Course Updates

- Assignment 2 questions?

- Midterm is on Wednesday
  - Material: up to mid-lecture today
  - Review lecture notes (up to and including set 6 – “Camera Models”)
  - Sample exam on the web (but includes material we did not cover)

- Tutorial this week
  - Finish reviewing assignment 1
  - Review of the rendering pipeline

- Assignment 2 starter code is available

Camera Models
Part 3

Computer Graphics, CSCD18

Fall 2007

Instructor: Leonid Sigal

Last time …

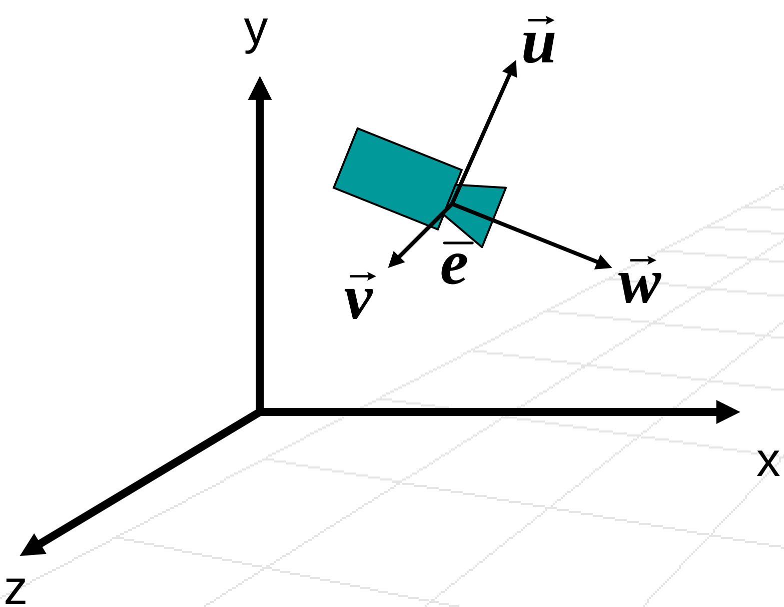
- Camera models
  - Perspective Projection
    - Similar triangles derivation
    - Algebraic derivation
  - Camera position and orientation
    - Transforming a point from camera coordinates to world coordinates
    - Transforming a point from world coordinates to camera coordinates
  - Homogeneous Perspective

Putting together a camera model

- Projecting a world point to image (film) plane

$$\overline{x}^* = M_p M_{wc} \overline{p}^w$$

$$\overline{x}^* = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/f & 0 \end{bmatrix} \begin{bmatrix} A^T & -A^T \overline{e} \\ [0,0,0] & 1 \end{bmatrix} \overline{p}^w$$



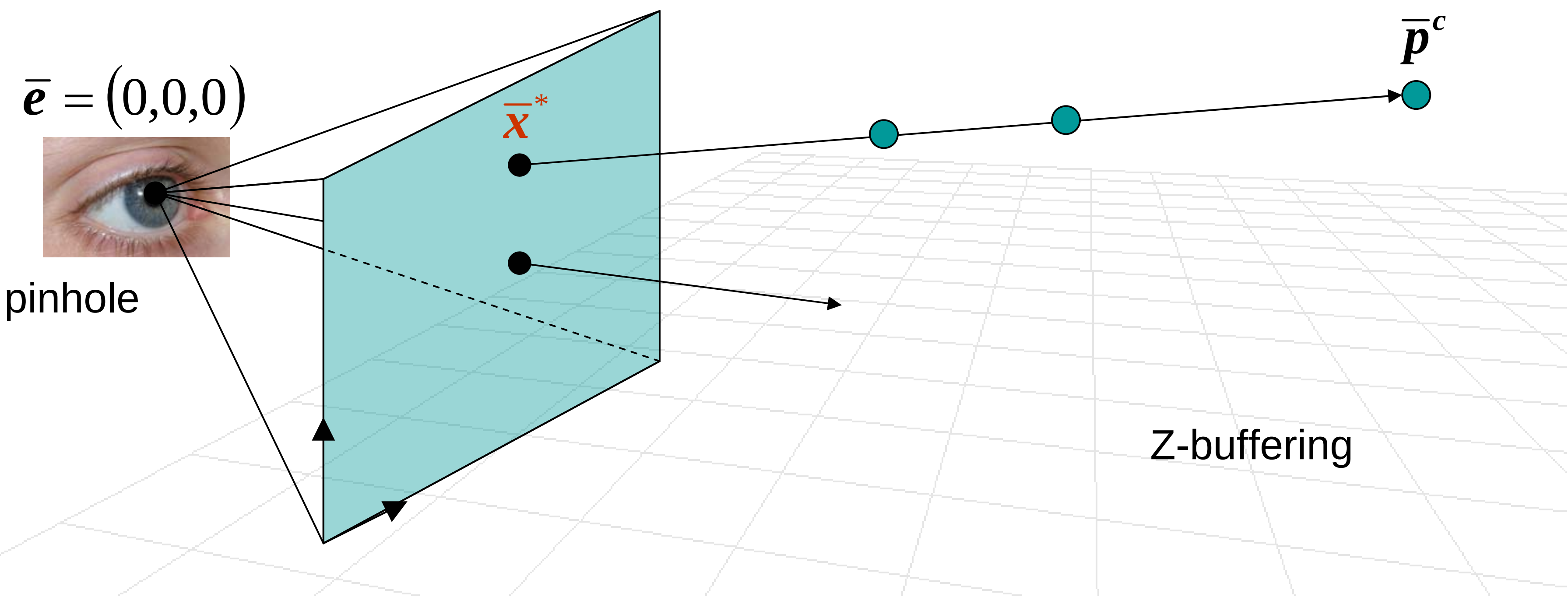
where $A^T = \begin{bmatrix} \leftarrow & \vec{u} & \rightarrow \\ \leftarrow & \vec{v} & \rightarrow \\ \leftarrow & \vec{w} & \rightarrow \end{bmatrix}$

Last time …

- Camera models
  - Perspective Projection
    - Similar triangles derivation
    - Algebraic derivation
  - Camera position and orientation
    - Transforming a point from camera coordinates to world coordinates
    - Transforming a point from world coordinates to camera coordinates
  - Homogeneous Perspective
  - Pseudodepth

Pseudodepth

- We would like to change the projection transform so that z-component of the projection gives us useful information (not just a constant f)
- We want it to encode something about depth of a point. Why?



Pseudodepth

- Standard homogeneous perspective projection

$$\boldsymbol{M}_p = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/\boldsymbol{f} & 0 \end{bmatrix}$$

- Pseudodepth projection matrix

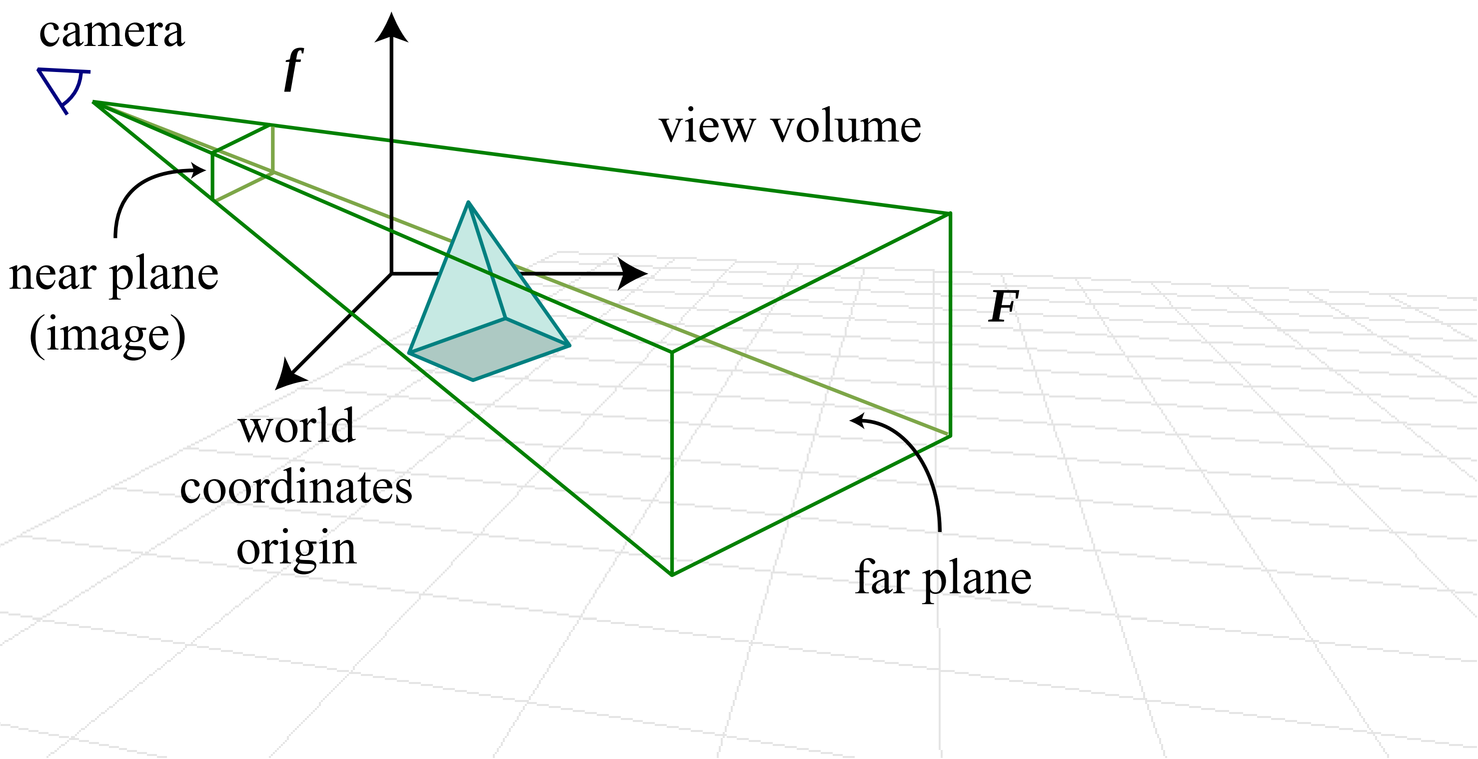
$$\boldsymbol{M}_p = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \boldsymbol{a} & \boldsymbol{b} \\ 0 & 0 & 1/\boldsymbol{f} & 0 \end{bmatrix}$$

$$\boldsymbol{z}^* = \frac{\boldsymbol{f}}{\boldsymbol{p}_z^c}\left(\boldsymbol{a}\boldsymbol{p}_z^c + \boldsymbol{b}\right)$$

Pseudodepth

- How do we pick ***a*** and ***b***?

