CS380: Computer Graphics

Clipping and Culling

Sung-Eui Yoon
(윤성의)

Course URL:
http://sgvr.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

Questions on last lecture

- the lecture slide specifies that we need to submit the question 2 times in a semester. does it mean “at least” two times? if not, this is my third question so is it going to be disregarded?

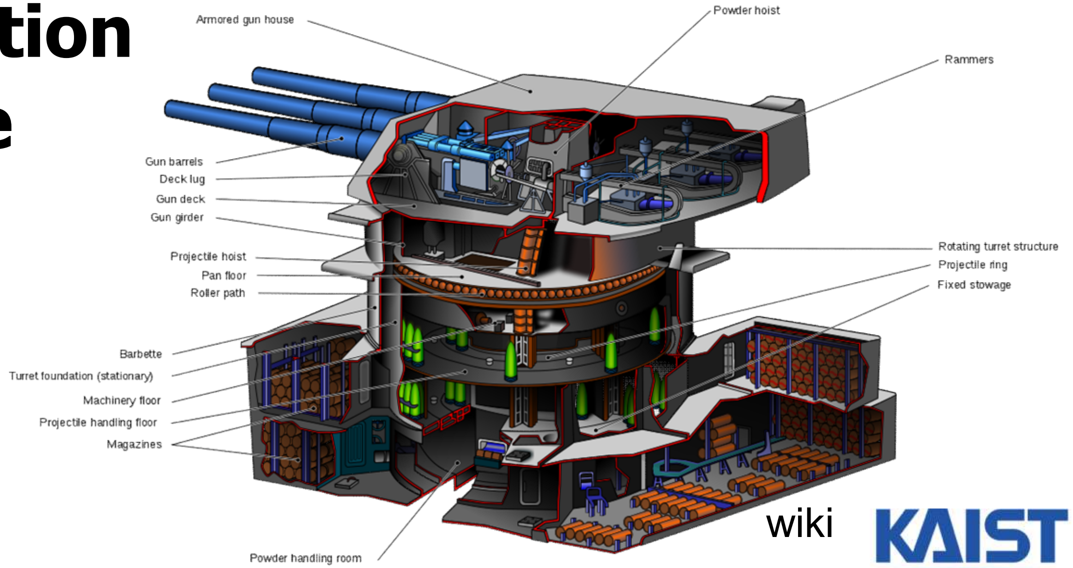
- my previous questions were left unanswered. may I ask them via e-mail to the professor?

Questions on last lecture

- It is mentioned in the class today that GPUs are made up to deal with specific tasks so their cores are much simpler than CPU.
- But I remember that in the earlier lecture it was also mentioned that GPU is getting more flexible and could run more programs, isn’t this flexibility making GPU become like CPU? isn’t it breaking our primary reasoning to separate GPU and CPU in the first place?

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



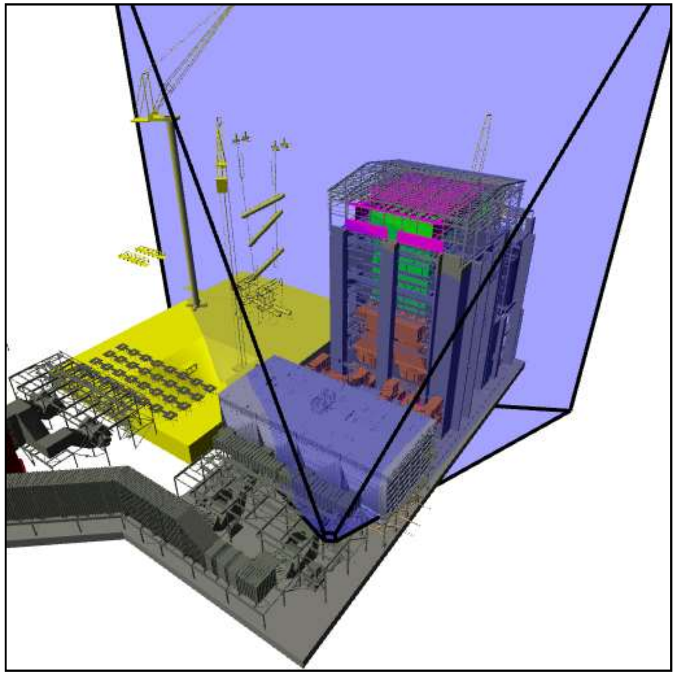
wiki

Culling Example

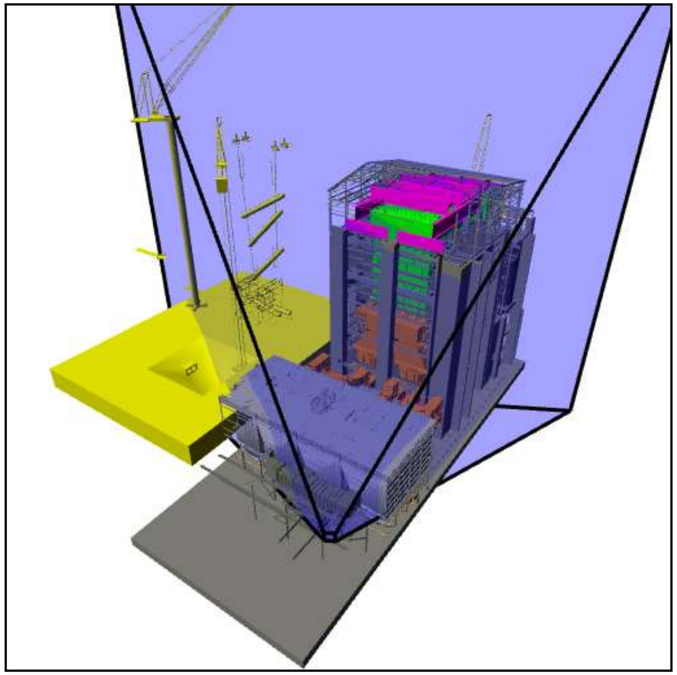
Power plant model

(12 million triangles)

