Announcements

- **Assignment 1**
  - programming (**due Friday**)
  - submission directories are fixed use (submit -N A1b cscd18f08 a1_solution.tgz)
  - theory will be returned (**Wednesday**)

- **Midterm**
  - Will cover **all of the materials so far including today's lecture**
    - Lecture notes, lecture slides, readings, assignment are all fair game
  - Practice midterms are on-line (no solutions will be given)

- **Tutorial this week**
  - Life of the polygon
  - A1 theory questions

- **Office Hours**
  - I will have office hours **today 1-2 pm**
  - Alex will have office hours later in the week
  - I will also have office hours on **Tuesday 4-5pm**

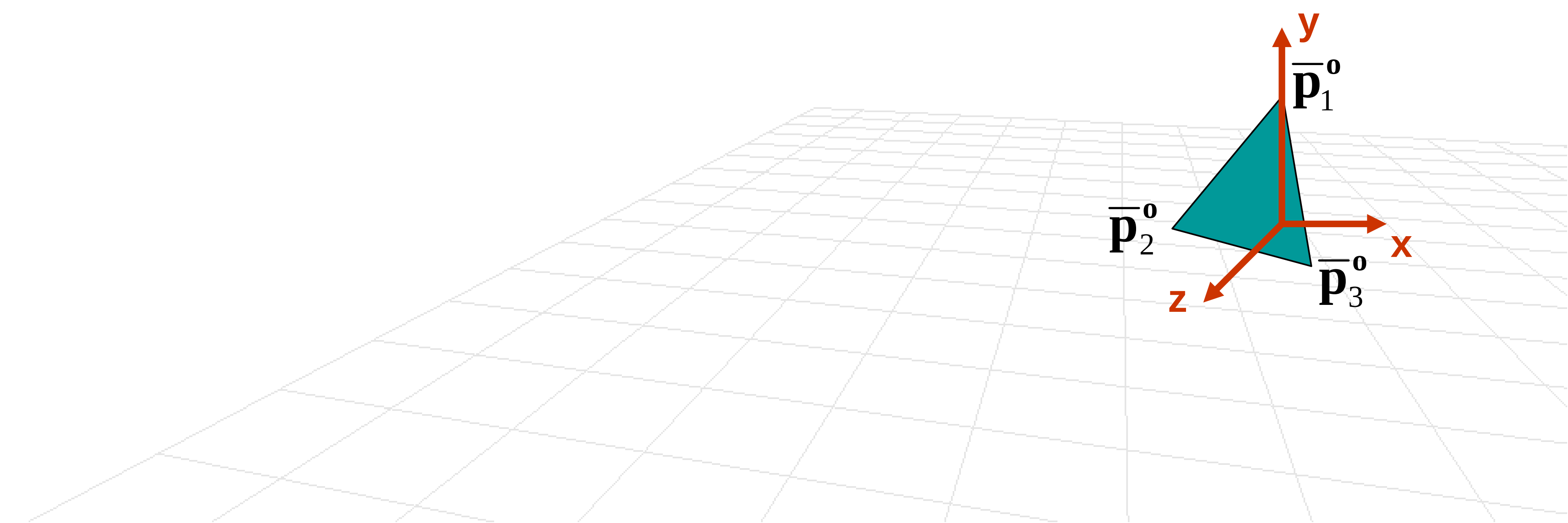
Last week's review …

- **Cameras (theory)**
  - Pinhole Camera
  - Thin Lens model
  - Virtual pinhole camera
  - Perspective and orthographic projections

- **Cameras (practice)**
  - Location of camera in space
  - Transformation of geometry from camera to world coordinate frame and (**vice versa**)
  - **Homogeneous Perspective Projection** (how do we represent perspective using a single 4x4 matrix)
  - Homogeneous Prospective Projection with **Pseudodepth**