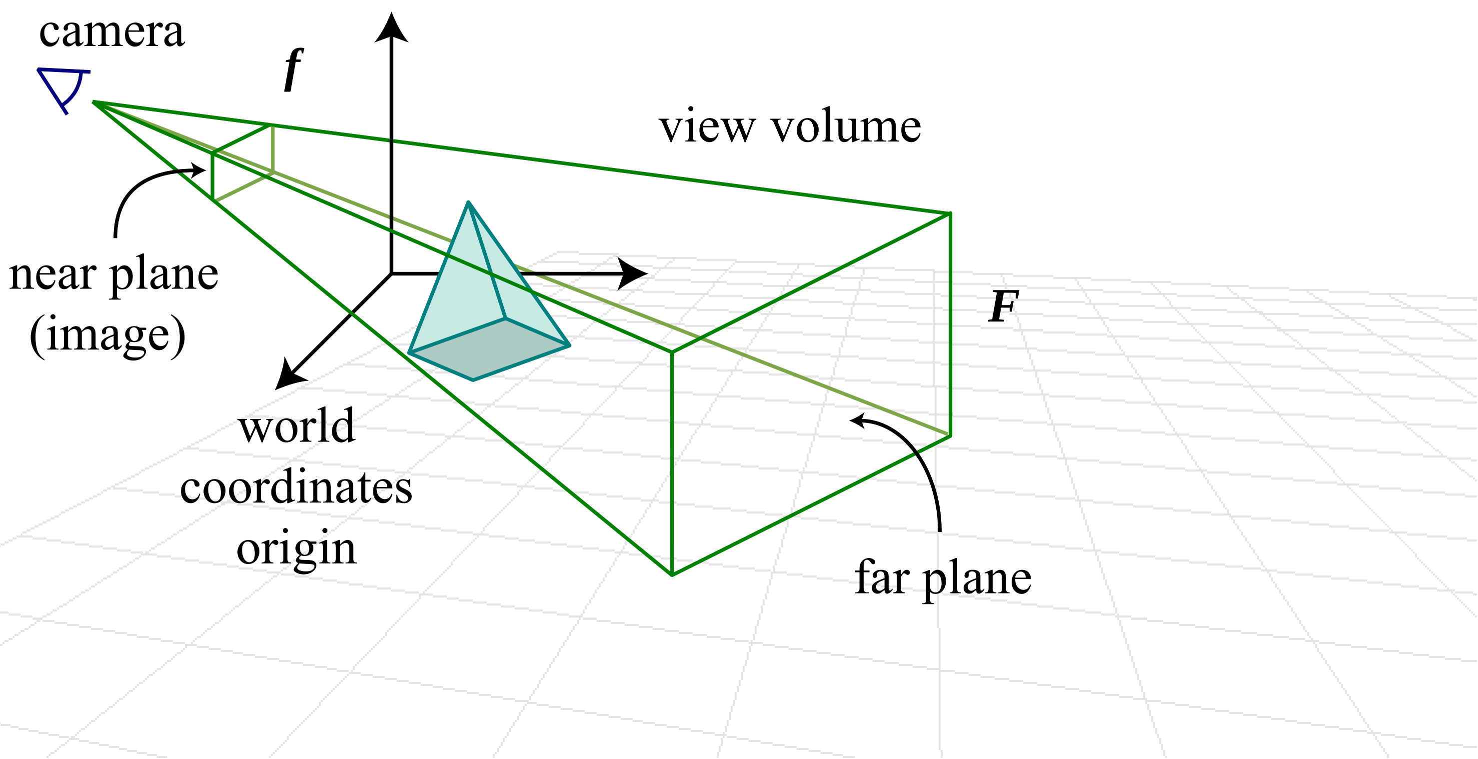
$$z^* = \frac{f}{p_z^c}\left(a p_z^c + b\right)$$



Pseudodepth

- How do we pick $\boldsymbol{a}$ and $\boldsymbol{b}$?

$$z^* = \frac{f}{p_z^c}\left(a p_z^c + b\right) \qquad z^* = \begin{cases} -1 & \text{when } p_z^c = f \\ 1 & \text{when } p_z^c = F \end{cases}$$



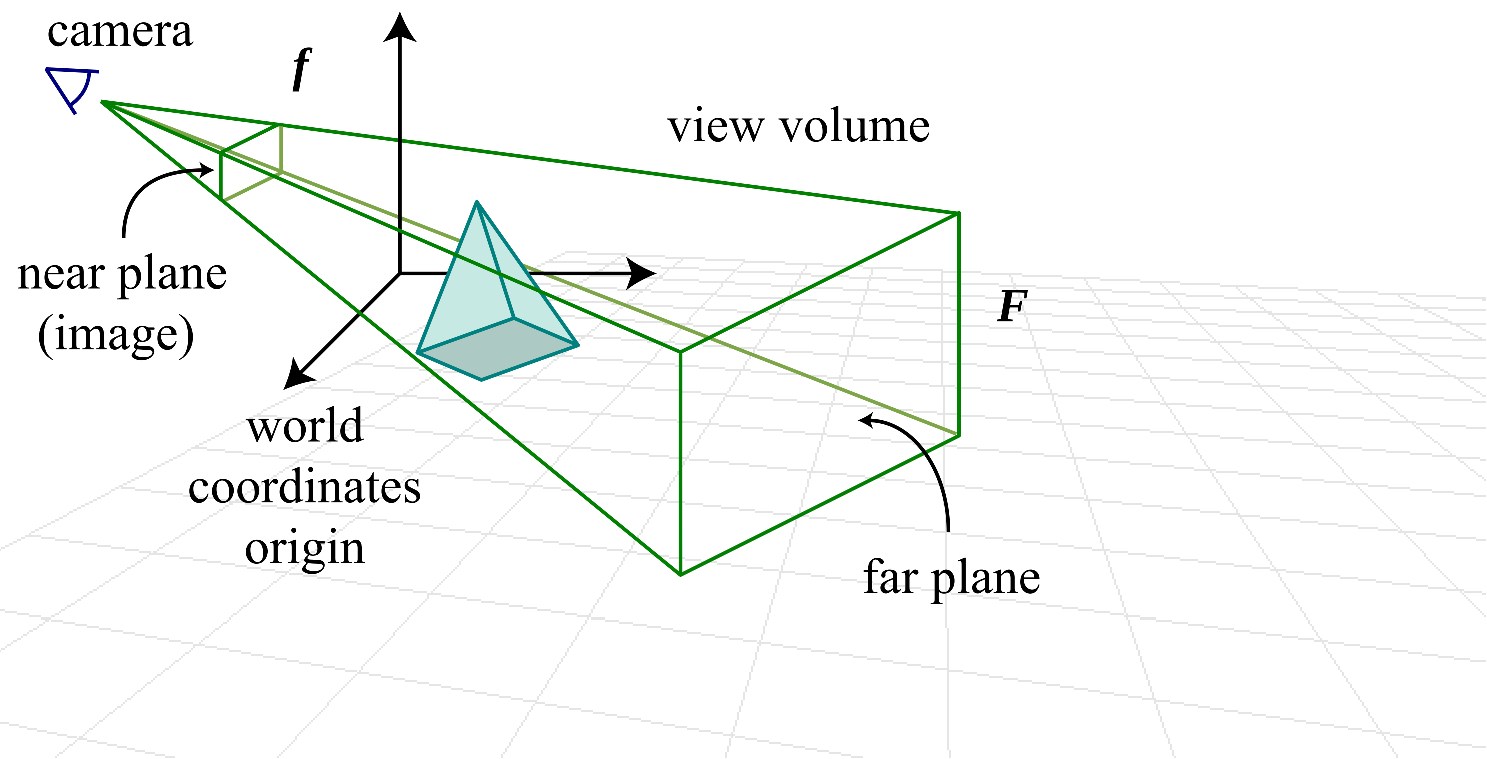
Pseudodepth

- How do we pick ***a*** and ***b***?

$$\boldsymbol{z}^* = \frac{\boldsymbol{f}}{\boldsymbol{p}_z^c}\left(\boldsymbol{a}\boldsymbol{p}_z^c + \boldsymbol{b}\right)$$

$$-1 = af + b$$

$$1 = af + b\frac{f}{F}$$



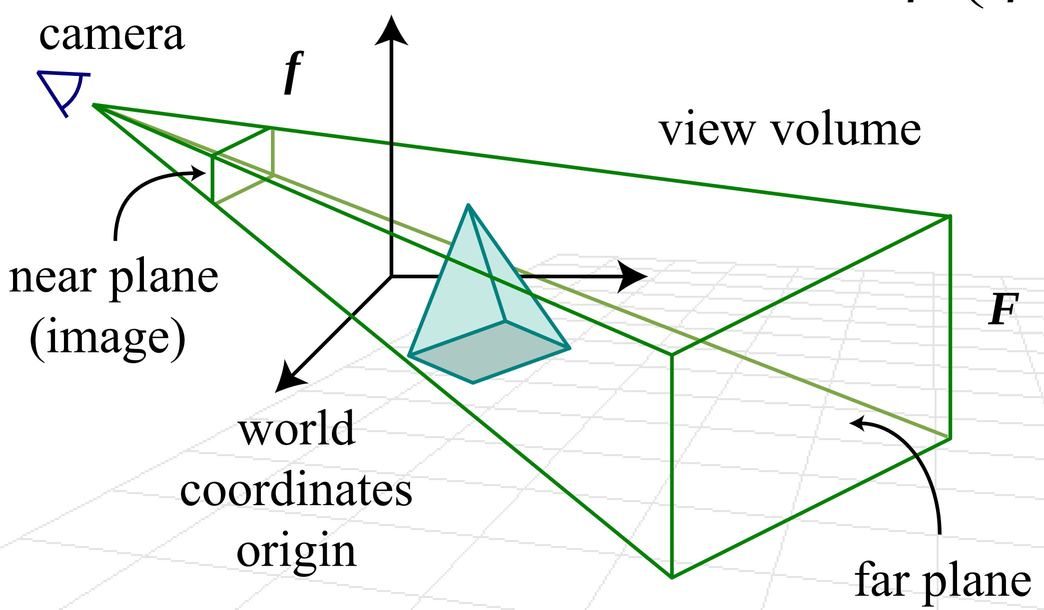
Pseudodepth

- How do we pick ***a*** and ***b***?

$$z^* = \frac{f}{p_z^c}\left(a p_z^c + b\right)$$

$$b = \frac{2F}{f - F}$$

$$a = -\frac{1}{f}\left(\frac{f + F}{f - F}\right)$$



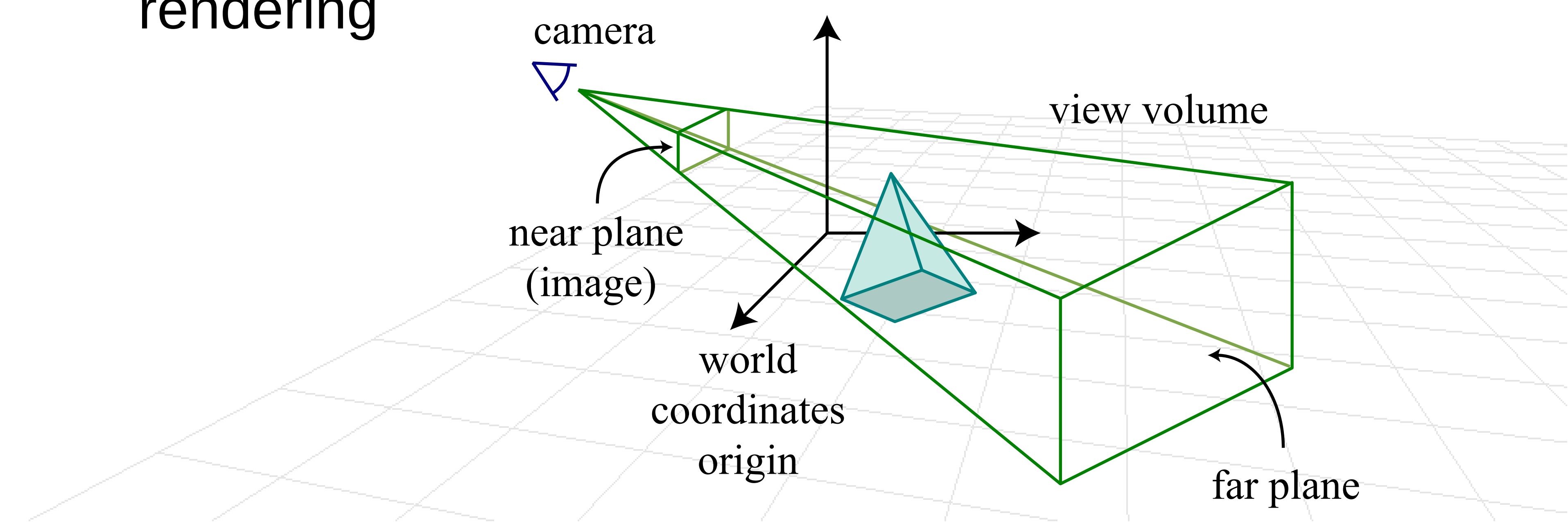
Pseudodepth

- Standard homogeneous perspective with pseudodepth

$$M_p = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \dfrac{2F}{f-F} & -\dfrac{1}{f}\left(\dfrac{f+F}{f-F}\right) \\ 0 & 0 & 1/f & 0 \end{bmatrix}$$