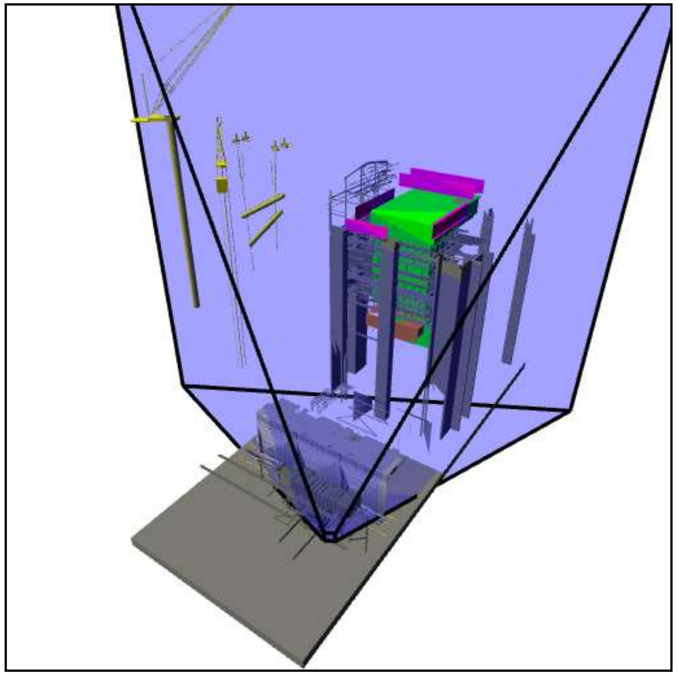
Culling Example



Full model
12 Mtris



View frustum culling
10 Mtris



Occulsion culling
1 Mtris

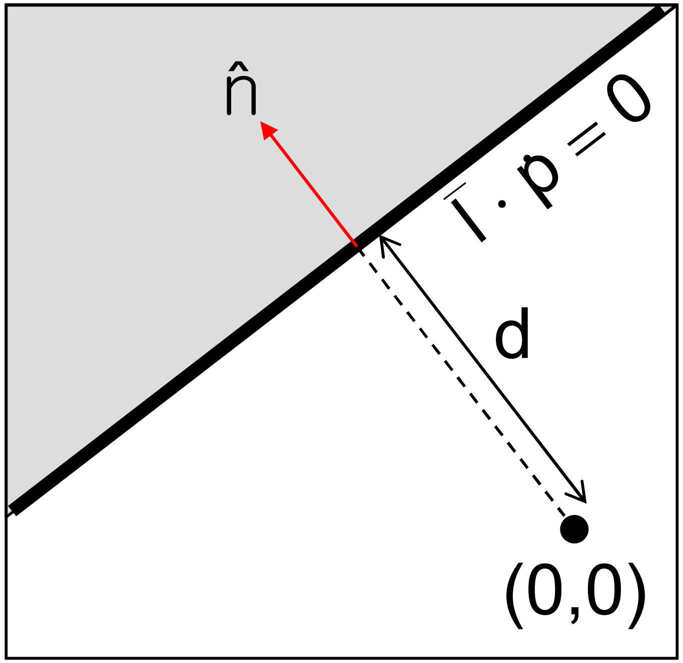
Lines and Planes

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

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

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

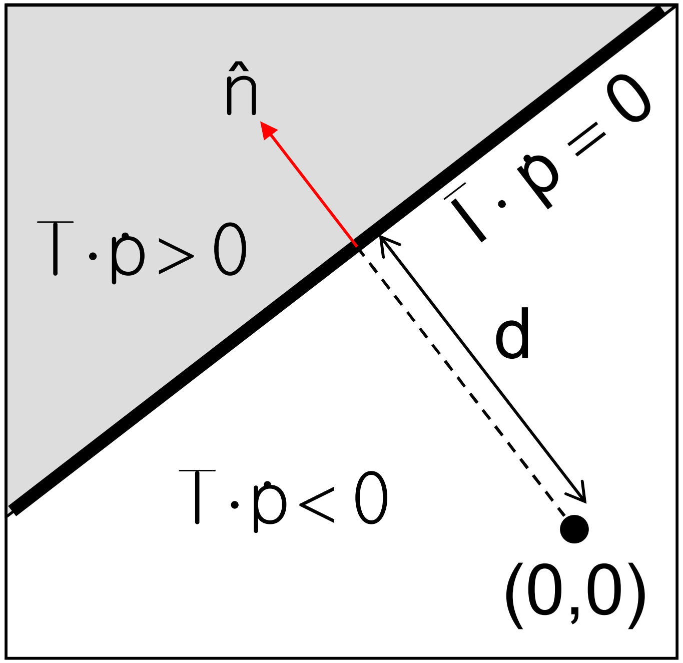
$$\Rightarrow \quad \bar{l} \cdot \dot{p} = 0$$

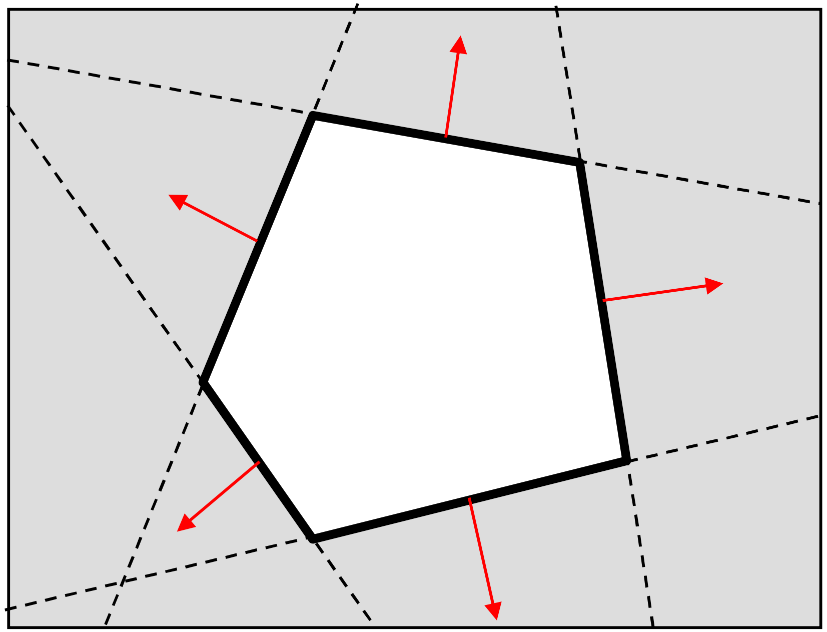


- **If $\vec{n}$ is normalized then d gives the distance of the line (plane) from the origin along $\vec{n}$**

