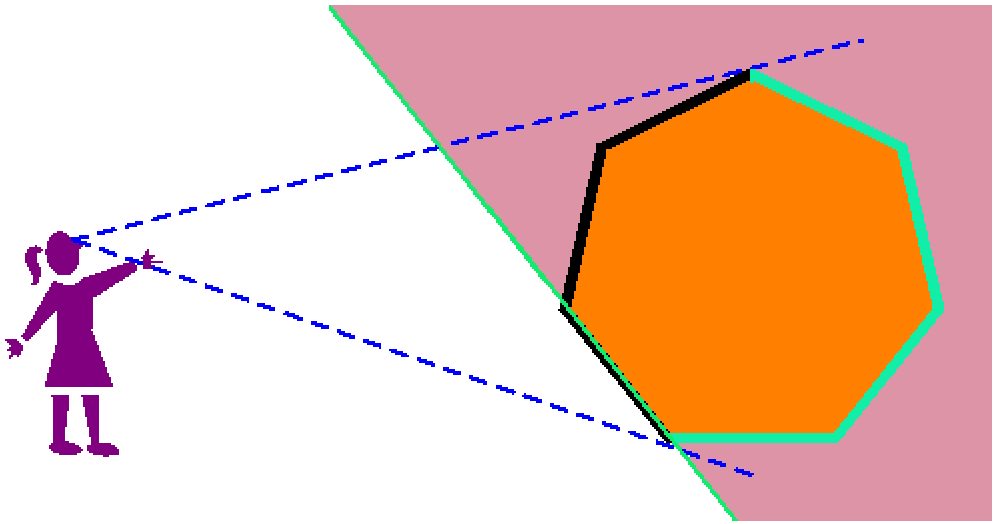
Face Plane Test

- **Compute the plane for the face:**

$$\vec{n} = (\dot{v}_1 - \dot{v}_0) \times (\dot{v}_2 - \dot{v}_0)$$
$$d = \vec{n} \cdot \dot{v}_0$$



- **Cull if eye point in the negative half-space**



Back-Face Culling in OpenGL

- **Can cull front faces or back faces**
- **Back-face culling can sometimes double performance**

```
if (cull):
    glFrontFace(GL_CCW)          # define winding order
    glEnable(GL_CULL_FACE)       # enable Culling
    glCullFace(GL_BACK)          # which faces to cull
else:
    glDisable(GL_CULL_FACE)
```

You can also do front-face culling!

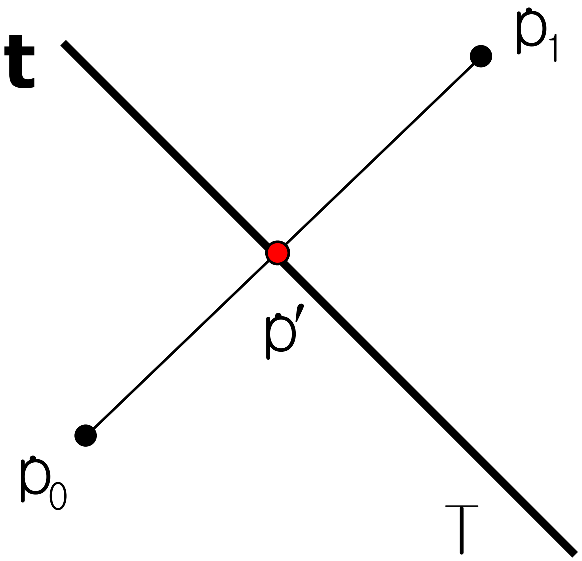
Clipping a Line Segment against a Line

- **First check endpoints against the plane**
  - **If they are on the same side, no clipping is needed**
- **Interpolate to get new point**