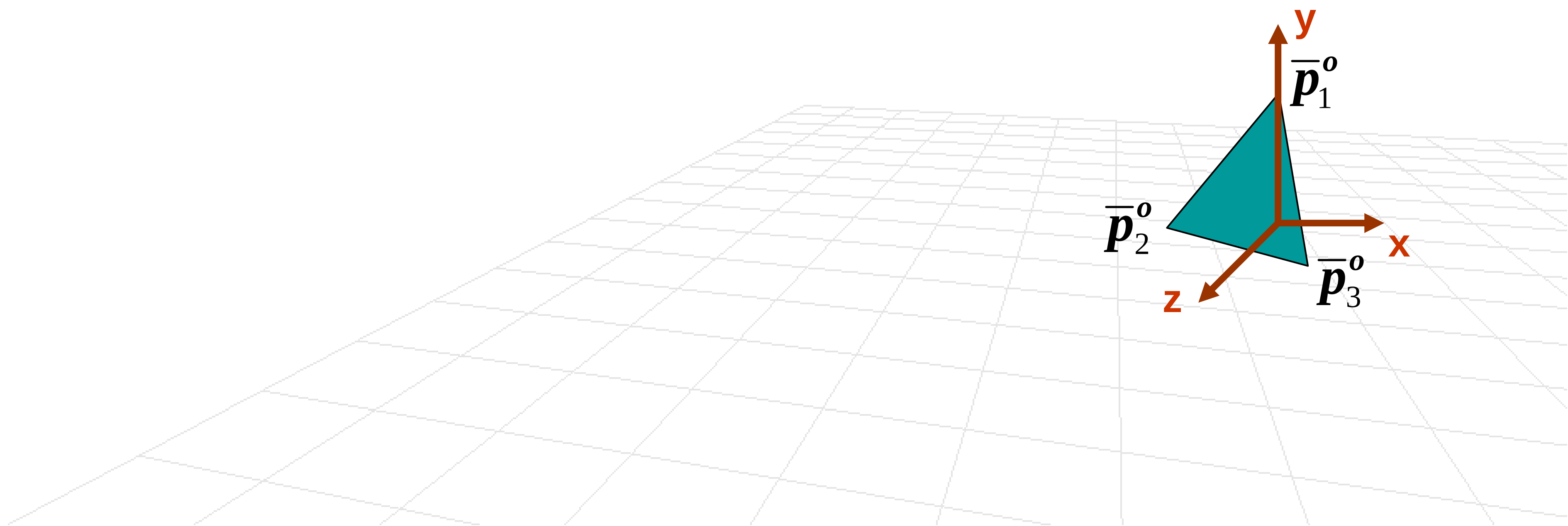
Near and Far Planes

- Anything closer than **near plane** is considered to be behind the camera and does not need to be rendered
- Anything further away from the camera than **far plane** is too far to be visible, so it is not rendered
- **Practical issue:** far plane too far away will lead to imprecision in the computed pseudodeph and hence rendering



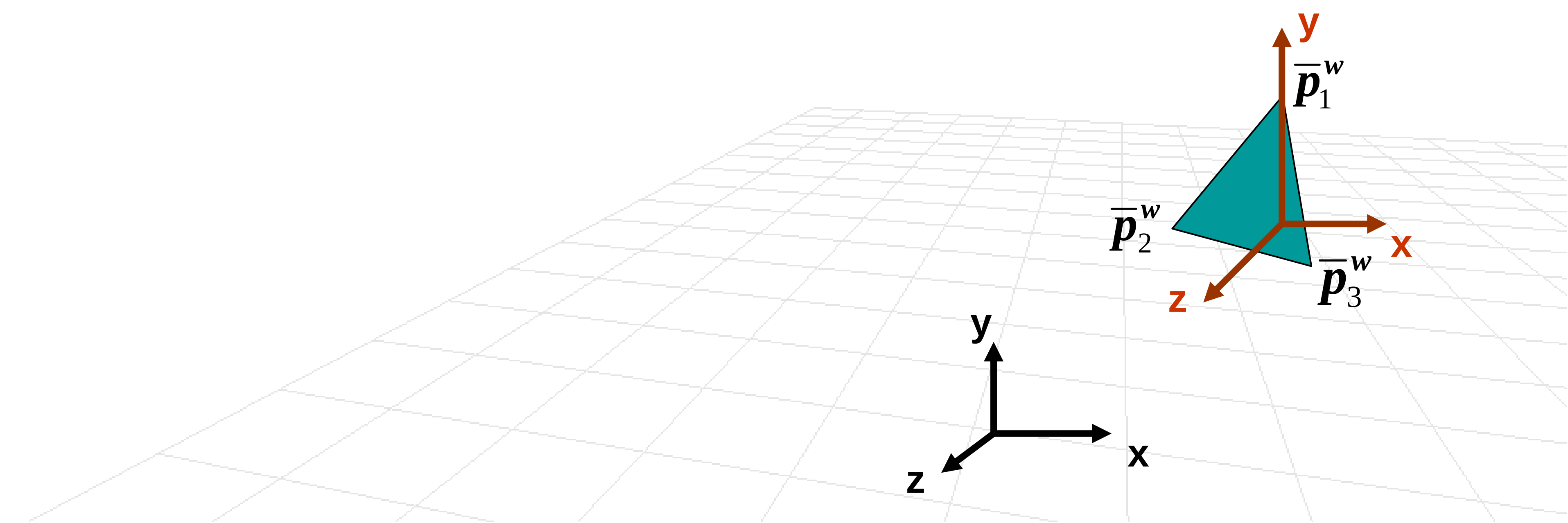
Projecting Triangle

- Lets review steps in the rendering hierarchy
  - Triangle is given in the object-based coordinate frame as three vertices



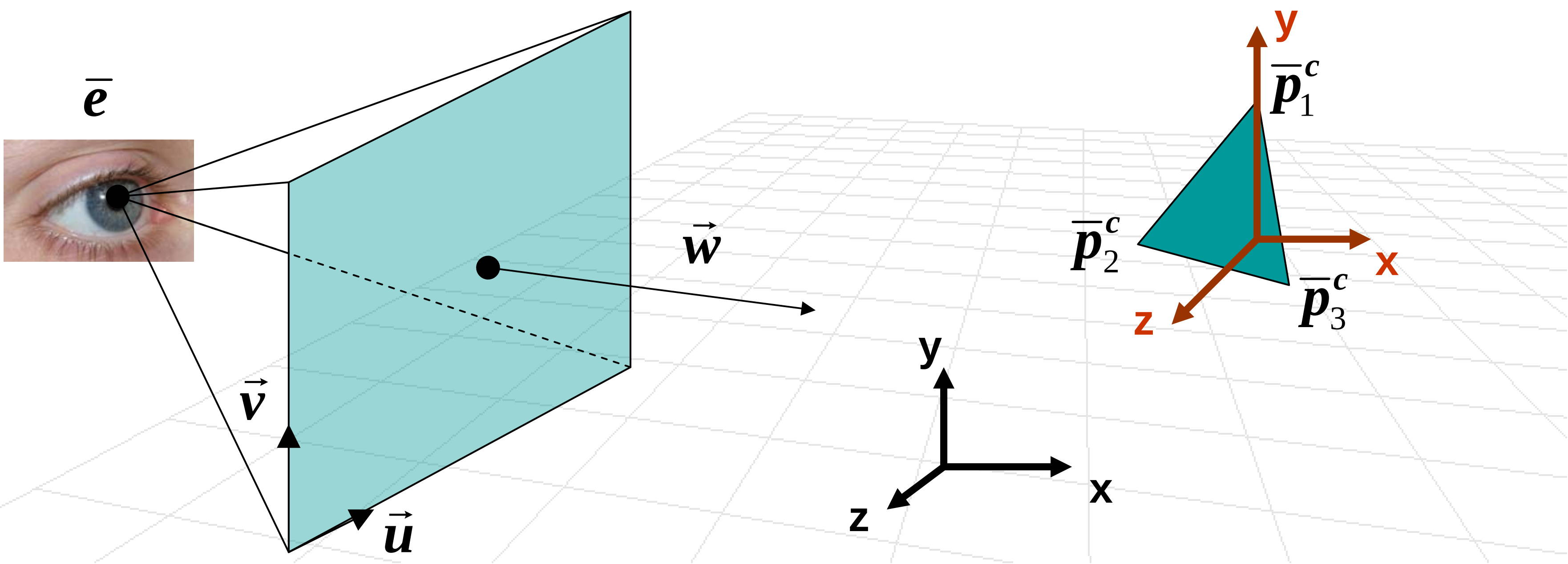
Projecting Triangle

- Lets review steps in the rendering hierarchy
  - Triangle is given in the object-based coordinate frame as three vertices
  - Transform to world coordinated $\overline{\boldsymbol{p}}_i^w = \boldsymbol{M}_{ow}\overline{\boldsymbol{p}}_i^o$



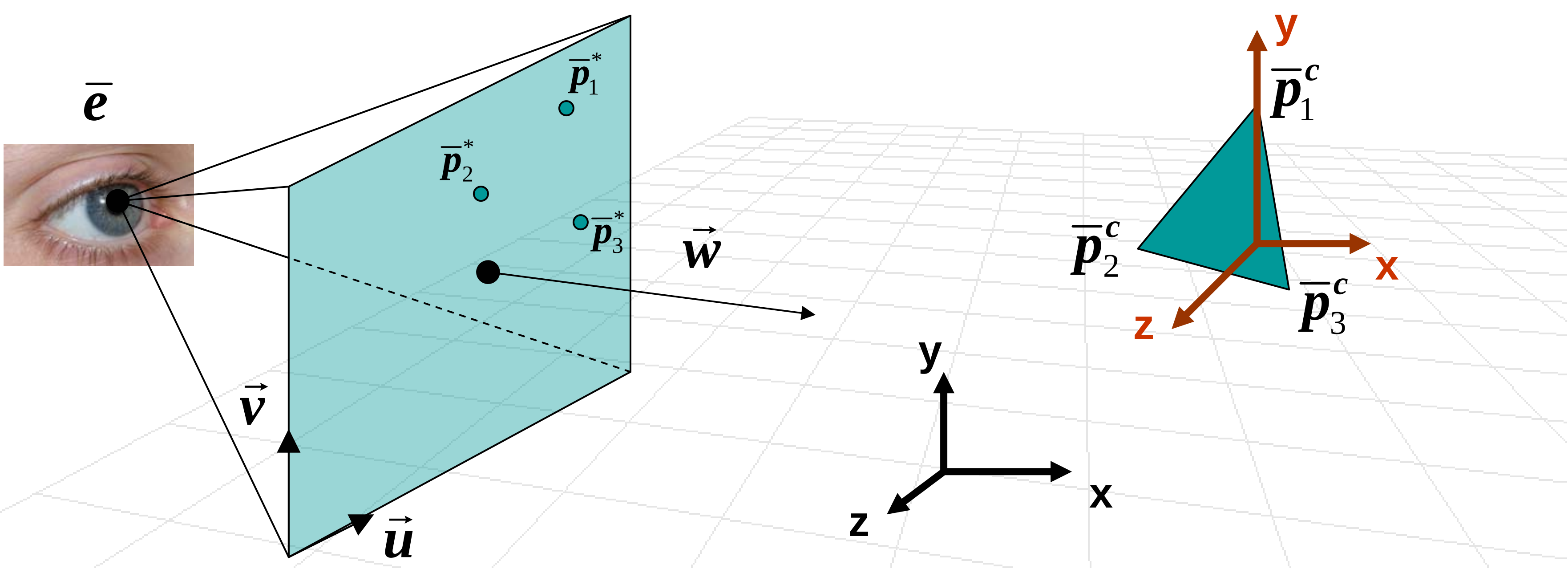
Projecting Triangle

- Lets review steps in the rendering hierarchy
  - Triangle is given in the object-based coordinate frame as three vertices
  - Transform to world coordinated $\overline{p}_i^w = M_{ow}\overline{p}_i^o$
  - Transform from world to camera coordinates $\overline{p}_i^c = M_{wc}\overline{p}_i^w$