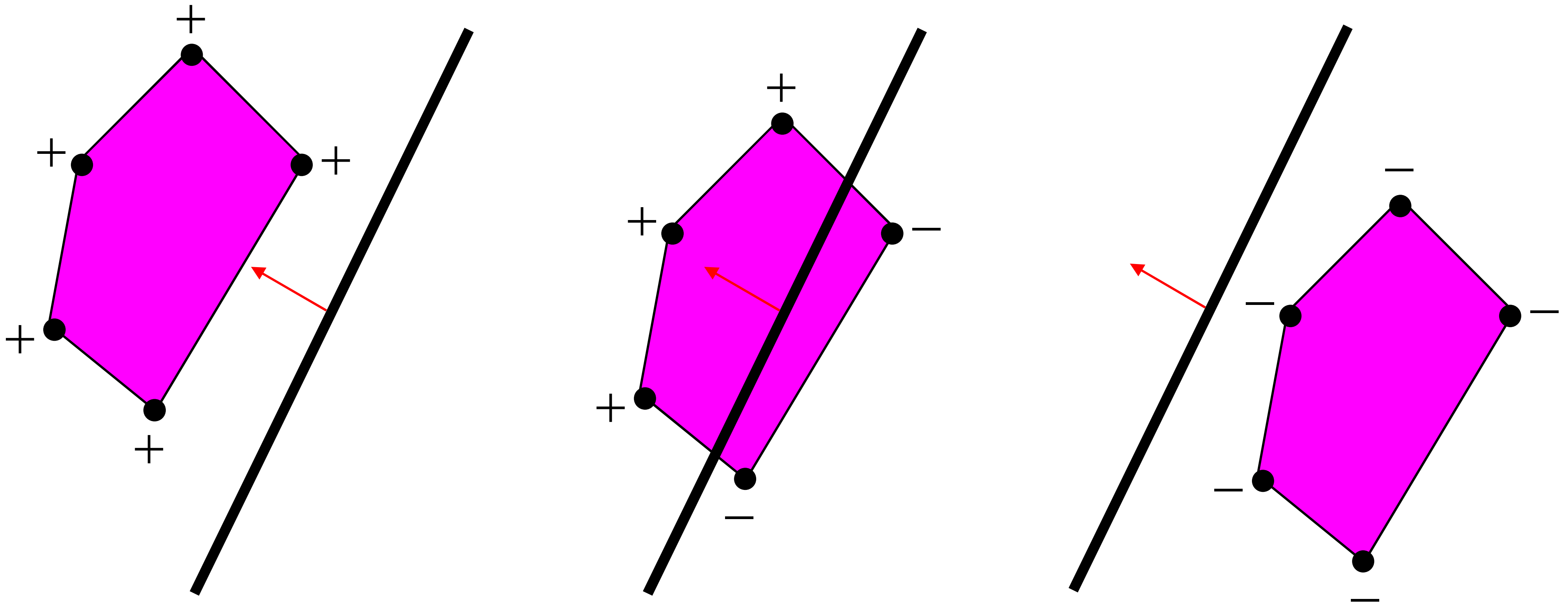
Lines and Planes

- **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

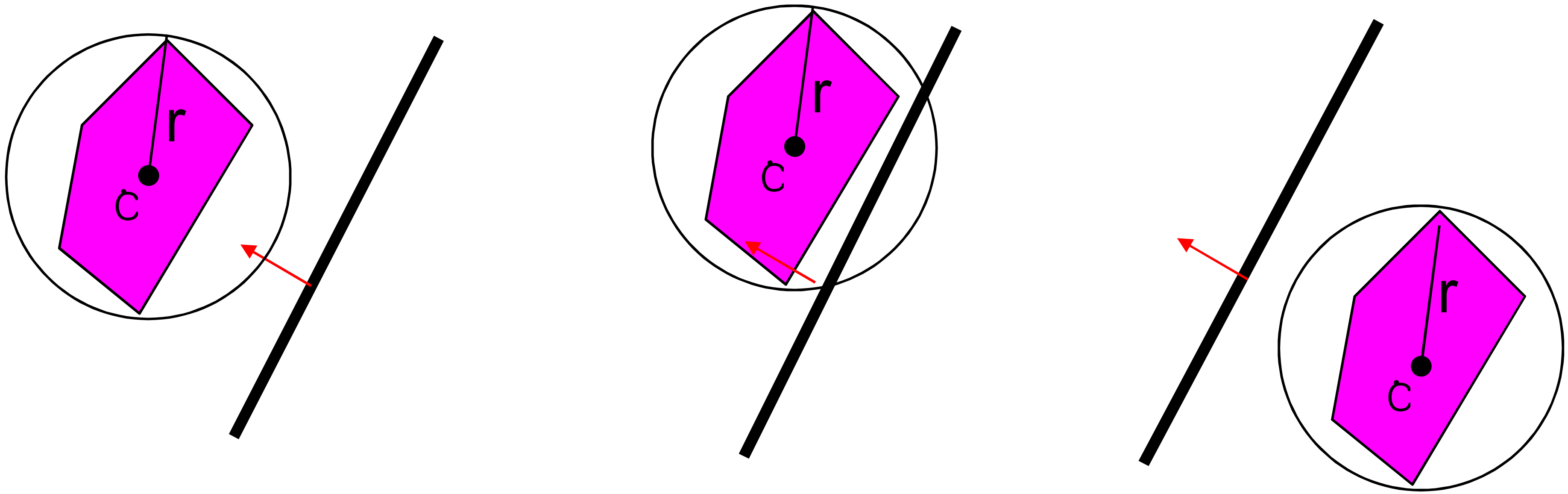


Outside

Straddling

Inside

Conservative Testing



Outside

$\bar{l} \cdot \dot{c} > r$

Indeterminate

$-r < \bar{l} \cdot \dot{c} < r$

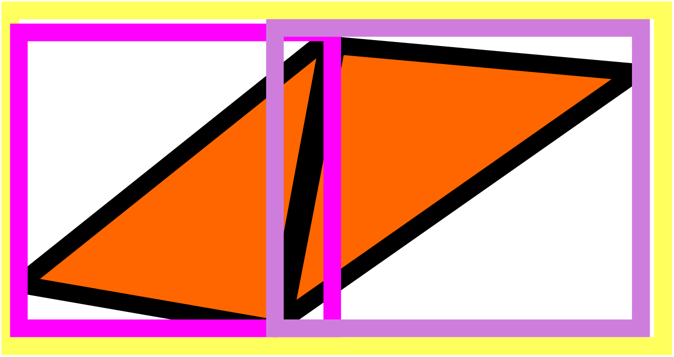
Inside

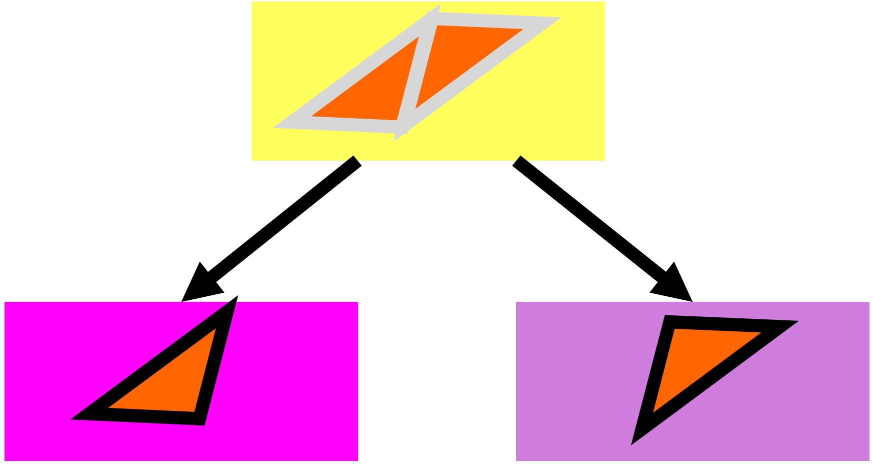
$\bar{l} \cdot \dot{c} < -r$

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

Hierarchical Culling

- **Bounding volume hierarchies (BVHs)**
  - **Accelerate culling by rejecting/accepting entire sub-trees at a time**
  - **Uses axis-aligned bounding boxes**
  - **Also known as object partitioning hierarchies**

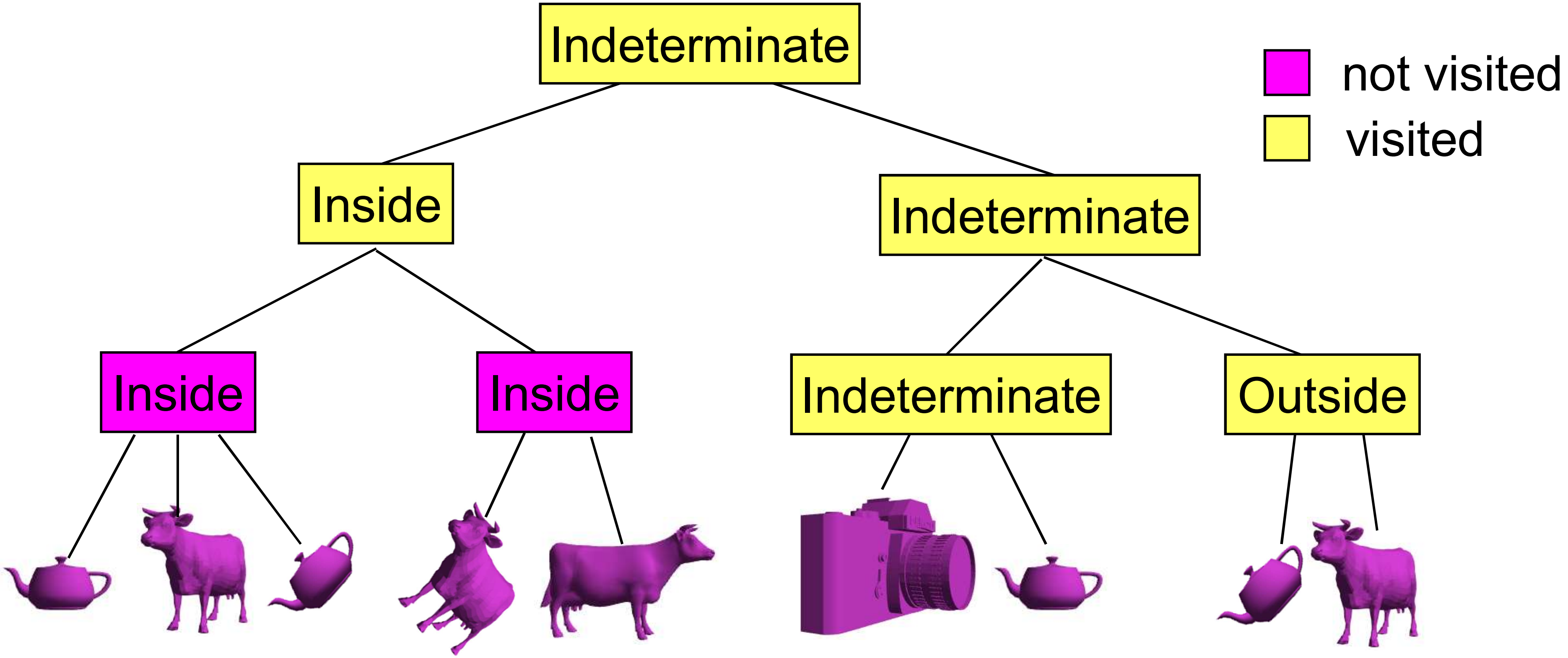




A BVH

Hierarchical Culling w/ BVH

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



Test-Of-Time 2006 Award

High-Performance Graphics 2015

Los Angeles, August 7–9, 2015

RT-DEFORM: Interactive Ray Tracing of Dynamic Scenes using BVHs

Christian Lauterbach, Sung-eui Yoon, David Tuft, Dinesh Manocha

IEEE Interactive Ray Tracing, 2006

RT-DEFORM: Interactive Ray Tracing of Dynamic Scenes using BVHs

Christian Lauterbach[*] University of North Carolina at Chapel Hill

Sung-Eui Yoon[†] Lawrence Livermore National Laboratory

David Tuft[‡] University of North Carolina at Chapel Hill

Dinesh Manocha[§] University of North Carolina at Chapel Hill

Figure 1: Dress simulation: Four different images of a 210 step sequence taken from a dynamic cloth simulation and consisting of 40K triangles. By updating in real-time instead of rebuilding the BVH of the deforming model according to our heuristic, we are able to render the animation at 13 frames per second with 512^2 screen resolution using a dual-core P4 processor at 2.8 GHz.