$$p' = p_0 + t\,(p_1 - p_0) \qquad T \cdot p' = 0$$

$$T \cdot (p_0 + t\,(p_1 - p_0)) = 0$$

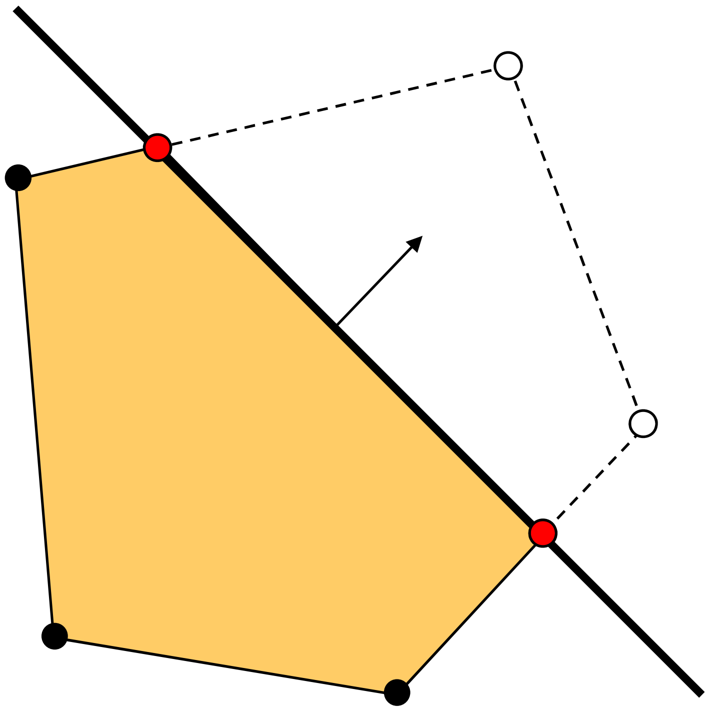
$$t = \frac{-(T \cdot p_0)}{T \cdot (p_1 - p_0)}$$



- **Vertex attributes interpolated the same way**

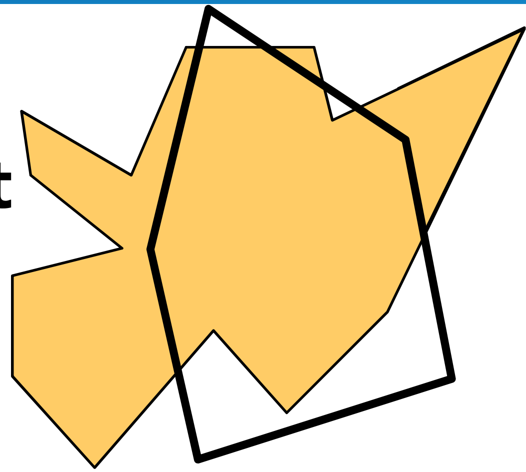
Clipping a Polygon against a Line

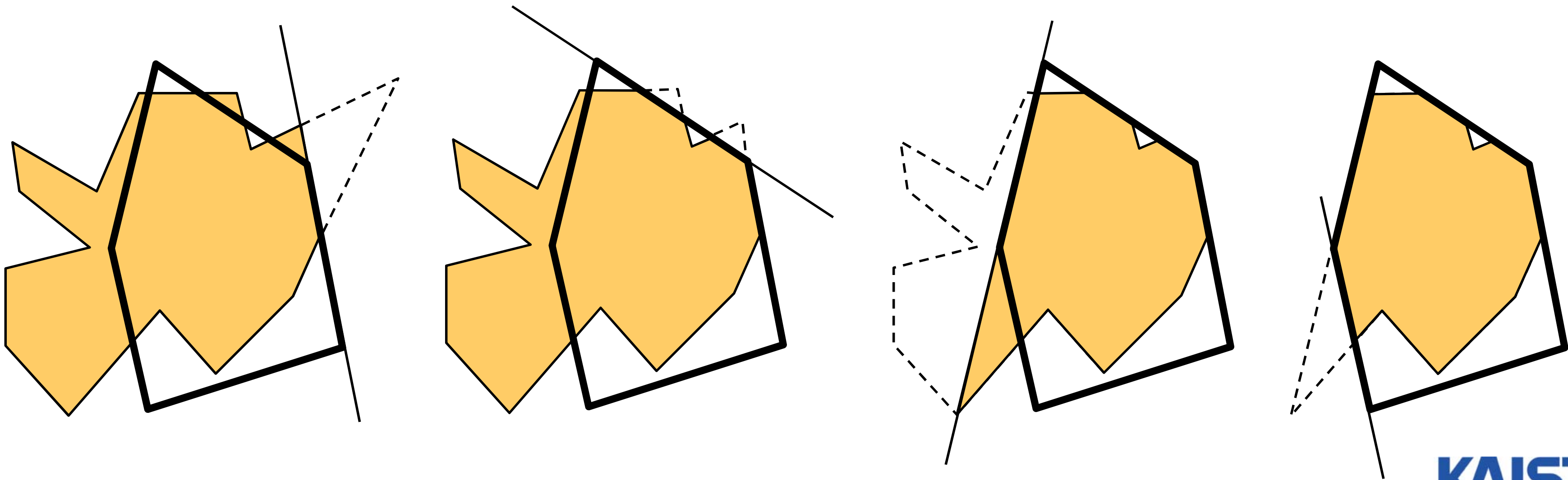
- Traverse edges
- Keep edges that are entirely inside
- Create new point when we exit
- Throw away edges entirely outside
- Create new point and new edge when we enter