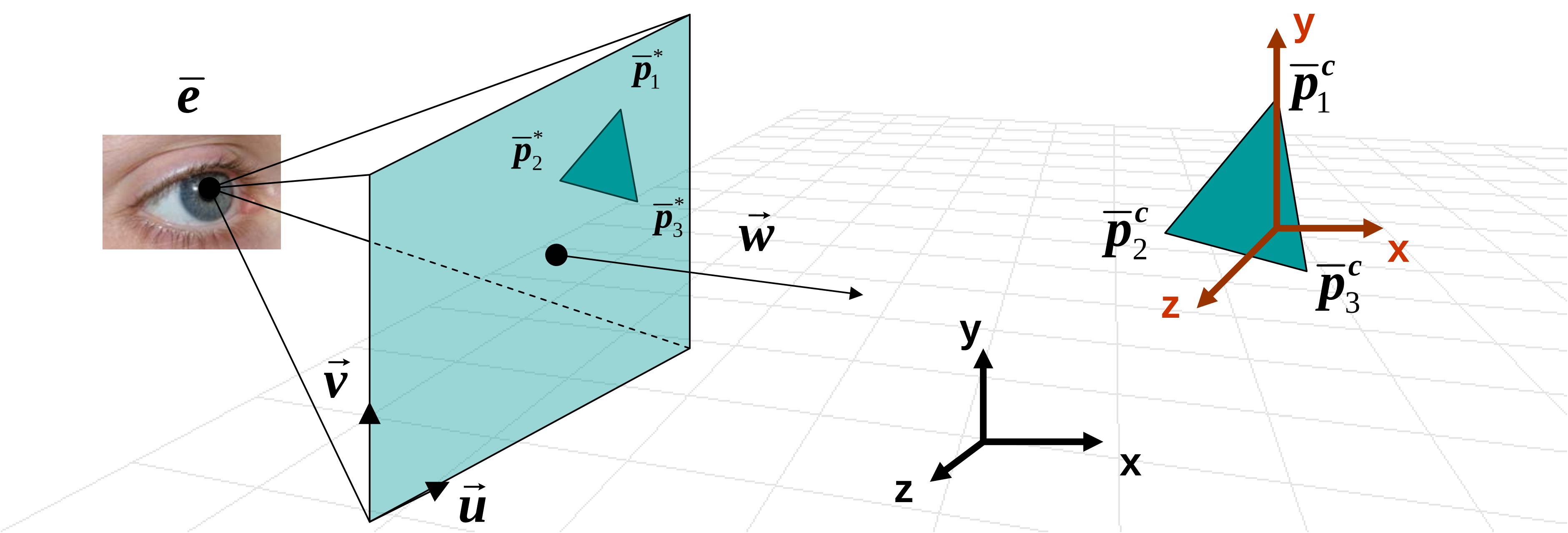
Projecting Triangle

- Lets review steps in the rendering hierarchy
  - Triangle is given in the object-based coordinate frame as three vertices
  - Transform to world coordinated $\overline{\boldsymbol{p}}_i^w = \boldsymbol{M}_{ow}\overline{\boldsymbol{p}}_i^o$
  - Transform from world to camera coordinates $\overline{\boldsymbol{p}}_i^c = \boldsymbol{M}_{wc}\overline{\boldsymbol{p}}_i^w$
  - Apply homogeneous perspective $\overline{\boldsymbol{p}}_i^* = \boldsymbol{M}_p\overline{\boldsymbol{p}}_i^c$
    - Divide by last component



Projecting Triangle

- Lets review steps in the rendering hierarchy
  - Triangle is given in the object-based coordinate frame as three vertices
  - Transform to world coordinated $\bar{p}_i^w = M_{ow}\bar{p}_i^o$
  - Transform from world to camera coordinates $\bar{p}_i^c = M_{wc}\bar{p}_i^w$
  - Apply homogeneous perspective $\bar{p}_i^* = M_p\bar{p}_i^c$
    - Divide by last component



Visibility

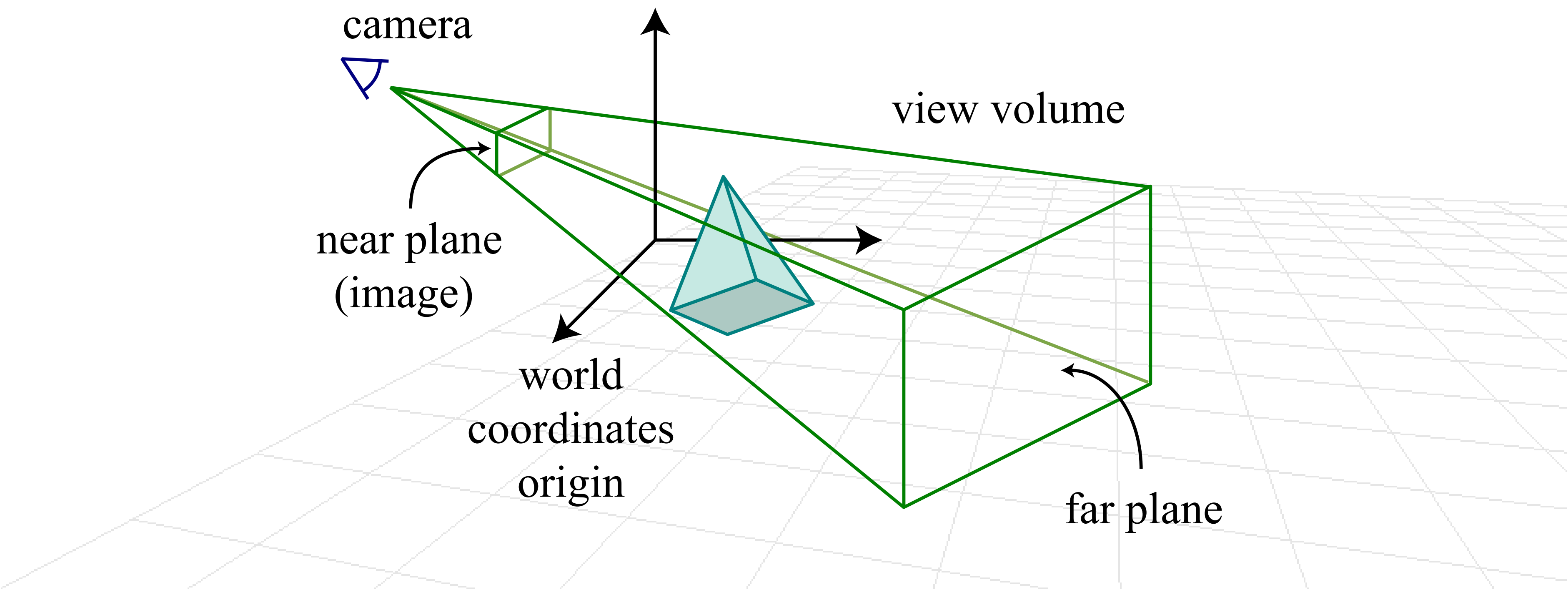
Computer Graphics, CSCD18

Fall 2007

Instructor: Leonid Sigal

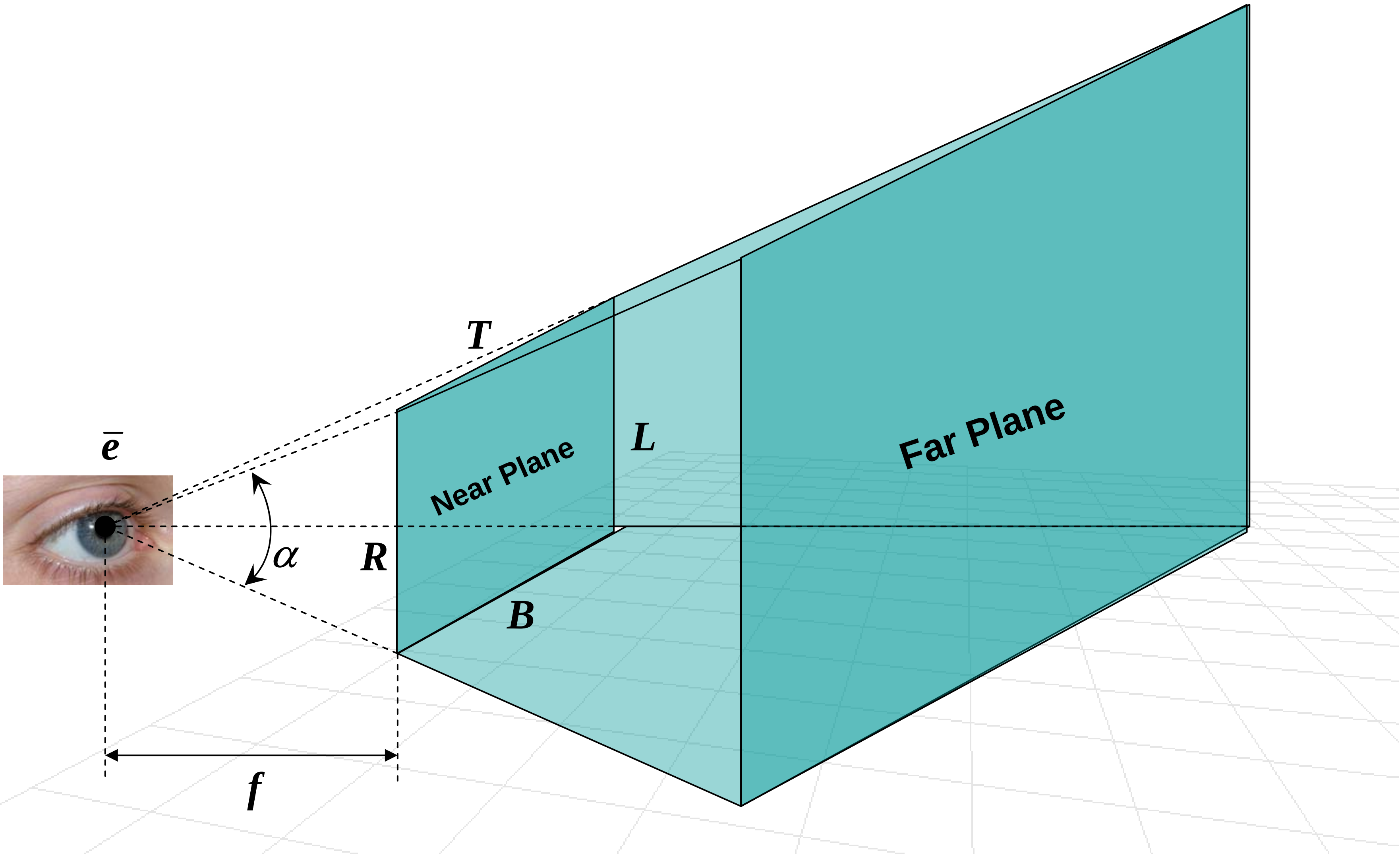
Clipping

- **Idea:** Remove points and parts of objects outside view volume
- Sounds simple, but consider if we have an object on a boundary