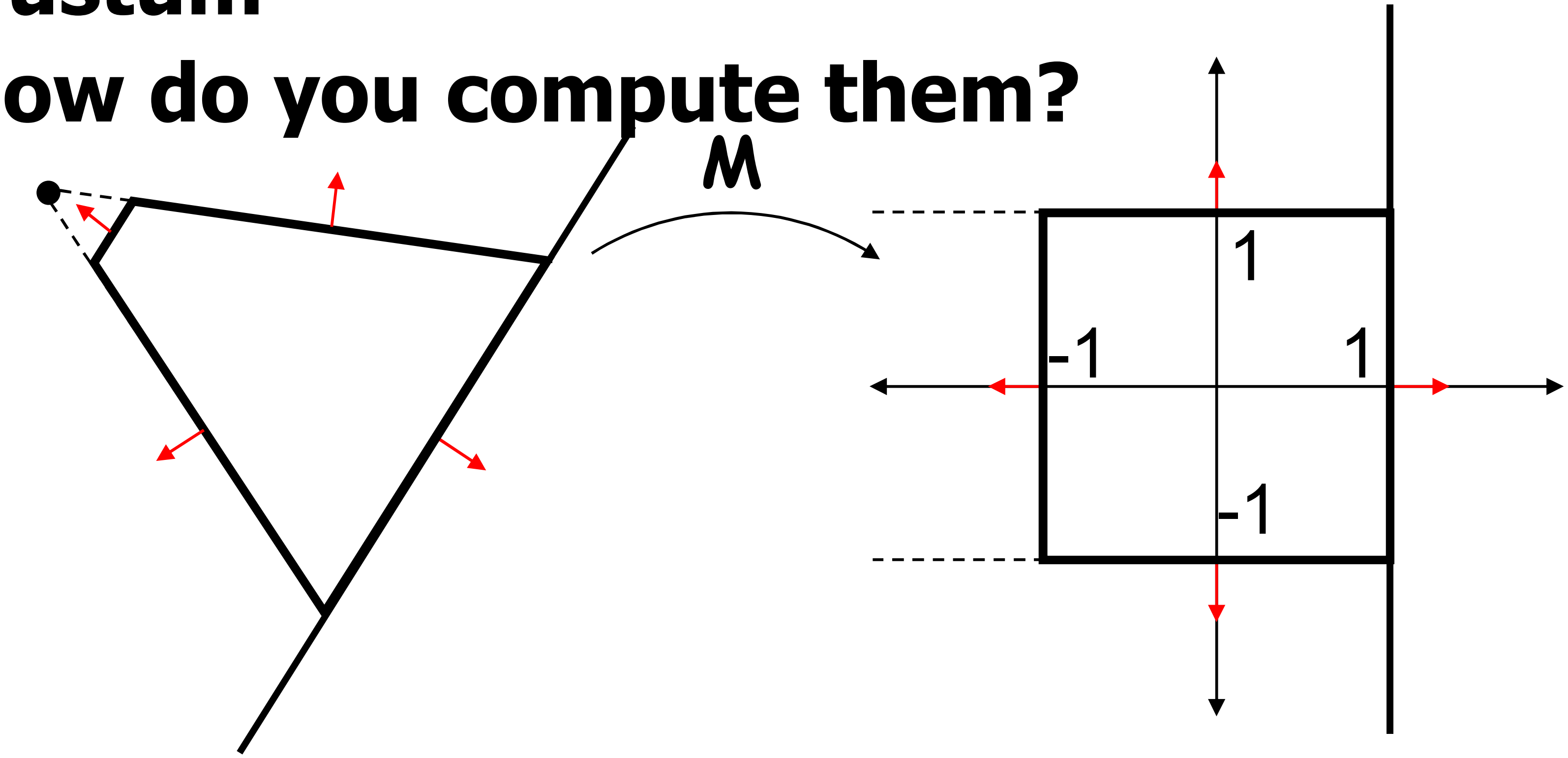
ABSTRACT

We present an efficient approach for interactive ray tracing of deformable or animated models. Unlike many of the recent approaches for ray tracing static scenes, we use bounding volume hierarchies (BVHs) instead of kd-trees as the underlying acceleration structure. Our algorithm makes no assumptions about the

earlier ray tracing algorithms were used to generate high quality images for offline rendering, but in the last few years, there has been renewed interest in real-time ray tracing. At a broad level, most of the work in real-time ray tracing algorithms can be classified into three main categories: improved techniques to compute acceleration structures, exploiting ray coherence, and parallel algorithms on shared memory or distributed memory systems.