Clipping against a Convex Region

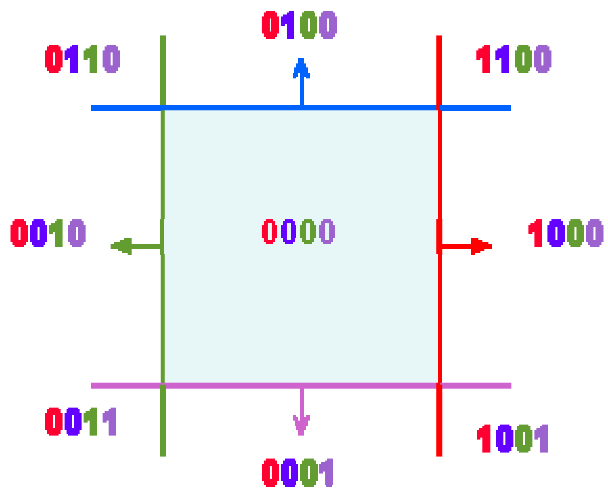
- **Sutherland-Hodgman**
  - **Just clip against one edge at a time**





Outcodes

- The Cohen-Sutherland clipping algorithm uses **outcodes** to quickly determine the visibility of a primitive
- An outcode is created for each vertex
  - It is a bit vector with a bit set for each plane the vertex is outside of
- Works for any convex region



KAIST

Outcode for Lines

`(outcode1 OR outcode2) == 0`

line segment is inside

`(outcode1 AND outcode2) != 0`

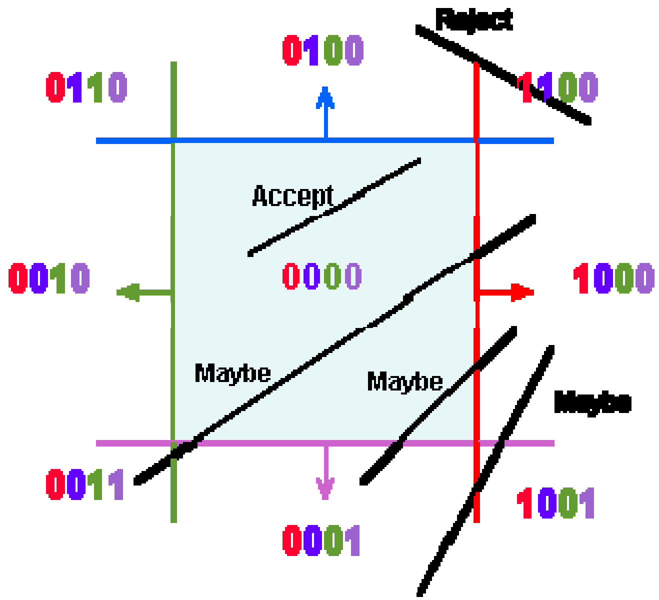
line segment is totally outside

`(outcode1 AND outcode2) == 0`

line segment potentially crosses clip region

at planes indicated by set bits in

`(outcode1 XOR outcode2)`



- False positive
  - **Some line segments that are classified as potentially crossing the clip region actually don't**