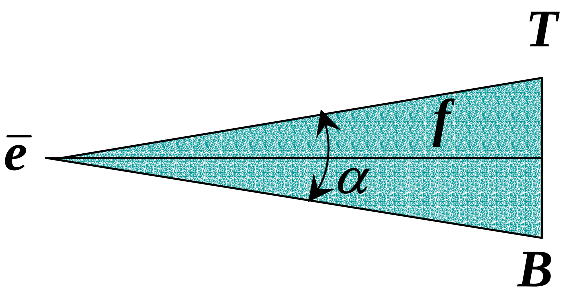


View Volume

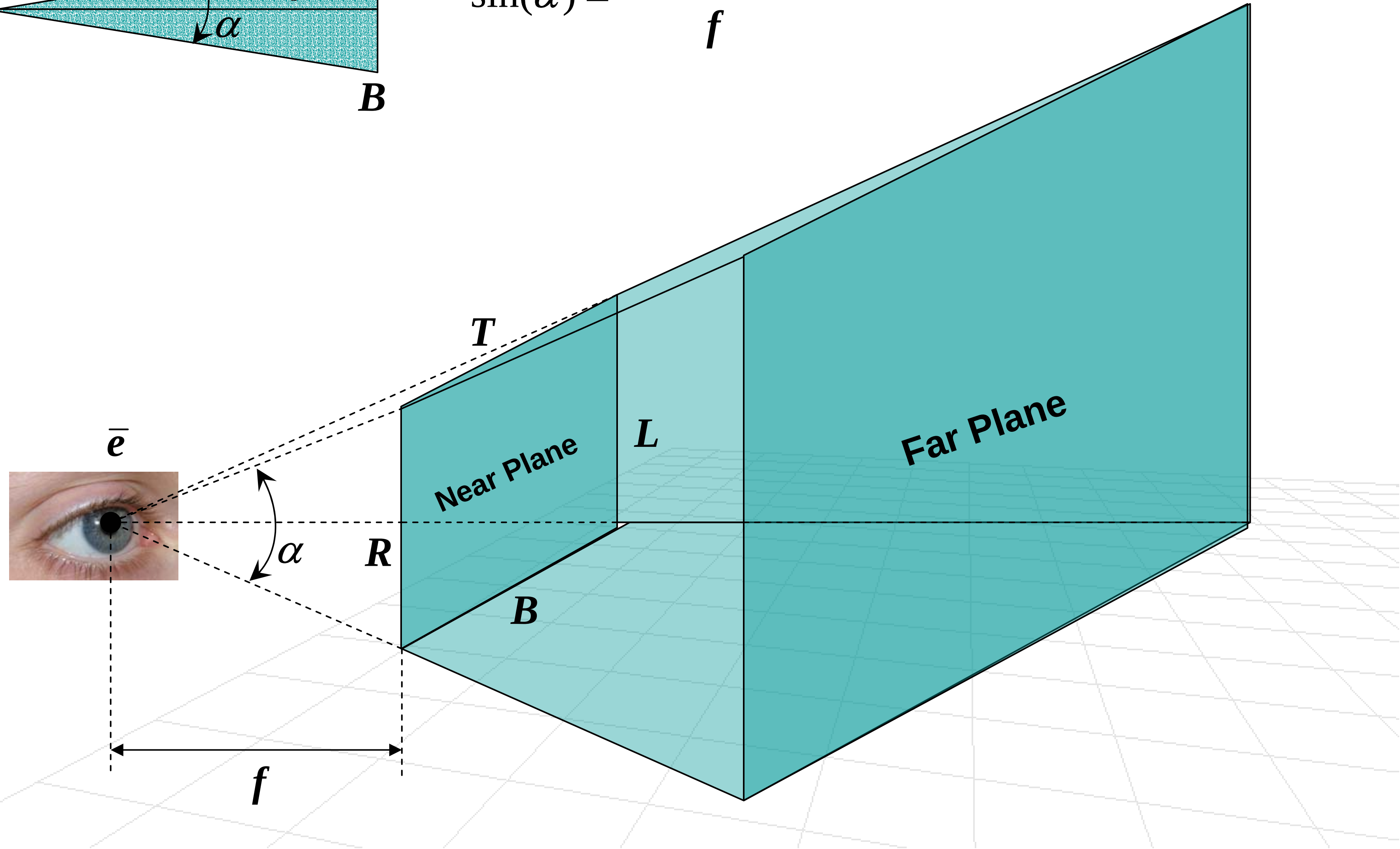
- Consider what we can actually see



Side note: Field of View



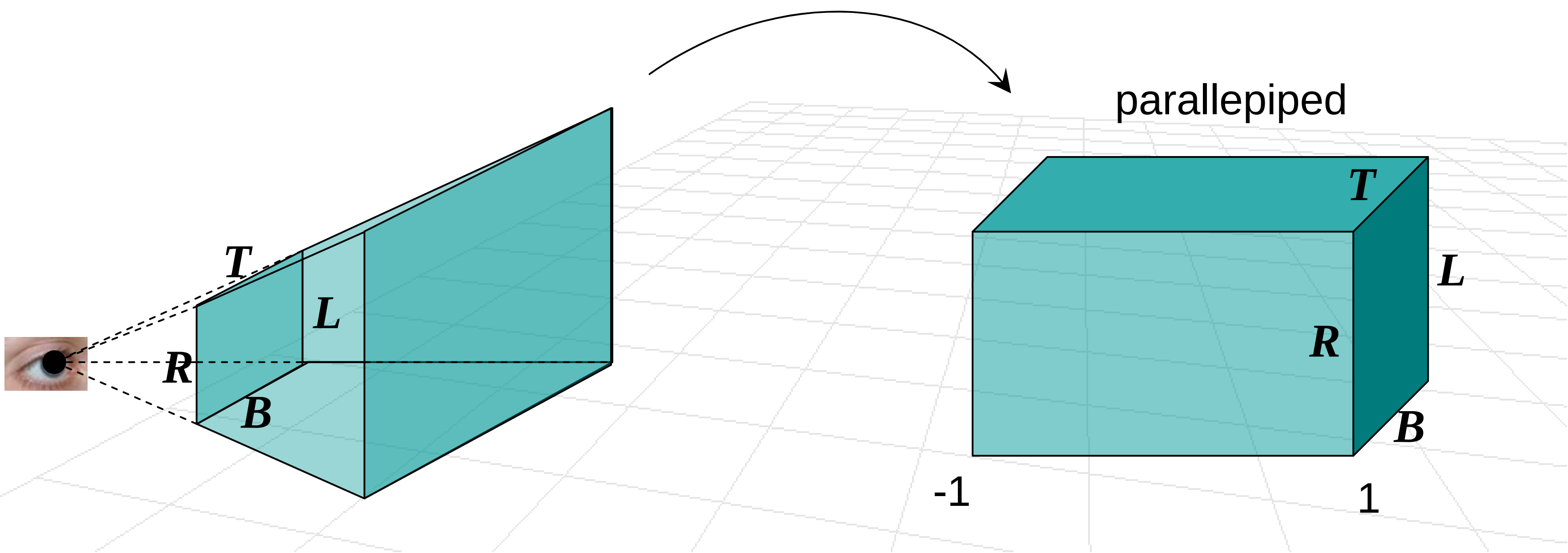
$$\sin(\alpha) = \frac{1/2(\boldsymbol{T} - \boldsymbol{B})}{\boldsymbol{f}}$$



View Volume

- What does homogeneous perspective projection do to our view volume?

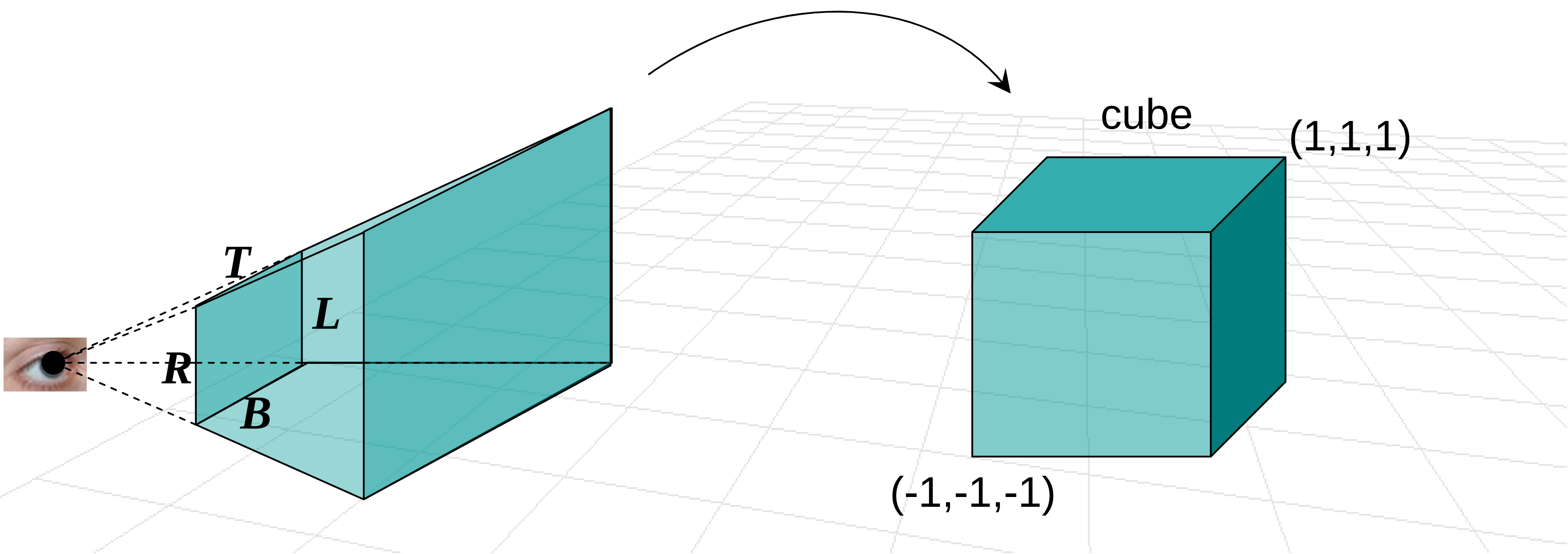
$$M_p = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \dfrac{2F}{f-F} & -\dfrac{1}{f}\left(\dfrac{f+F}{f-F}\right) \\ 0 & 0 & 1/f & 0 \end{bmatrix}$$



Canonical View Volume

- Can we alter homogeneous perspective projection to help us clip?

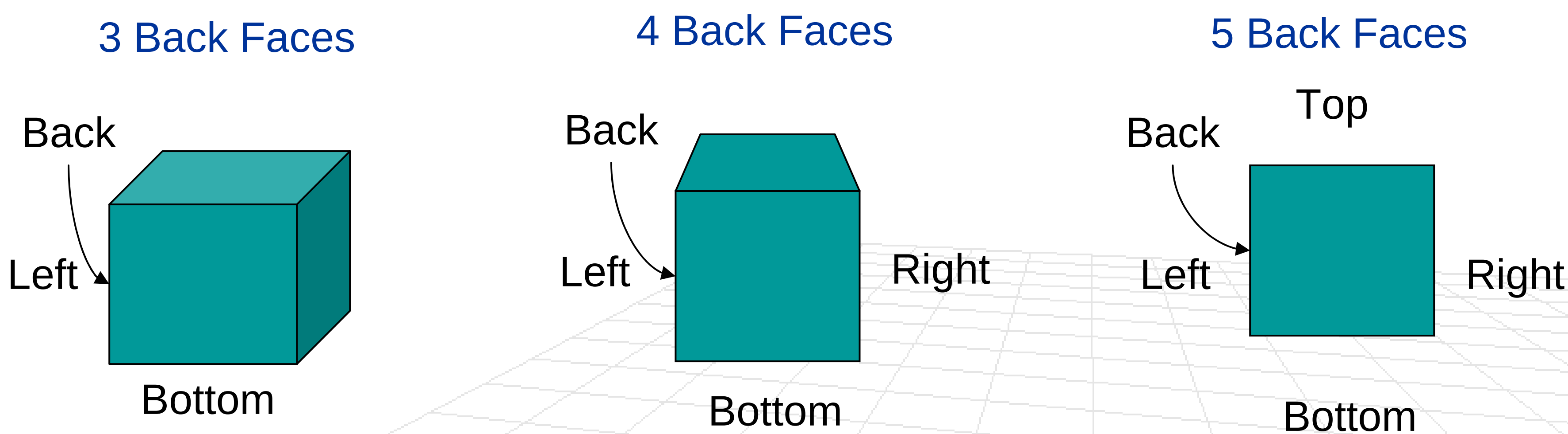
$$M_p = \begin{bmatrix} \frac{2}{R-L} & 0 & \frac{R+L}{R-L} & 0 \\ 0 & \frac{2}{T-B} & \frac{T+B}{T-B} & 0 \\ 0 & 0 & \frac{2F}{f-F} & -\frac{1}{f}\left(\frac{f+F}{f-F}\right) \\ 0 & 0 & 1/f & 0 \end{bmatrix}$$



Back-face Removal

- **Idea:** Remove surface patches that point away from the camera (like backside of the object as it viewed from the front)