View Frustum Culling

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

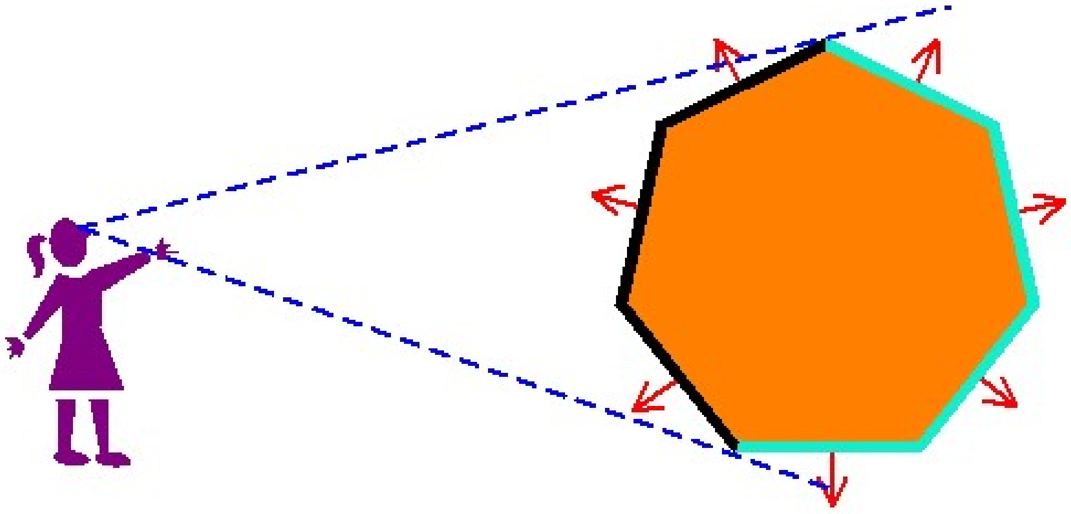


$$\bar{l} = [1 \quad 0 \quad -1]$$

- Other planes can be computed similarly

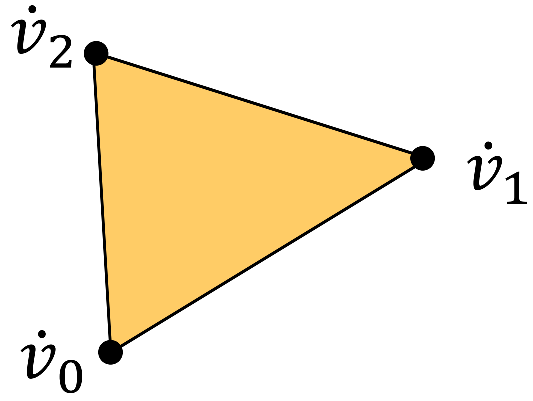
Back-Face Culling

- 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



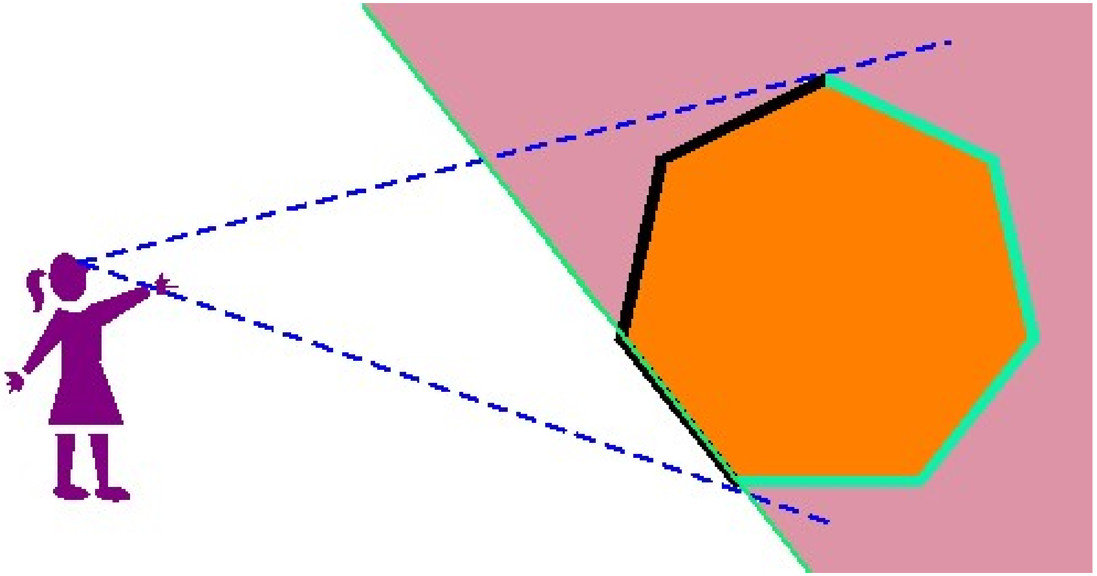
Face Plane Test

- **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!

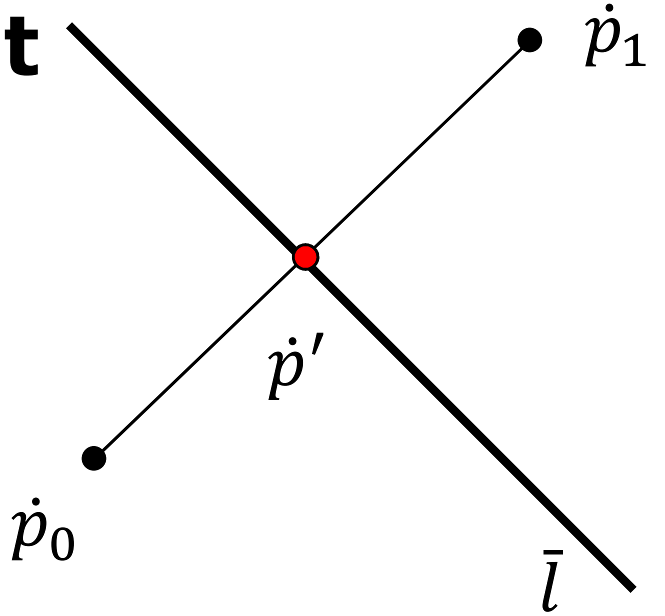
Clipping a Line Segment against a Line

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

$$\dot{p}' = \dot{p}_0 + t(\dot{p}_1 - \dot{p}_0) \qquad \bar{l} \cdot \dot{p}' = 0$$