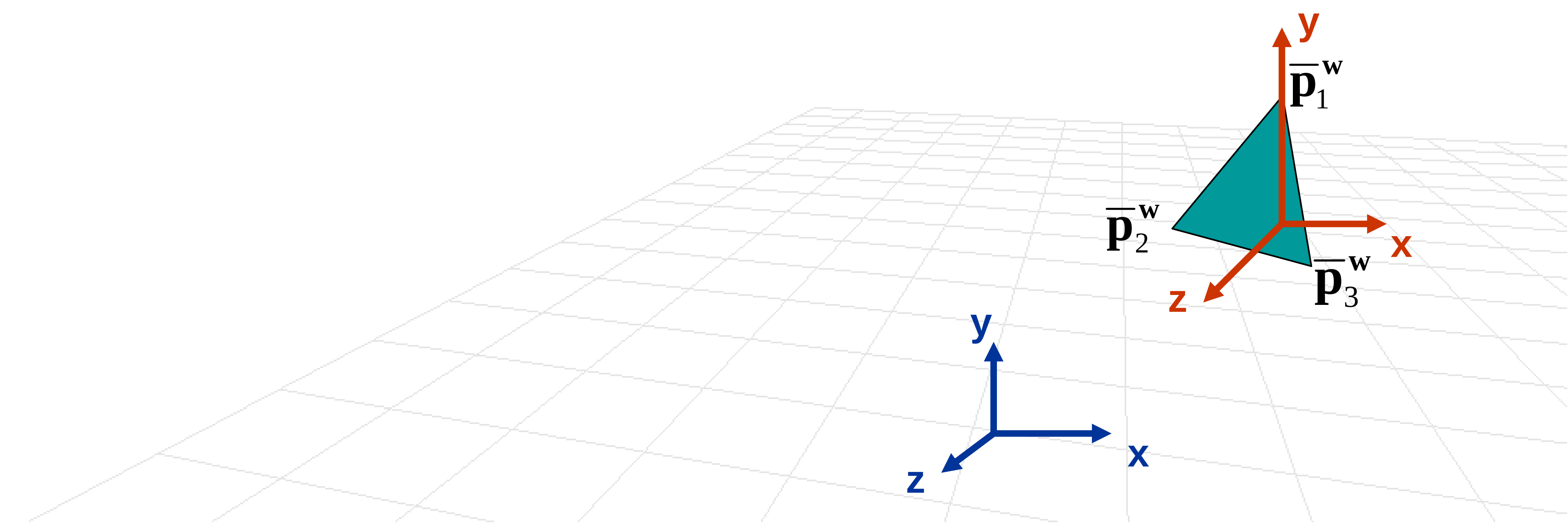
Projecting Triangle

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



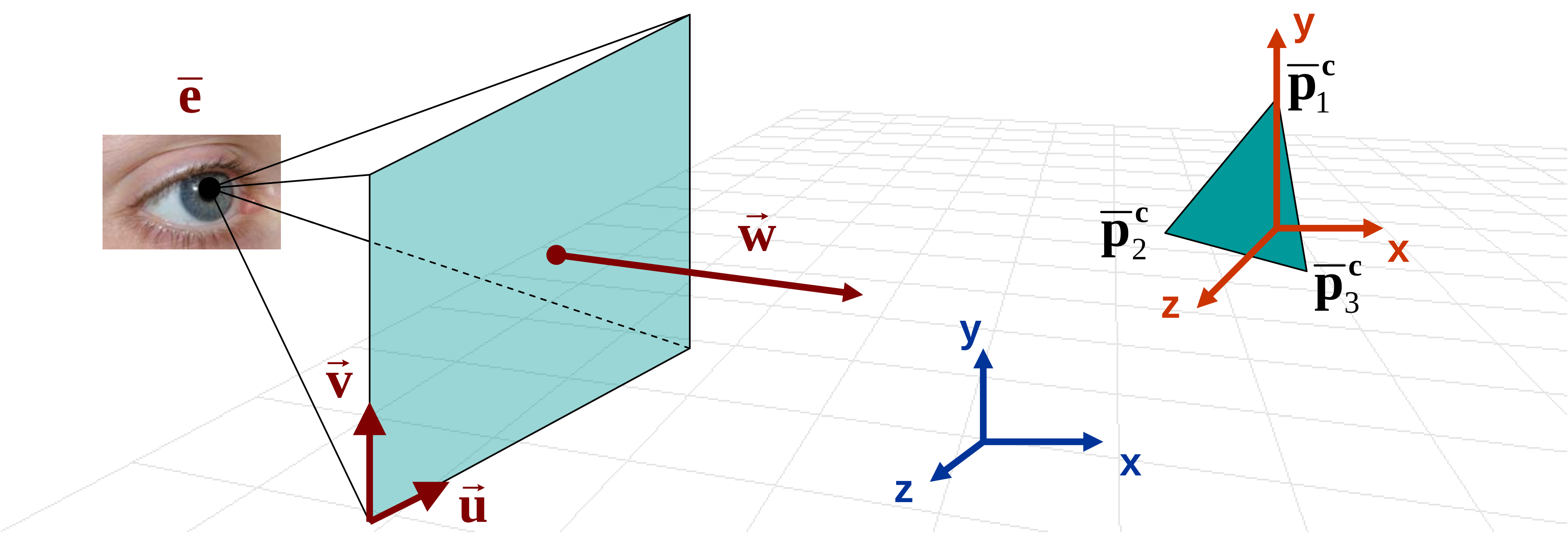
Projecting Triangle

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



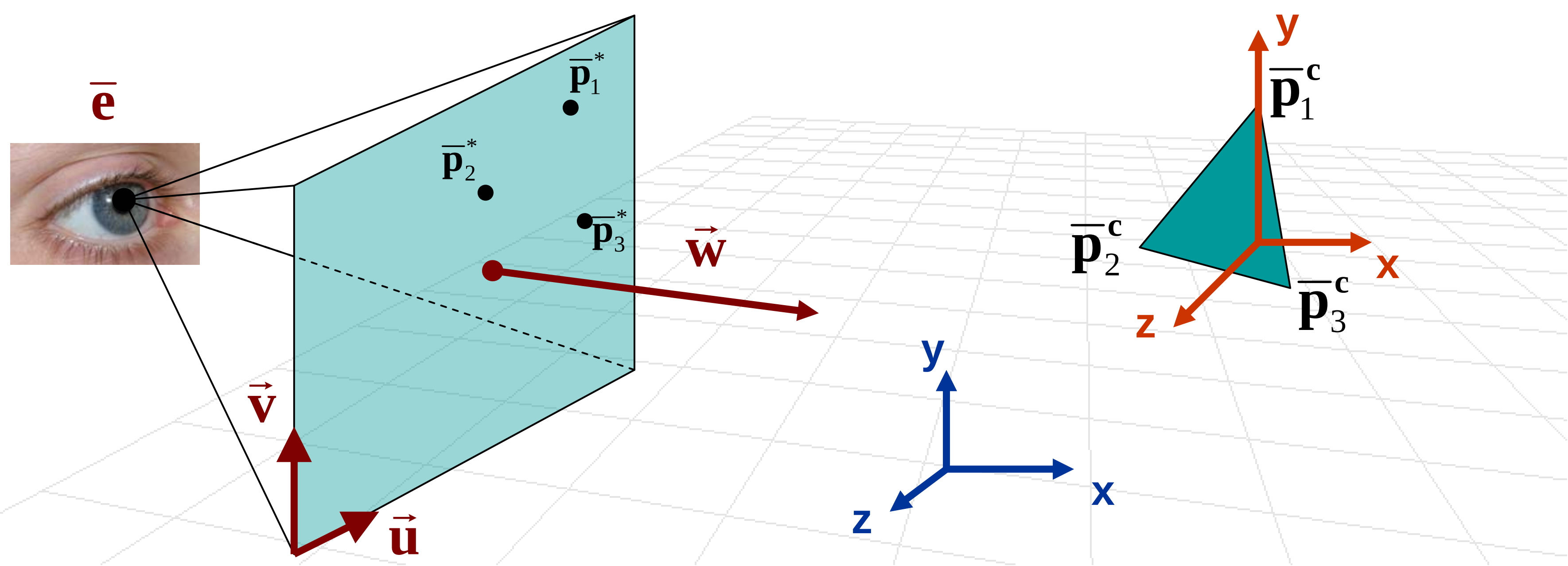
Projecting Triangle

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