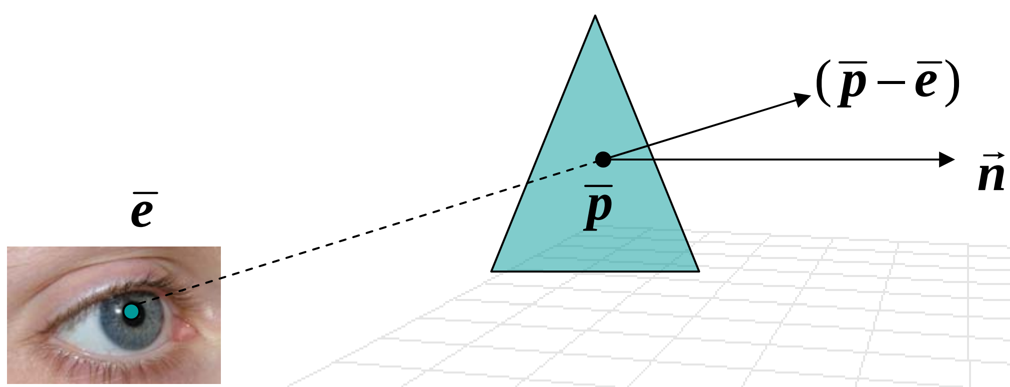
- Consider a cube



- We only need to render at most half of the sides depending on the view

Back-face Removal

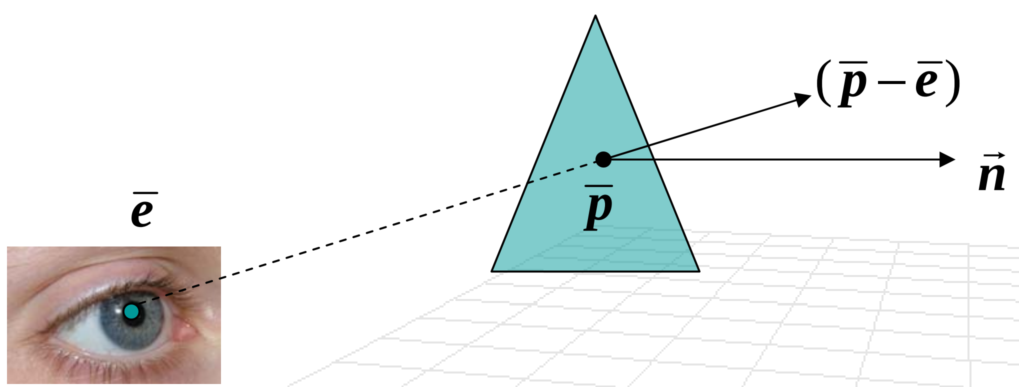
- How do we know if the patch (triangle) points away from the camera?
- Consider normal of the triangle



- If $(\bar{p}-\bar{e})\cdot\vec{n} > 0$ then triangle is part of the back-face and needs to be removed
- If $(\bar{p}-\bar{e})\cdot\vec{n} < 0$ then triangle **may** be visible

Back-face Removal

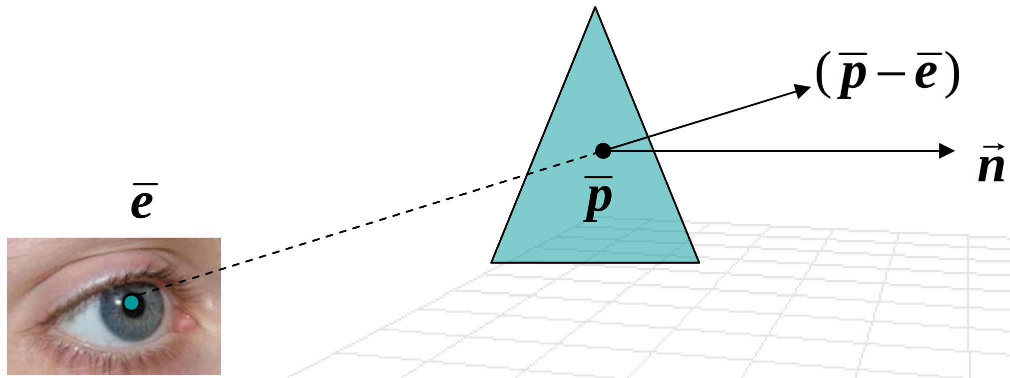
- Does it matter which point we consider on the patch?



- If $(\bar{p} - \bar{e}) \cdot \vec{n} > 0$ then triangle is part of the back-face and needs to be removed
- If $(\bar{p} - \bar{e}) \cdot \vec{n} < 0$ then triangle **may** be visible

Back-face Removal

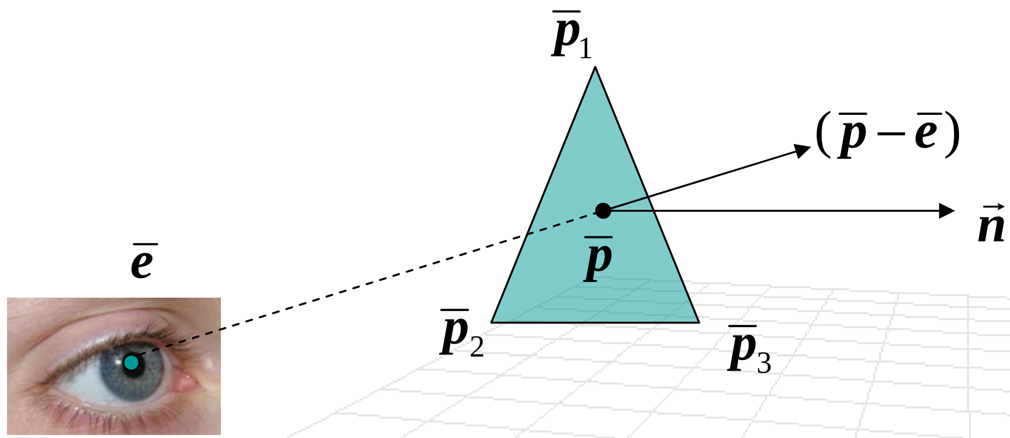
- Does it matter which point we consider on the patch?
  - Not if this is a **planar** patch



- If $(\bar{p}-\bar{e})\cdot\vec{n}>0$ then triangle is part of the back-face and needs to be removed
- If $(\bar{p}-\bar{e})\cdot\vec{n}<0$ then triangle **may** be visible

Back-face Removal

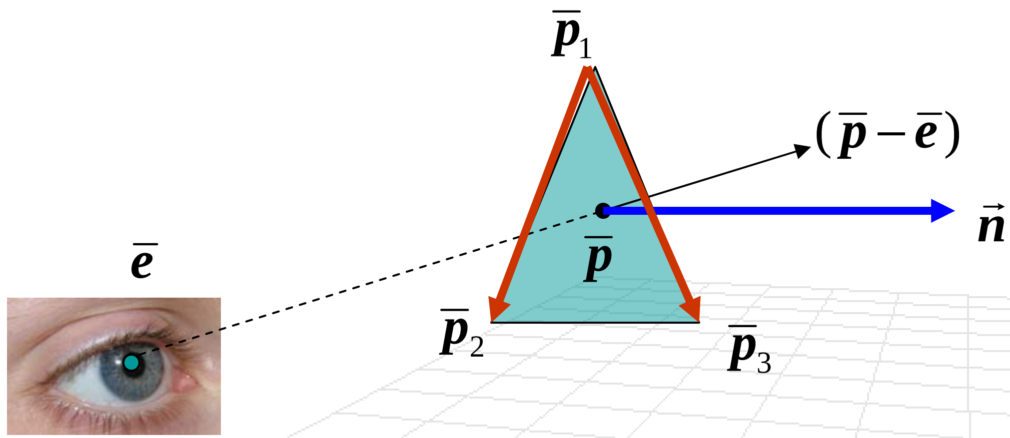
- Does it matter which point we consider on the patch?
  - Not if this is a **planar** patch
- How do we compute $\vec{n}$
  - If $\bar{p}_1, \bar{p}_2, \bar{p}_3$ are patch vertices in CCW order



- If $(\bar{p} - \bar{e}) \cdot \vec{n} > 0$ then triangle is part of the back-face and needs to be removed
- If $(\bar{p} - \bar{e}) \cdot \vec{n} < 0$ then triangle **may** be visible

Back-face Removal

- Does it matter which point we consider on the patch?
  - Not if this is a **planar** patch
- How do we compute $\vec{n} = \frac{(\bar{p}_2 - \bar{p}_1)\times(\bar{p}_3 - \bar{p}_1)}{\|(\bar{p}_2 - \bar{p}_1)\times(\bar{p}_3 - \bar{p}_1)\|}$



- If $(\bar{p}-\bar{e})\cdot\vec{n} > 0$ then triangle is part of the back-face and needs to be removed
- If $(\bar{p}-\bar{e})\cdot\vec{n} < 0$ then triangle **may** be visible

Z-Buffer (a.k.a Depth Buffer)

- We have a **frame-buffer** (this is where an image that we see on the screen is stored)
- We also have a **z-buffer** that keeps track of the z^* coordinate for every pixel in the frame-buffer

- To draw point in the world with color c that projects to $(x^*, y^* z^*)$ we can execute the following algorithm

```
if z* < z-buffer(x*, y*) then
        frame-buffer(x*, y*) = c
        z-buffer(x*, y*) = z*
end
```

Z-Buffer (a.k.a Depth Buffer)

- We need to initialize the z-buffer with some value. What is the good value to initialize with?
  - If we are using canonical view volume then 1 would work

- To draw point in the world with color c that projects to $(x^*, y^* z^*)$ we can execute the following algorithm

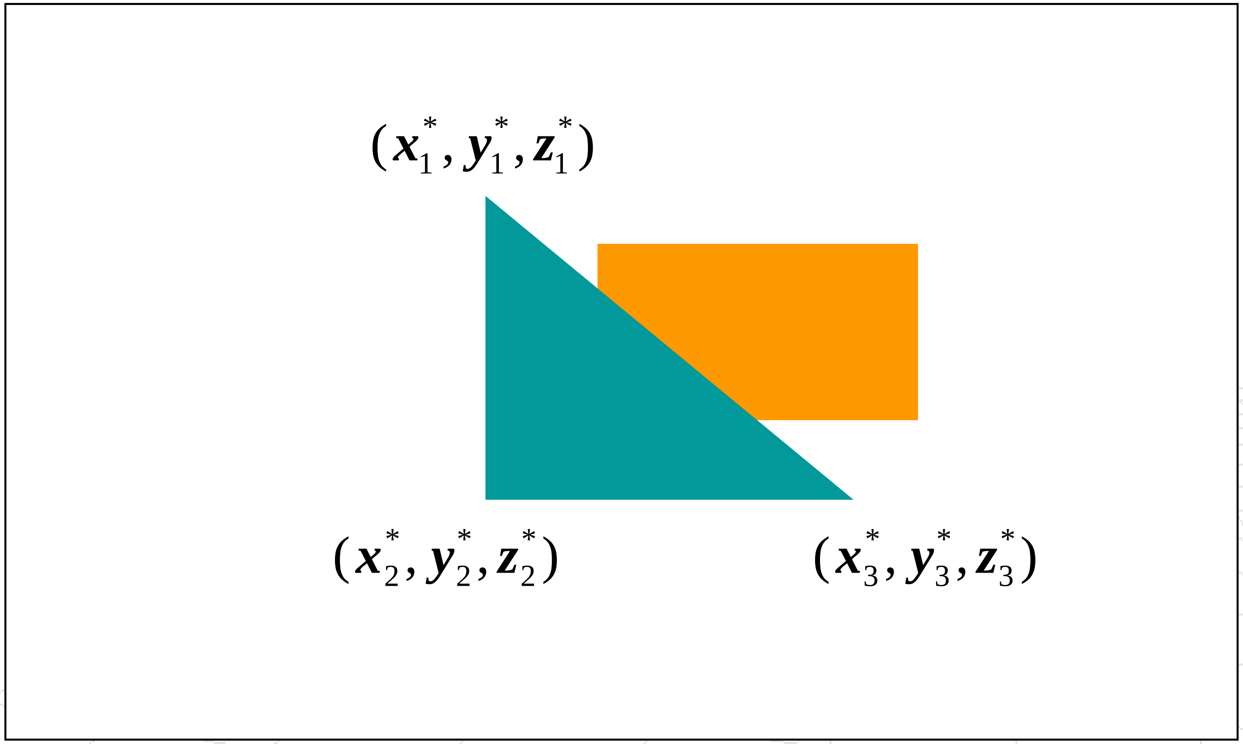
```
if z* < z-buffer(x*, y*) then
        frame-buffer(x*, y*) = c
        z-buffer(x*, y*) = z*
end
```

Z-Buffer (a.k.a Depth Buffer)

- Advantages of Z-buffering
  - Simple and accurate
  - Independent of the order the polygons are drawn
- Disadvantages of Z-buffering
  - Memory for a Z-buffer (small consideration)
  - Wasted computation in drawing distant points first (this potentially can be a large drawback)