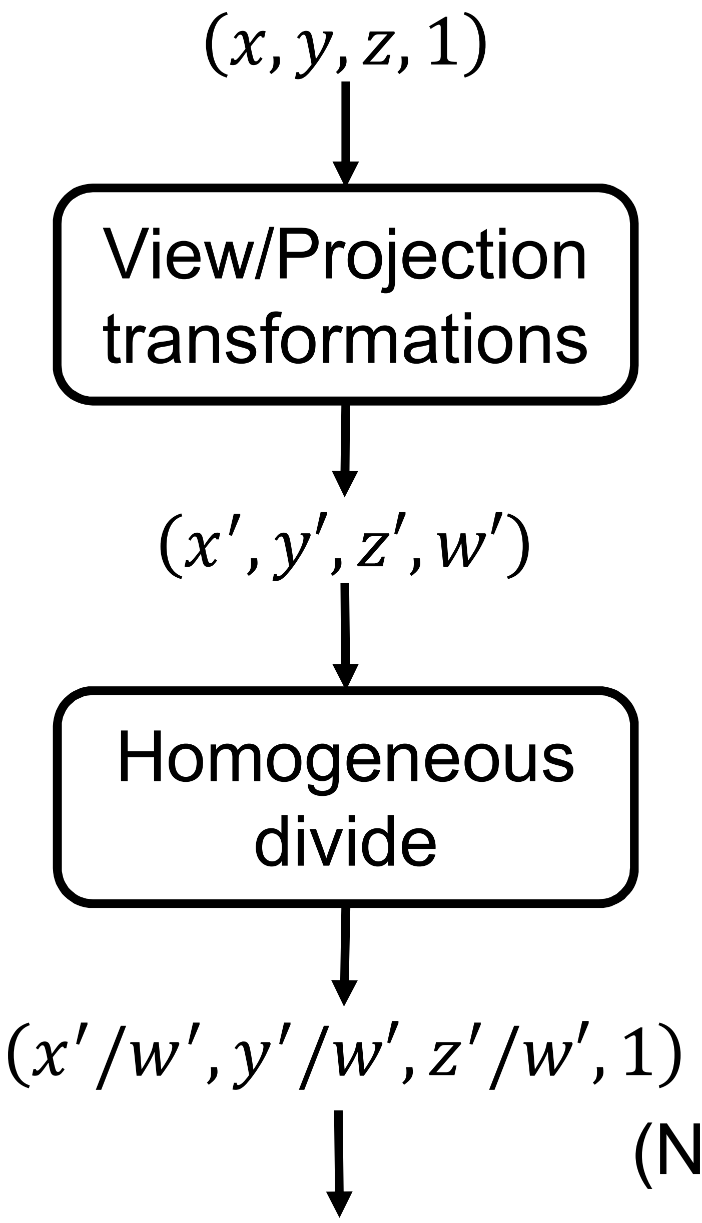
$$\bar{l} \cdot (\dot{p}_0 + t(\dot{p}_1 - \dot{p}_0)) = 0$$

$$t = \frac{-(\bar{l} \cdot \dot{p}_0)}{\bar{l} \cdot (\dot{p}_1 - \dot{p}_0)}$$



- **Vertex attributes interpolated the same way**

Clipping in the Pipeline



$(x, y, z, 1)$

View/Projection transformations

(x', y', z', w')

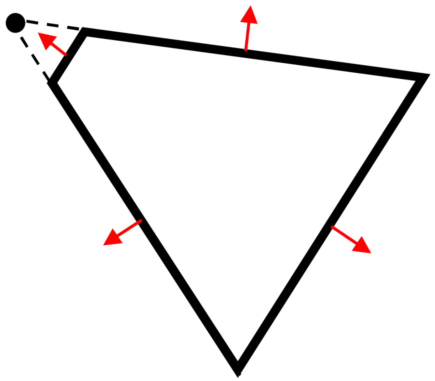
Homogeneous divide

$(x'/w', y'/w', z'/w', 1)$

Option 1

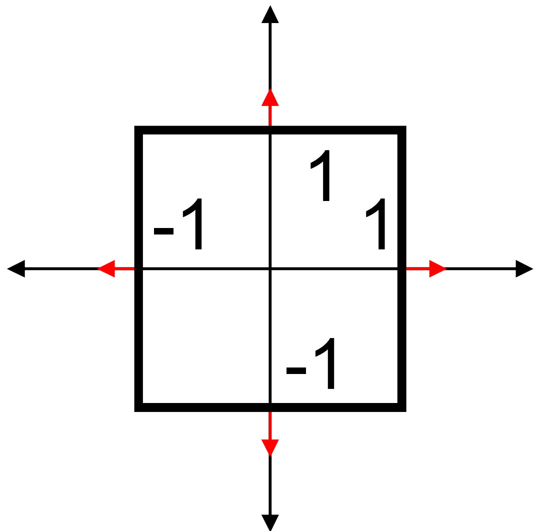
Option 2

Option 3 (NDC space)



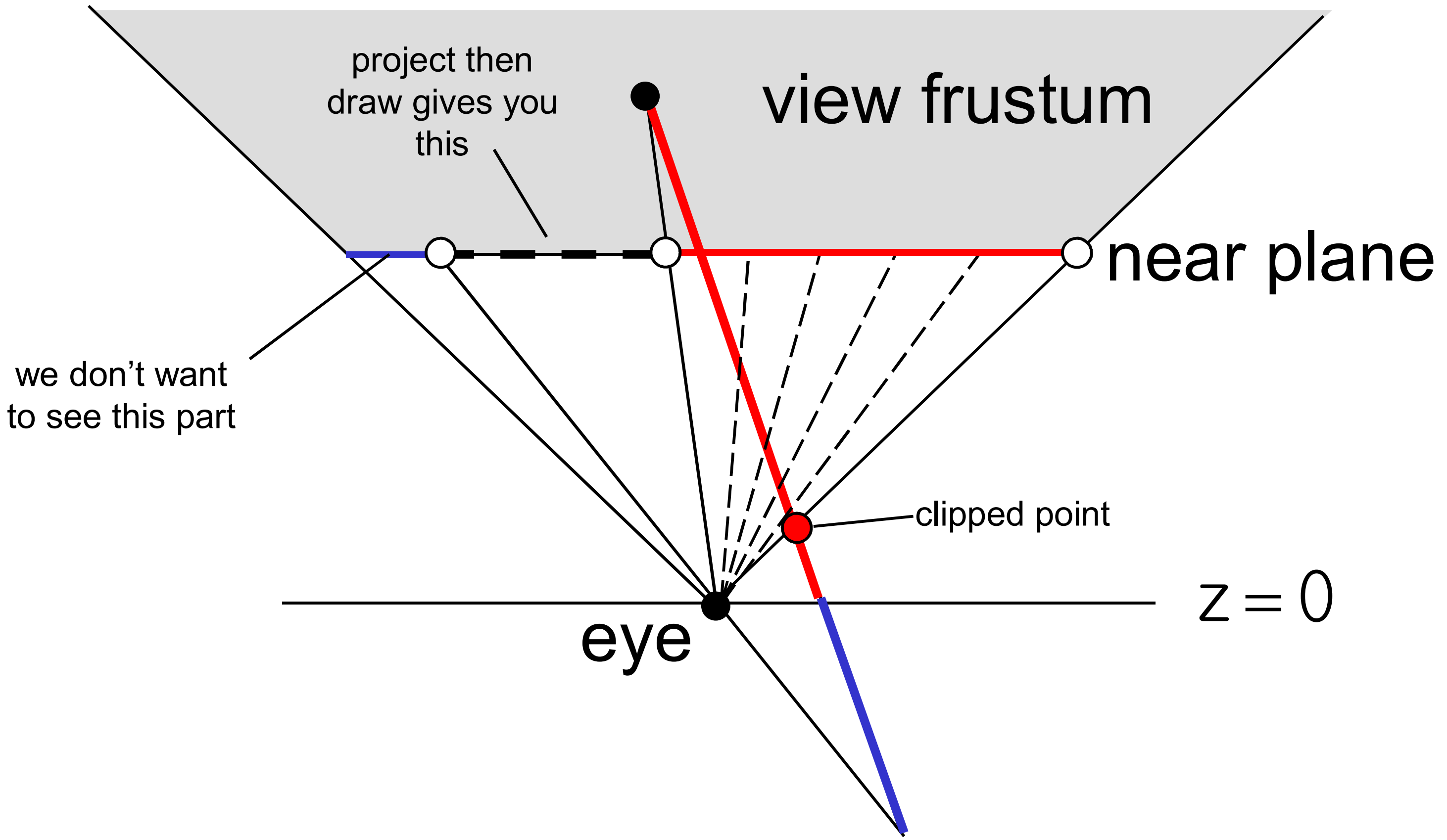
What is the best place?