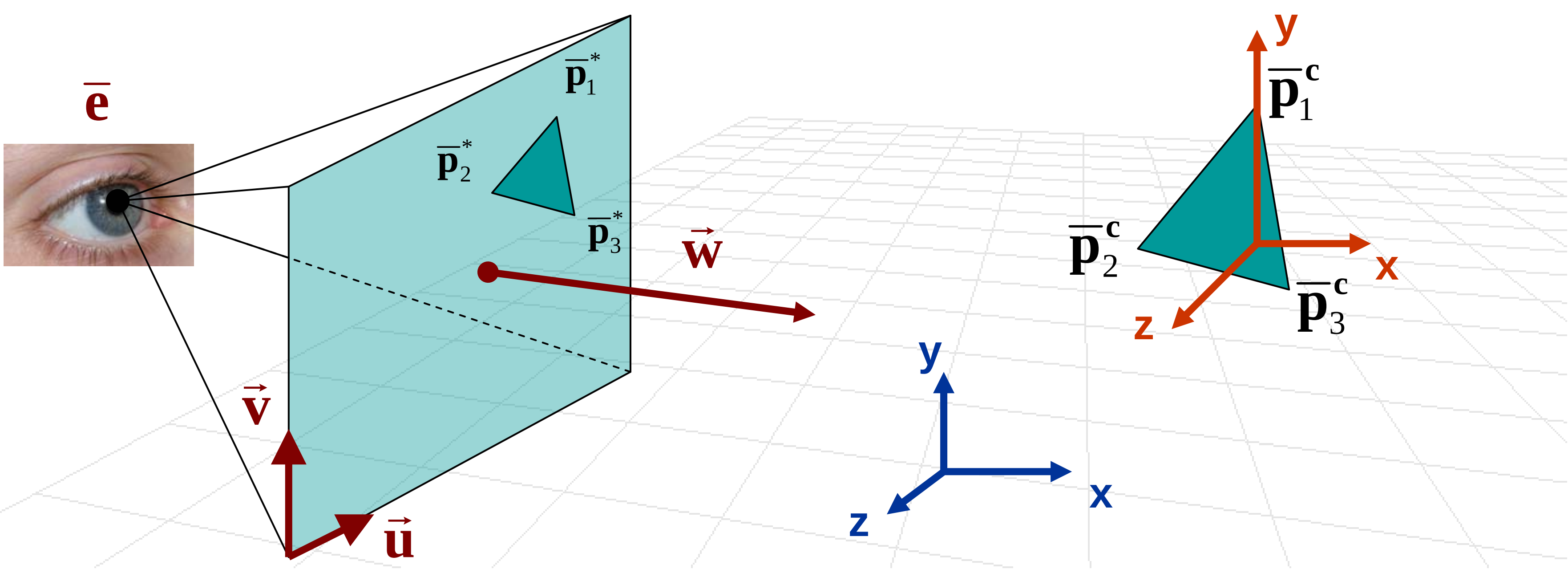
Projecting Triangle

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



Visibility

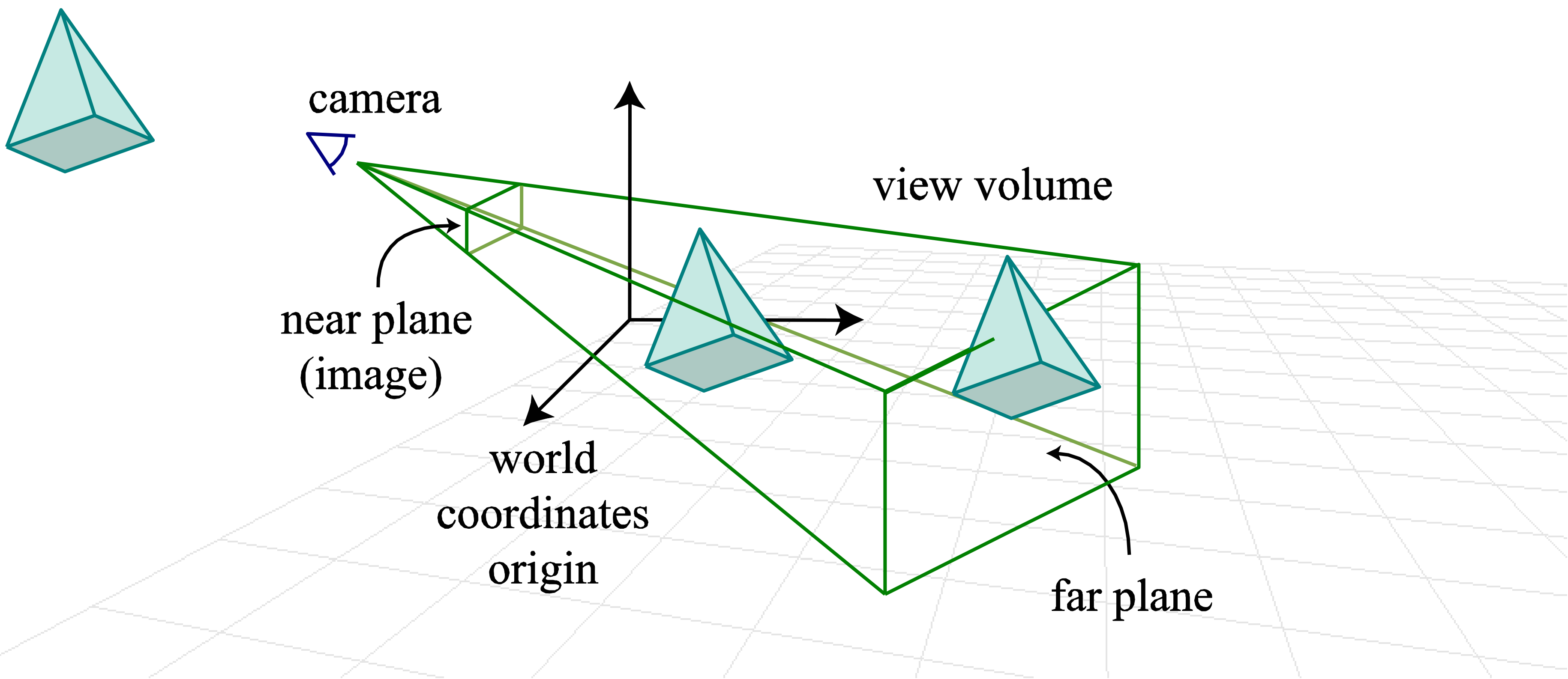
Computer Graphics, CSCD18

Fall 2008

Instructor: Leonid Sigal

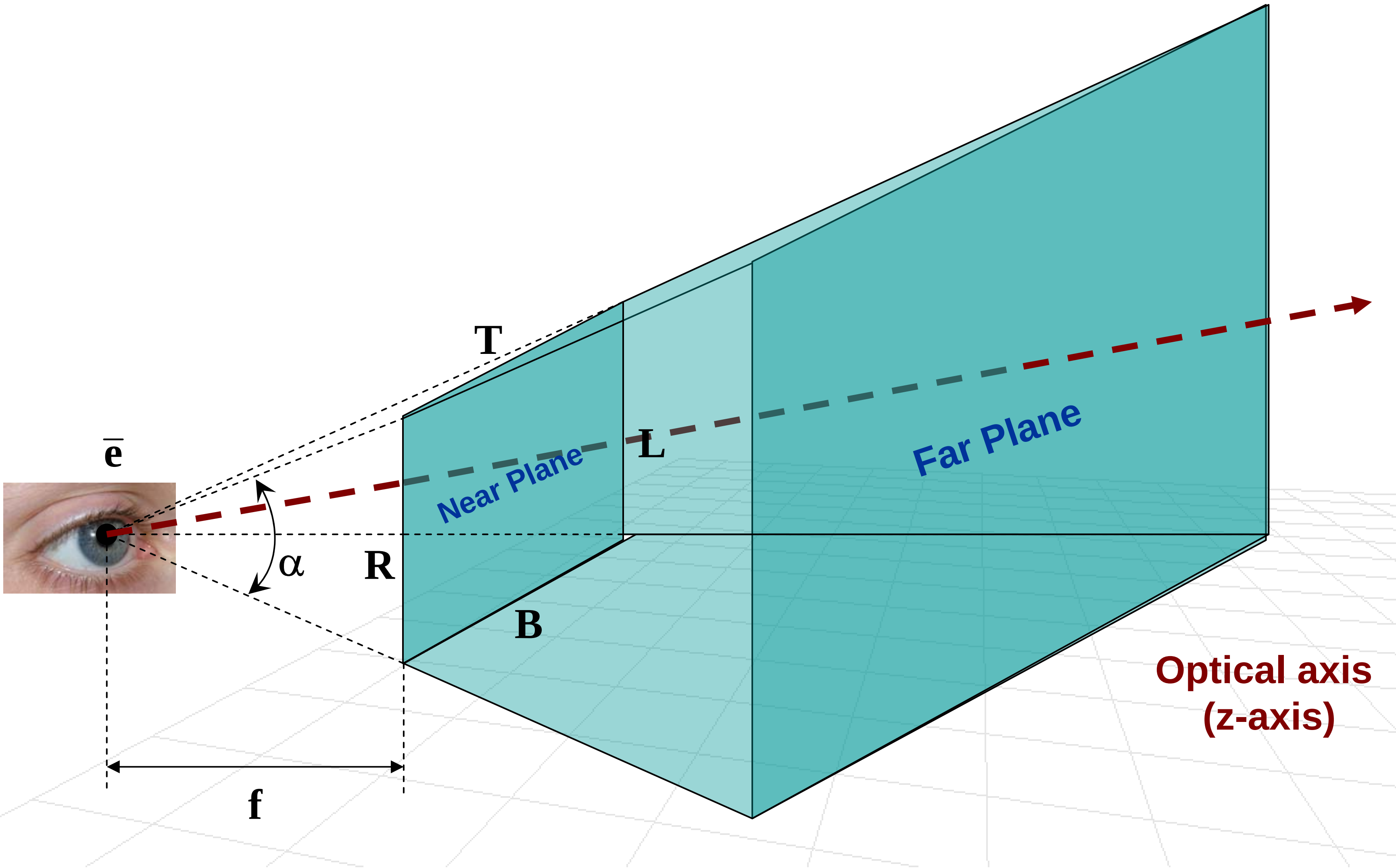
Clipping