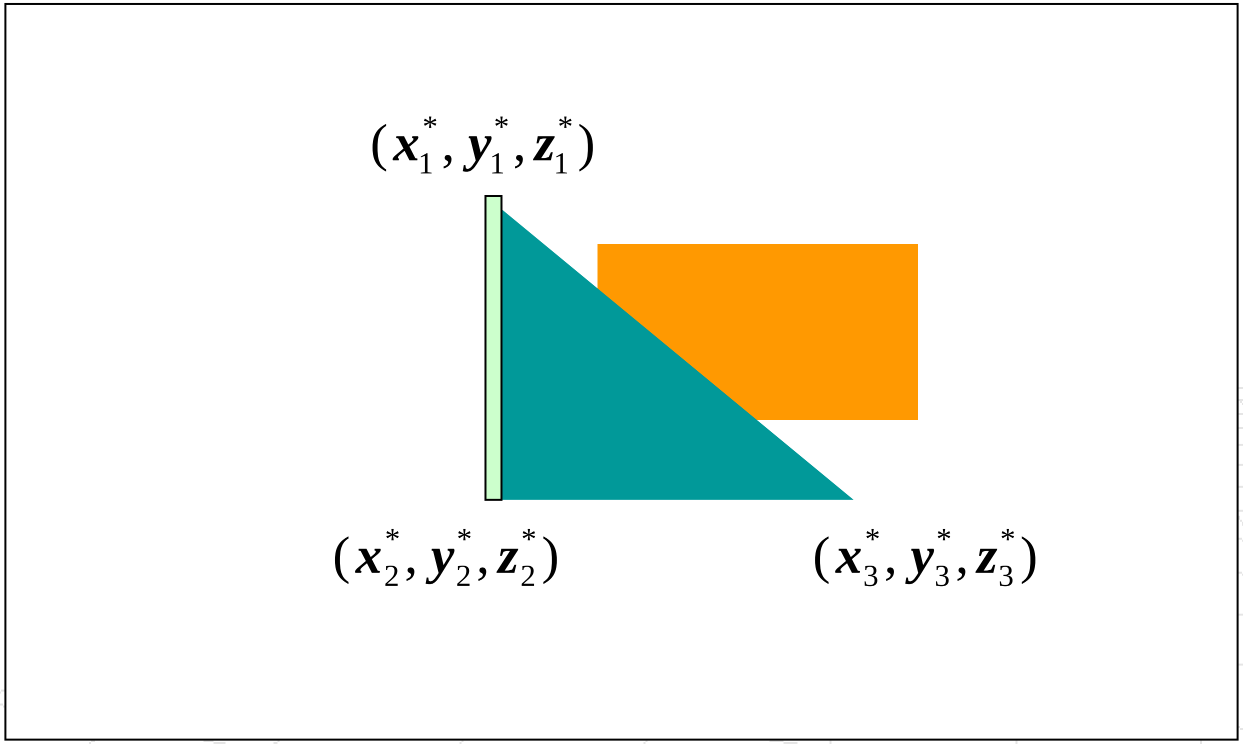
Z-Buffer (a.k.a Depth Buffer)

- We represent a patch using vertices
- How do we get a pseudodeph and proper rendering everywhere else?



Z-Buffer (a.k.a Depth Buffer)

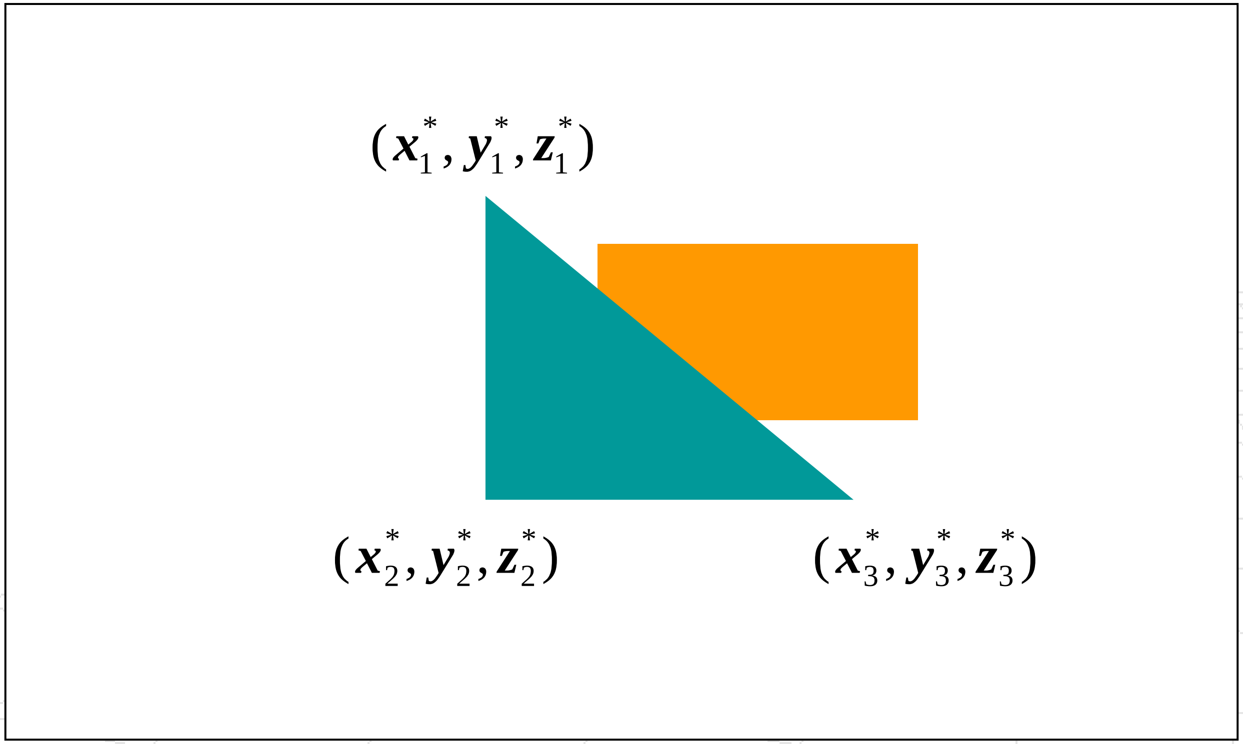
- We represent a patch using vertices
- How do we get a pseudodeph and proper rendering everywhere else?



Linearly interpolate z^* along a scan line

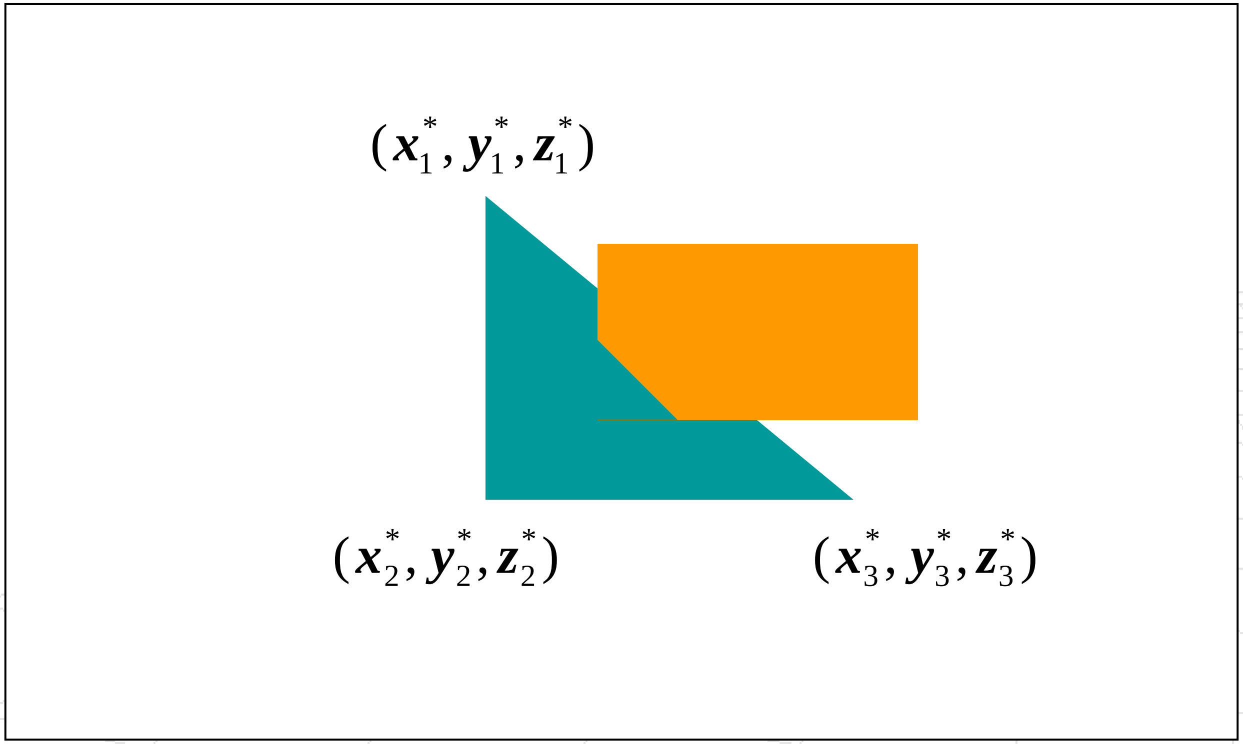
Painter's Algorithm

- **Idea:** Order the patches and draw them in the order of depth (with most distant patches first)
- This is an alternative to Z-buffering



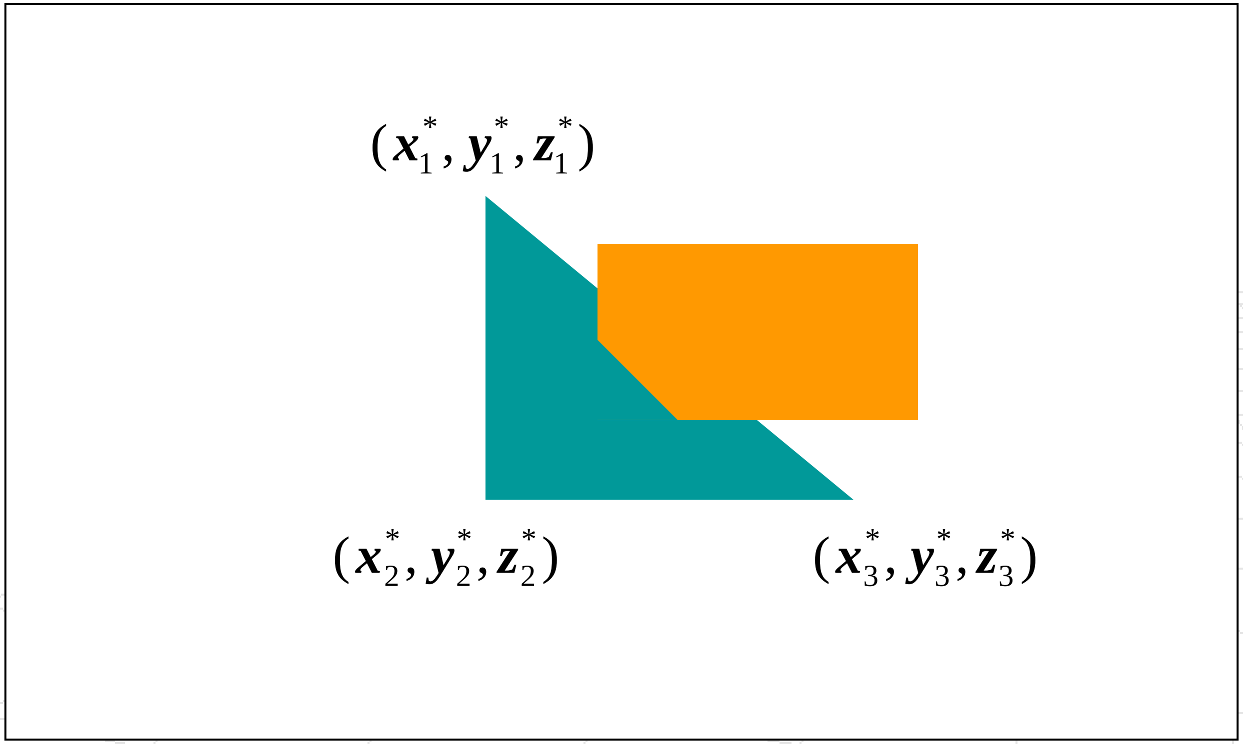
Painter's Algorithm

- How do we deal with intersecting patches?
  - Break patches into smaller patches