Outcodes for Triangles

Combine outcodes from vertices

```
(outcode1 OR outcode2 OR outcode3) == 0
```

triangle is inside

```
(outcode1 AND outcode2 AND outcode3) != 0
```

triangle is outside

```
(outcode1 AND outcode2 AND outcode3) == 0
```

triangle potentially crosses clip region

Clipping in the Pipeline

$(x, y, z, 1)$ — Option 1

↓

Various transformations

↓

(x', y', z', w') — Option 2

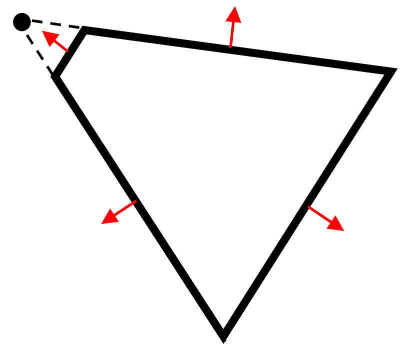
↓

Homogeneous divide

↓

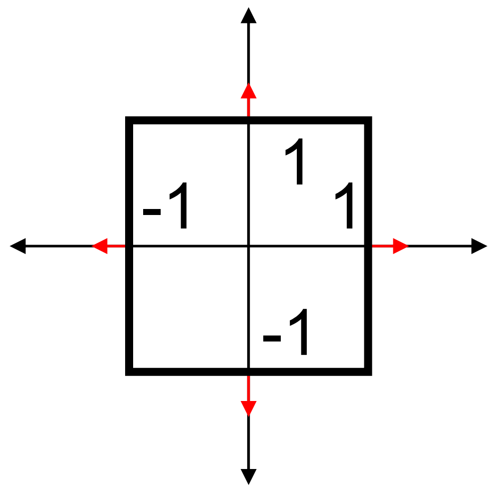
$(x'/w', y'/w', z'/w', 1)$ — Option 3 (NDC space)

↓



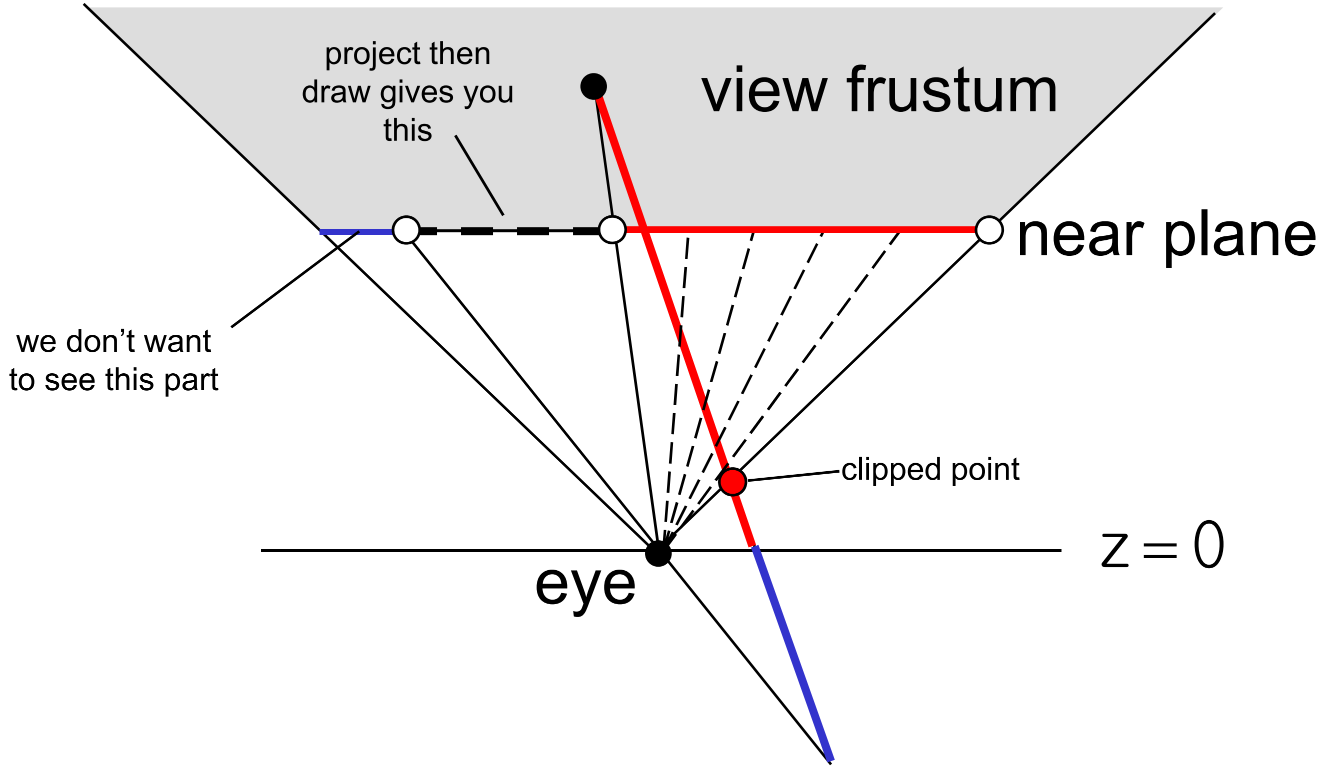
What is the best place?