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

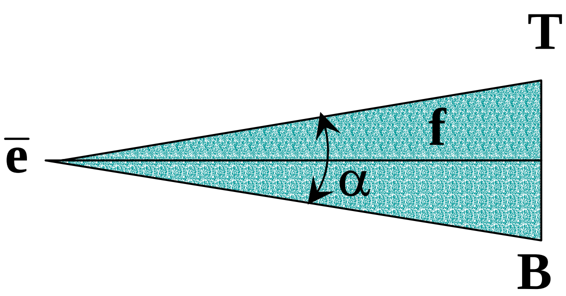


View Volume

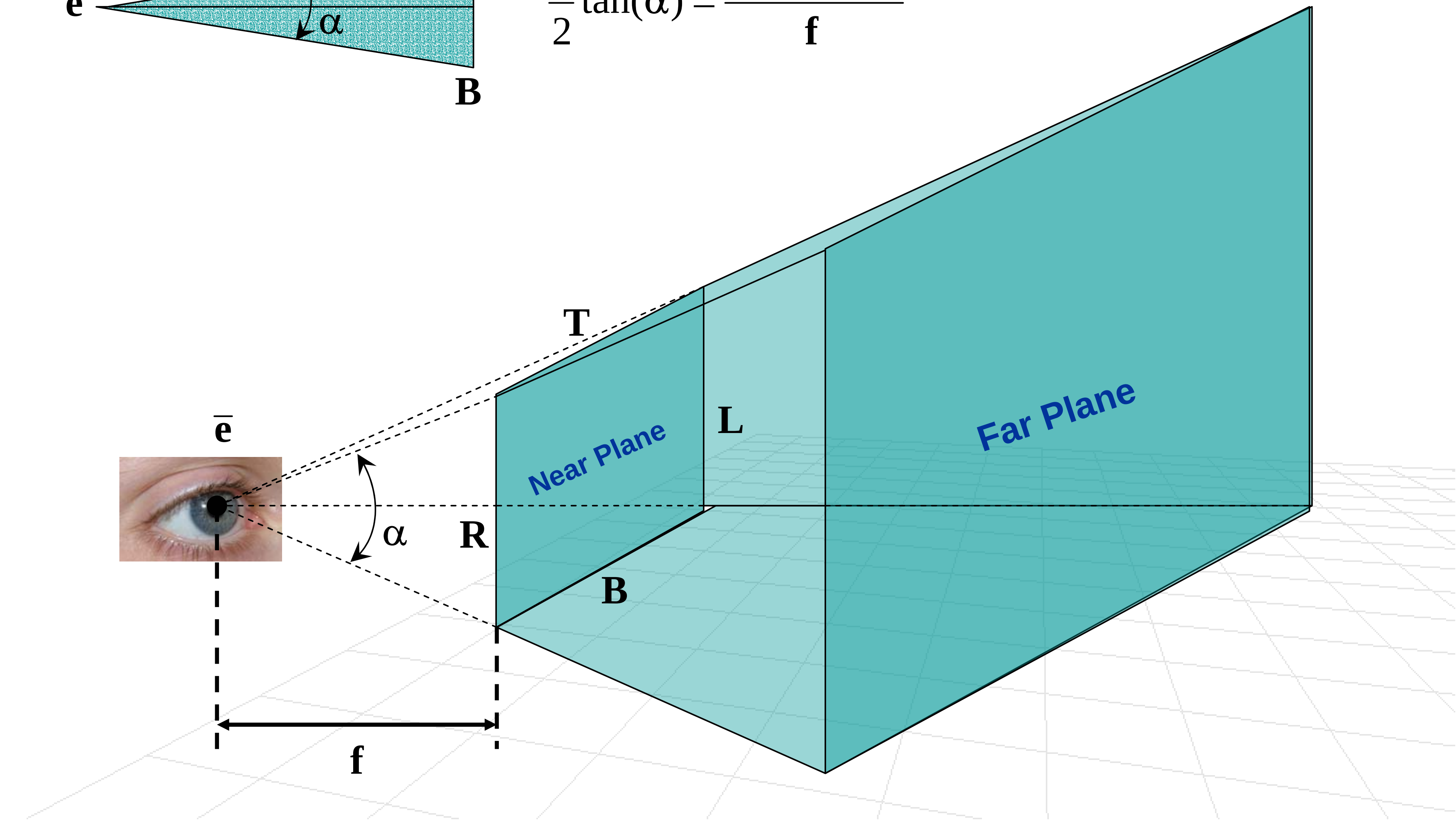
- Consider what we can actually see



Side note: Field of View



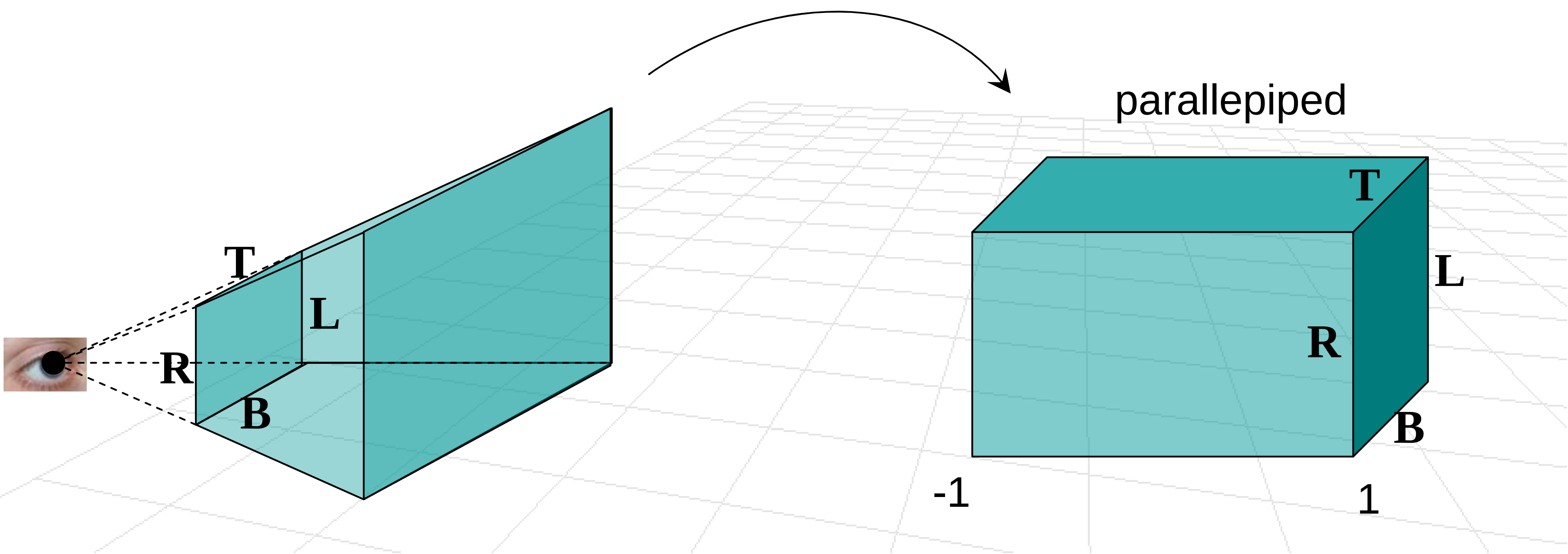
$$\frac{1}{2}\tan(\alpha) = \frac{1/2(\mathbf{T}-\mathbf{B})}{\mathbf{f}}$$



View Volume