BSP Trees

- **Binary space partition tree** (BSP tree) is an algorithm for making back-to-front ordering of polygons efficient and to break polygons to avoid intersections



BSP Tree

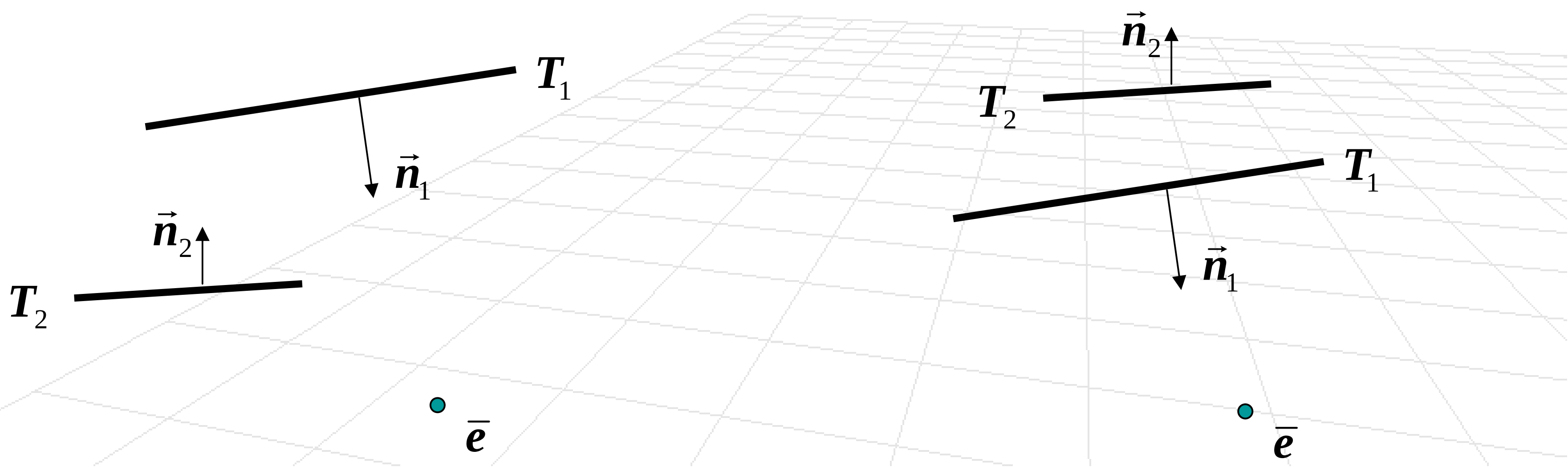
- If $\bar{e}$ and T_2 on the same side of T_1 (left) then draw T_1 first then T_2
- If $\bar{e}$ and T_2 are on different sides of T_1 (right) then draw T_2 first then T_1
- How do we know if points are on the same side?

$$f_1(\bar{x}) = (\bar{x} - \bar{p}_1) \cdot \vec{n}_1$$

$$f_1(\bar{x}) = 0 \quad \textit{on the plane}$$

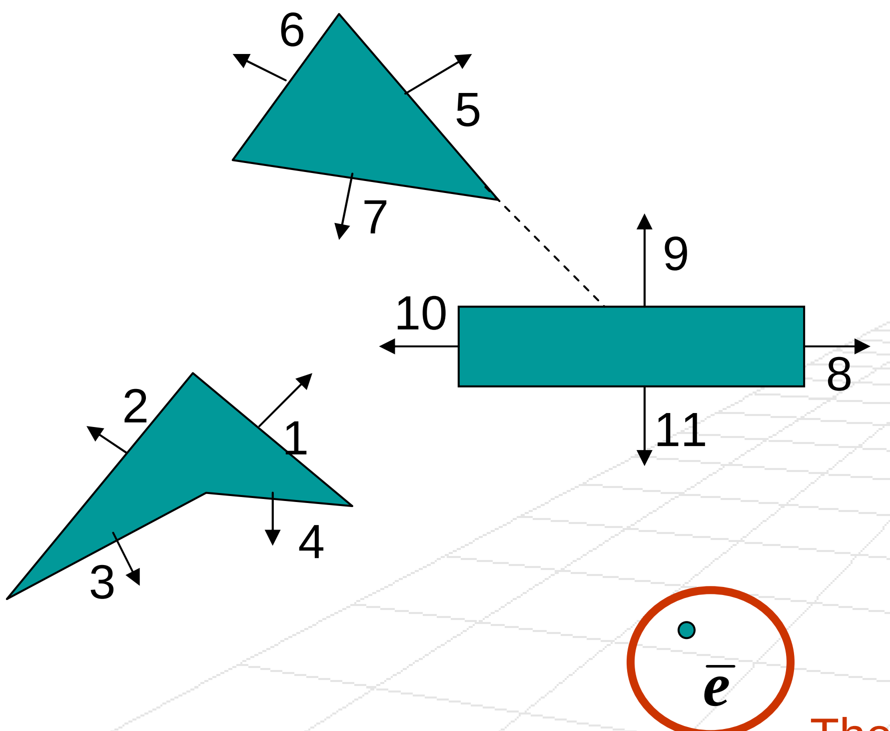
$$f_1(\bar{x}) > 0 \quad \text{"}\textit{outside}\text{"}$$

$$f_1(\bar{x}) < 0 \quad \text{"}\textit{inside}\text{"}$$



BSP Tree Example

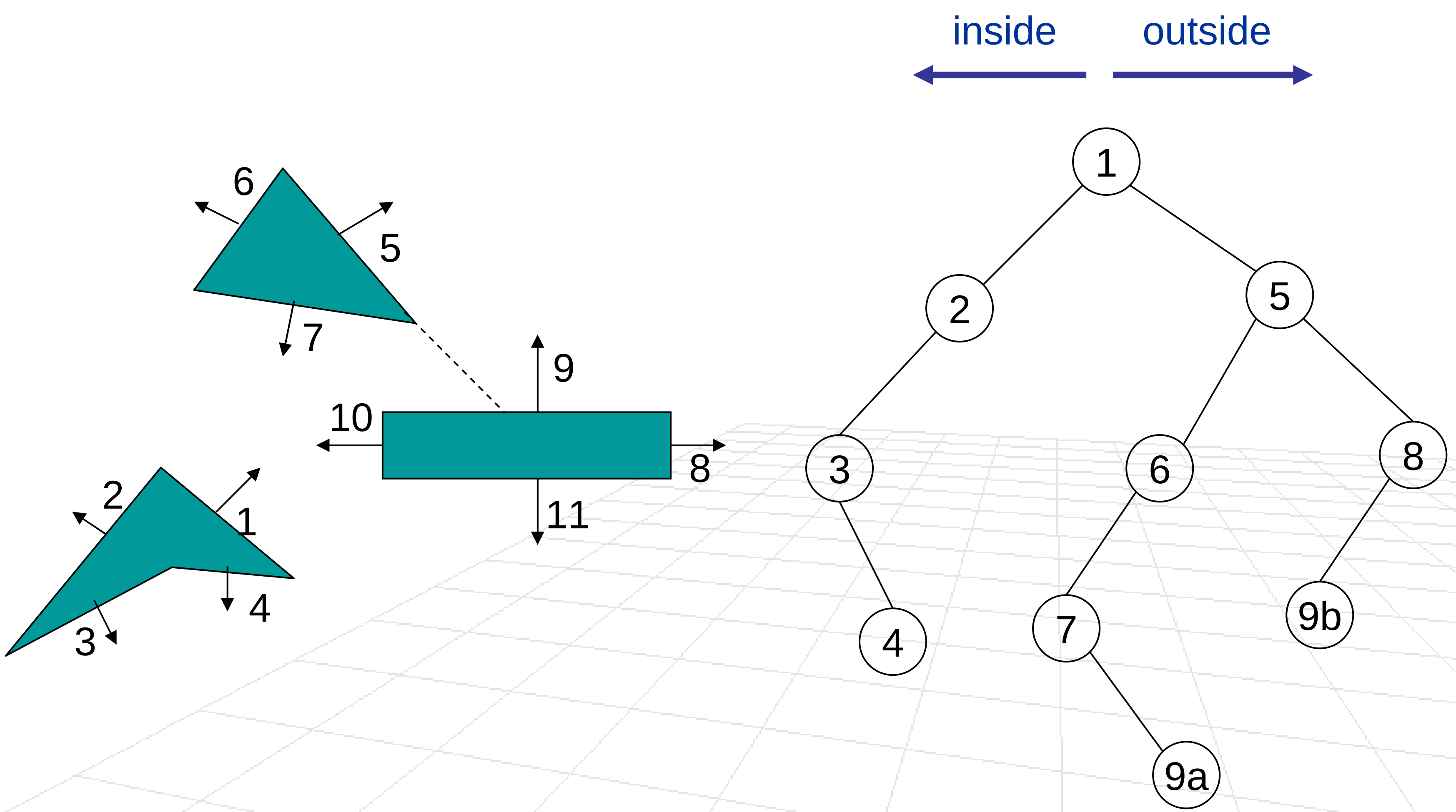
- Let's try building a BSP tree for this scene



The tree will be the same regardless of the camera placement

BSP Tree Example

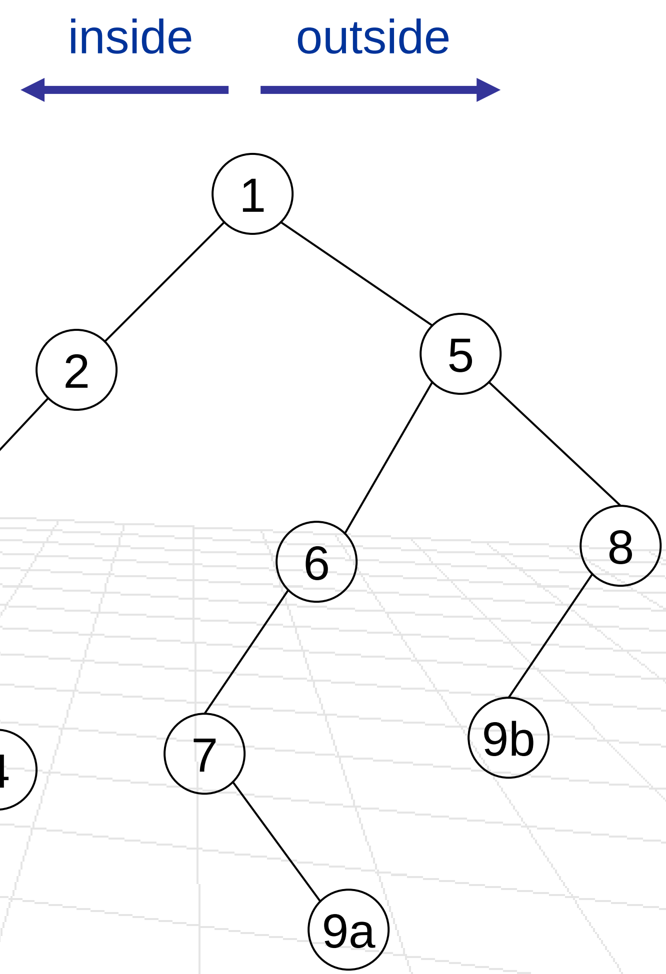
- Let's try building a BSP tree for this scene



BSP Tree Traversal

- Tree traversal algorithm

```
if eye in the outside half-space of the root
            Draw faces on inside sub-tree of the root
            Draw the root
            Draw faces is the outside of sub-tree of the root
      else
            Draw faces is the outside of sub-tree of the root
            Draw the root
            Draw faces on inside sub-tree of the root
      end
```



- Easy to modify to do back-face removal

BSP Tree

- Advantages
  - Can easily discard portions of the scene behind the camera
  - Artifacts of z-buffer quantization are not seen
  - Tree construction fixed for the static scenes
- Disadvantages
  - How can we handle dynamic scenes?