- Option 2 (clip space)



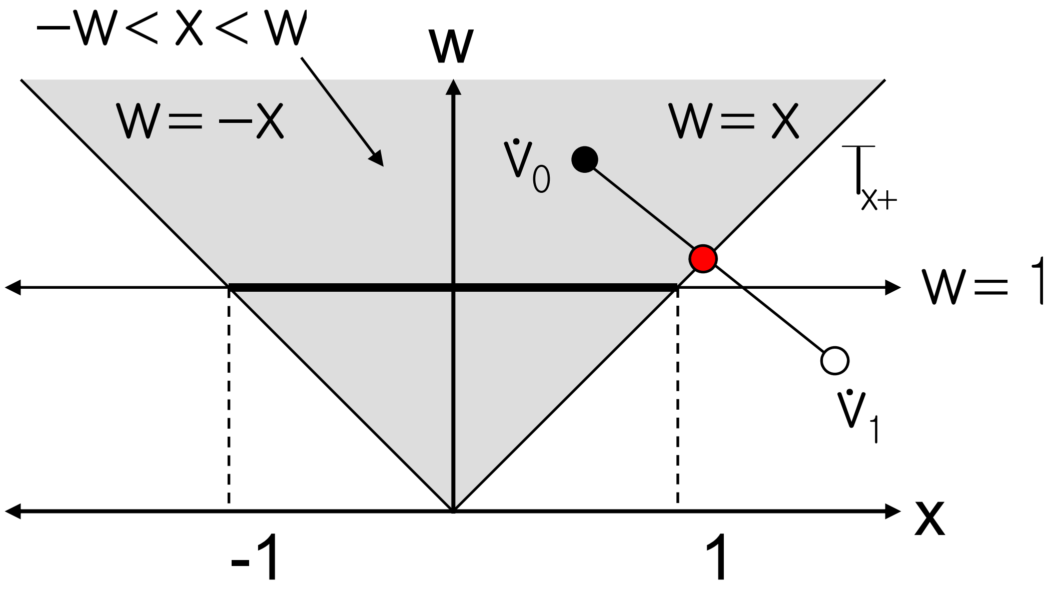
View Frustum Clipping

- **Points in projective space need to be clipped <u>before</u> projection**
- **Primitives that straddle the z=0 plane "flip" around infinity when projected**



Clipping in the Clip Space

- **NDC simplify view frustum clipping**
- **Clip after applying projection matrix, but before the divide by w**
  - **clip coordinates**



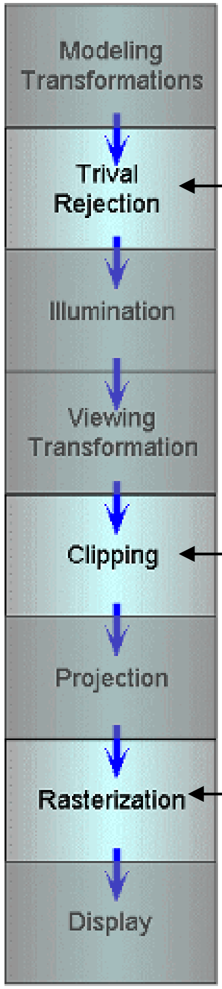
$$T_{x+} = [1 \quad -1 \quad 0]$$

$$\dot{v}_i = [x_i \quad w_i \quad 1]^T$$

$$t = \frac{w_0 - x_0}{(w_0 - x_0) - (w_1 - x_1)}$$

- **Easy in/out test and interpolation**

Culling and Clipping in the Rendering Pipeline



View frustum culling, but performed in the application level

View frustum clipping and back-face culling can be done here

Back-face culling done in setup phase of rasterization

Class Objectives were:

- **Understand clipping and culling**
- **Understand view-frustum, back-face culling, and hierarchical culling methods**
- **Know various possibilities to perform culling and clipping in the rendering pipeline**

Next Time

- Triangulating a polygon
- Rasterizing triangles
- Interpolating parameters