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

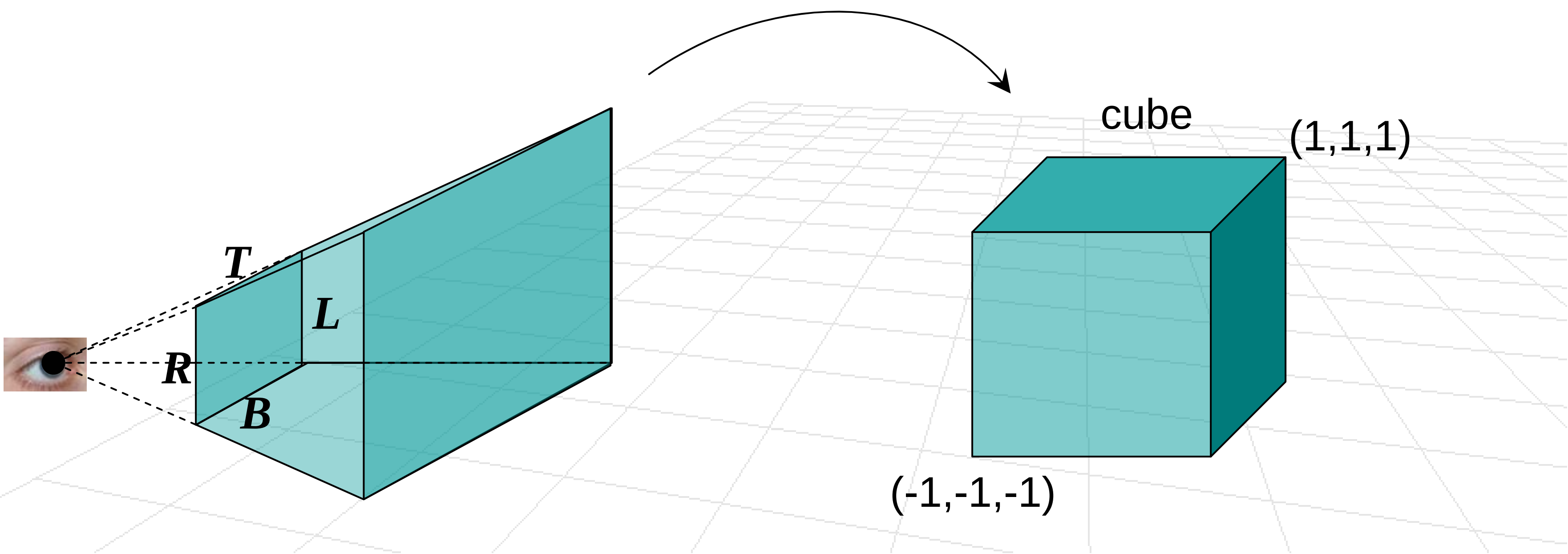
$$\mathbf{M_p} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \dfrac{2\mathbf{F}}{\mathbf{f}-\mathbf{F}} & -\dfrac{1}{\mathbf{f}}\left(\dfrac{\mathbf{f}+\mathbf{F}}{\mathbf{f}-\mathbf{F}}\right) \\ 0 & 0 & 1/\mathbf{f} & 0 \end{bmatrix}$$