- Option 2 (clip space)



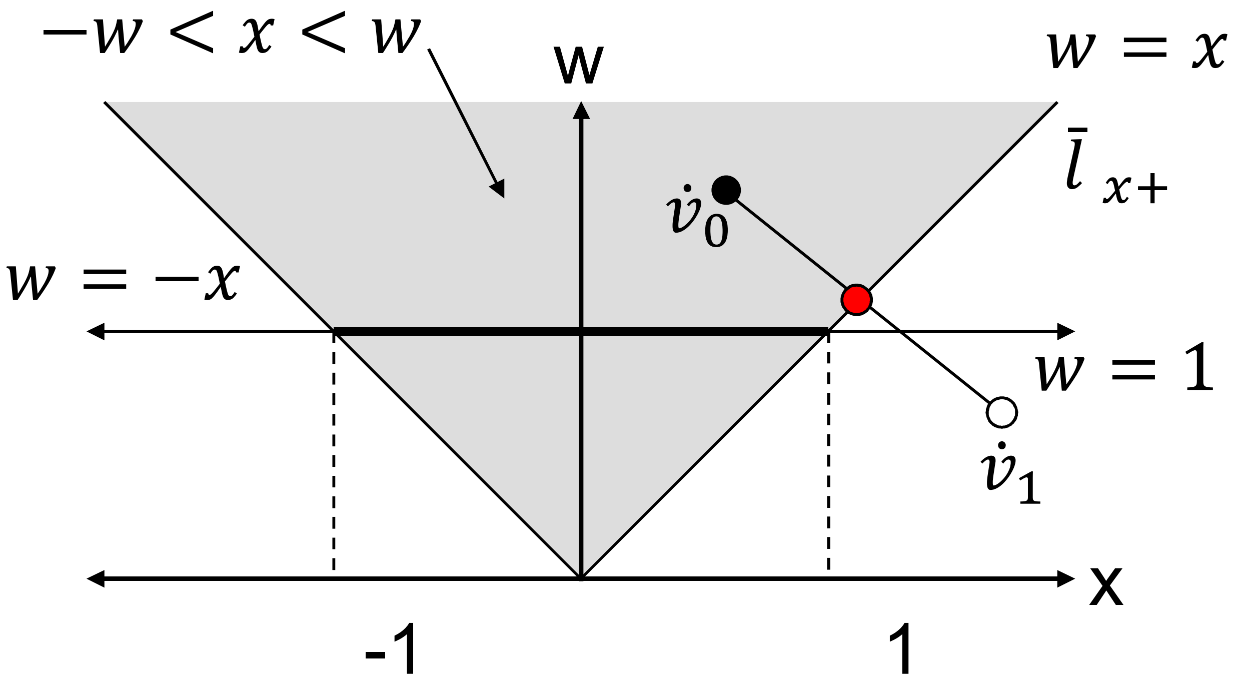
View Frustum Clipping in NDC Space

- Points in projective space need to be clipped before projection
- Primitives that straddle the z=0 plane “flip” around infinity when projected


project then draw gives you this
view frustum
near plane
we don’t want to see this part
clipped point
$z = 0$
eye

Clipping in the Clip Space

- **NDC simplify view frustum clipping**
- **Clip after applying projection matrix, but before the divide by w; we call that space clip space**



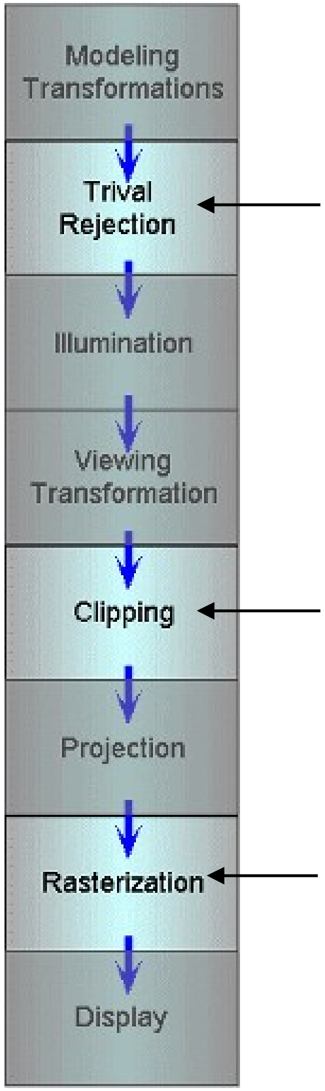
$$\bar{l}_{x+} = [1 \quad -1 \quad 0]$$

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

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

- **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**

Homework

- Go over the next lecture slides before the class
- Watch 2 SIGGRAPH videos and submit your summaries before every Mon. class
- Submit your questions two times during the whole semester

Next Time

- Rasterizing triangles
    - Triangulating a polygon
    - Interpolating parameters