Canonical View Volume

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

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

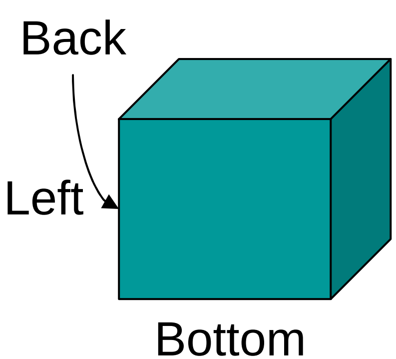


Back-face Removal

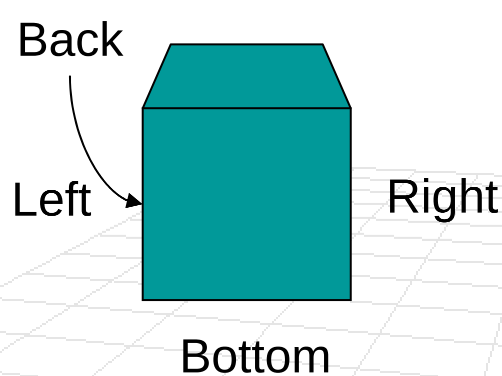
- **Idea:** Remove surface patches that point away from the camera (like backside of the object as it viewed from the front)

- Consider a cube

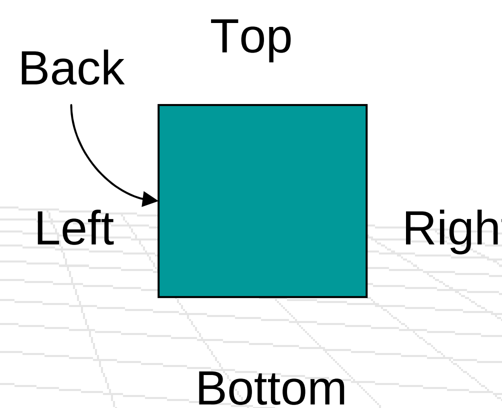
3 Back Faces



4 Back Faces



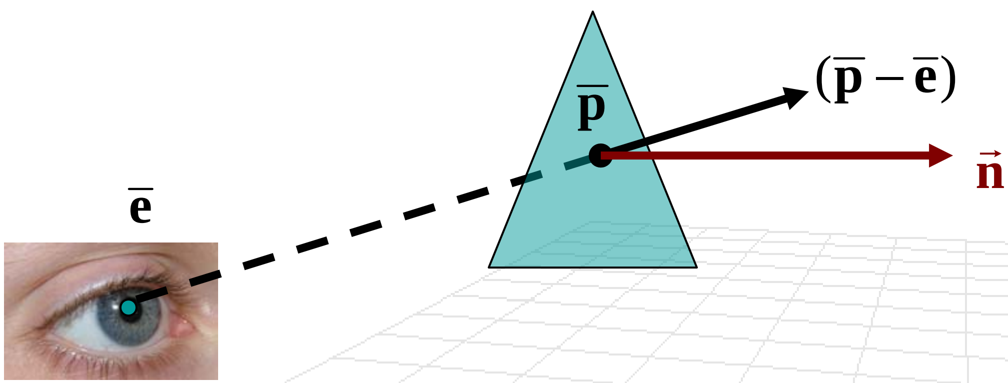
5 Back Faces



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?

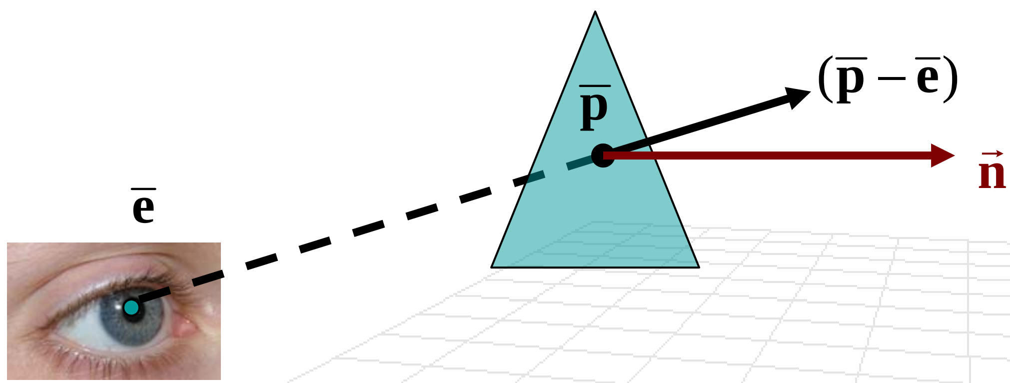


- If $(\bar{\mathbf{p}} - \bar{\mathbf{e}}) \cdot \vec{\mathbf{n}} > 0$ then triangle is part of the back-face and needs to be removed
- If $(\bar{\mathbf{p}} - \bar{\mathbf{e}}) \cdot \vec{\mathbf{n}} < 0$ then triangle **may** be visible

Back-face Removal

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

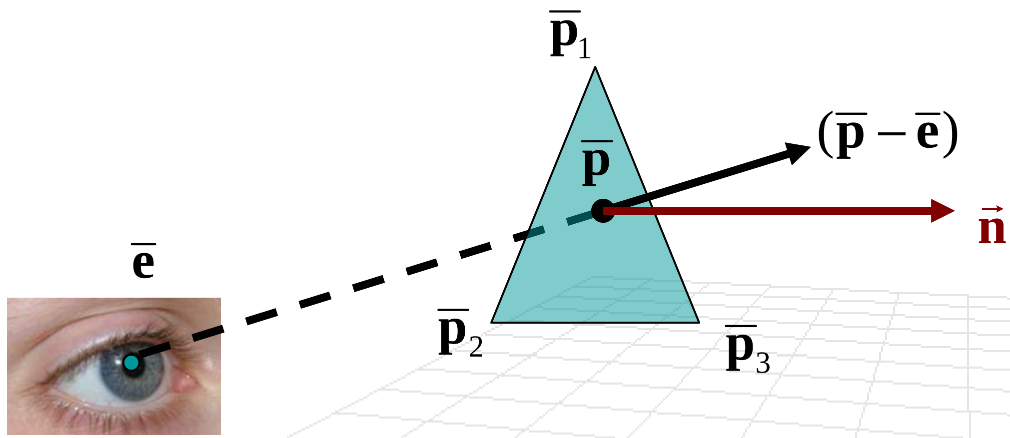
Consider a normal of the patch (triangle)



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

Back-face Removal

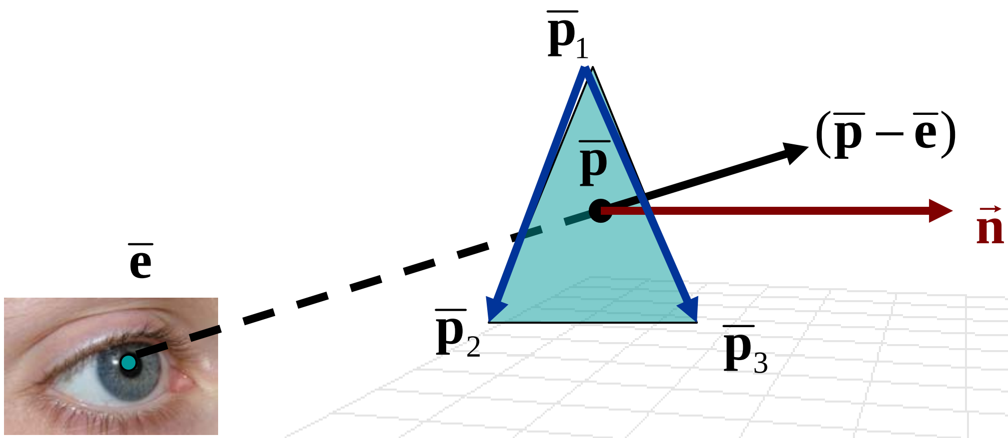
- Does it matter which point we consider on the patch?
  - Not if this is a **planar** patch
- How do we compute $\vec{\mathbf{n}}$
  - If $\overline{\mathbf{p}}_1, \overline{\mathbf{p}}_2, \overline{\mathbf{p}}_3$ are patch vertices in CCW order



- If $(\overline{\mathbf{p}} - \overline{\mathbf{e}}) \cdot \vec{\mathbf{n}} > 0$ then triangle is part of the back-face and needs to be removed
- If $(\overline{\mathbf{p}} - \overline{\mathbf{e}}) \cdot \vec{\mathbf{n}} < 0$ 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 $\vec{\mathbf{n}} = \dfrac{(\overline{\mathbf{p}}_2 - \overline{\mathbf{p}}_1)\times(\overline{\mathbf{p}}_3 - \overline{\mathbf{p}}_1)}{\left\|(\overline{\mathbf{p}}_2 - \overline{\mathbf{p}}_1)\times(\overline{\mathbf{p}}_3 - \overline{\mathbf{p}}_1)\right\|}$



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

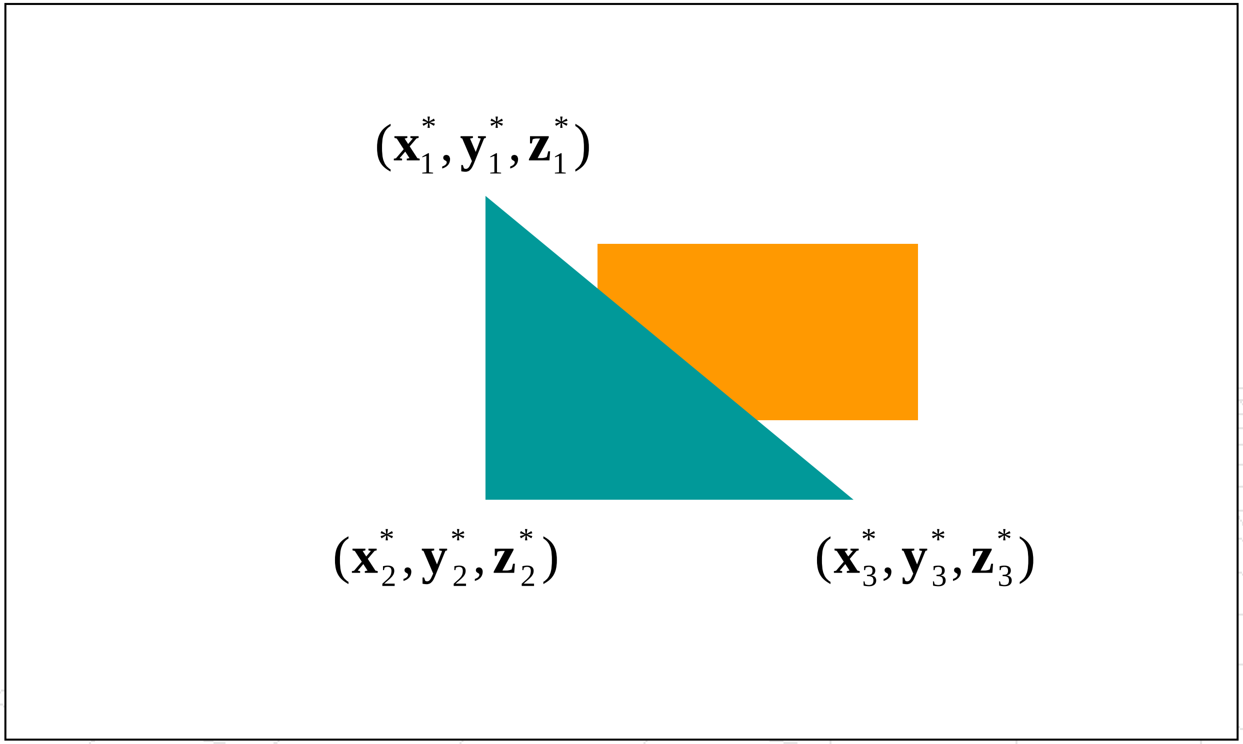
Z-Buffer (a.k.a Depth Buffer)

- **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)

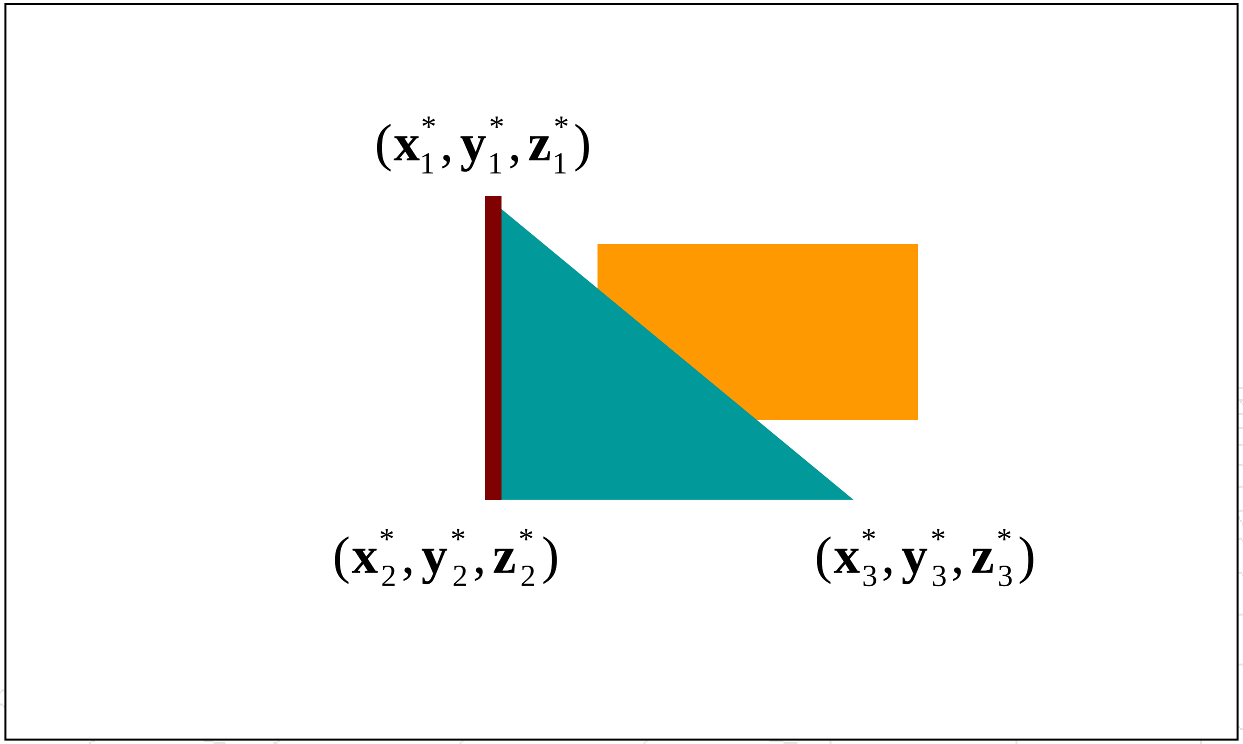
Z-Buffer (a.k.a Depth Buffer)

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



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

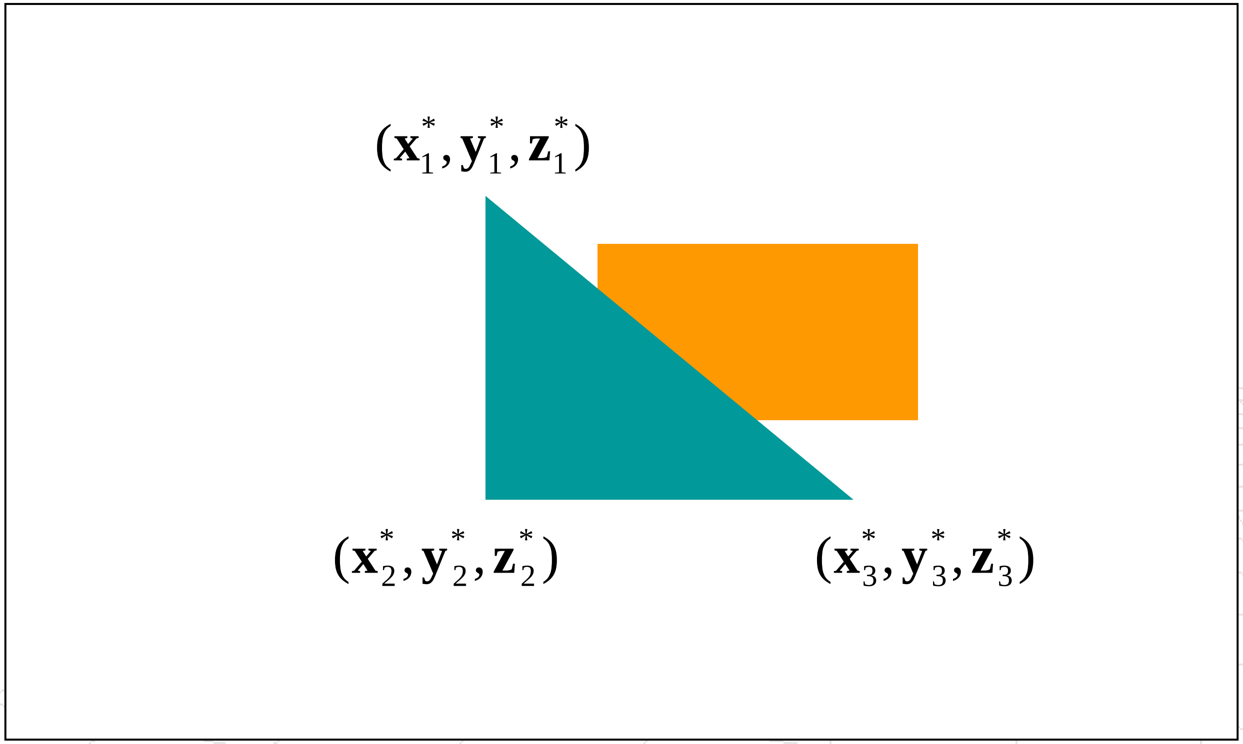
- We represent a patch using vertices
- How do we get a pseudodeph and proper rendering everywhere else?



Linearly interpolate $\mathbf{z}^*$ along a scan line

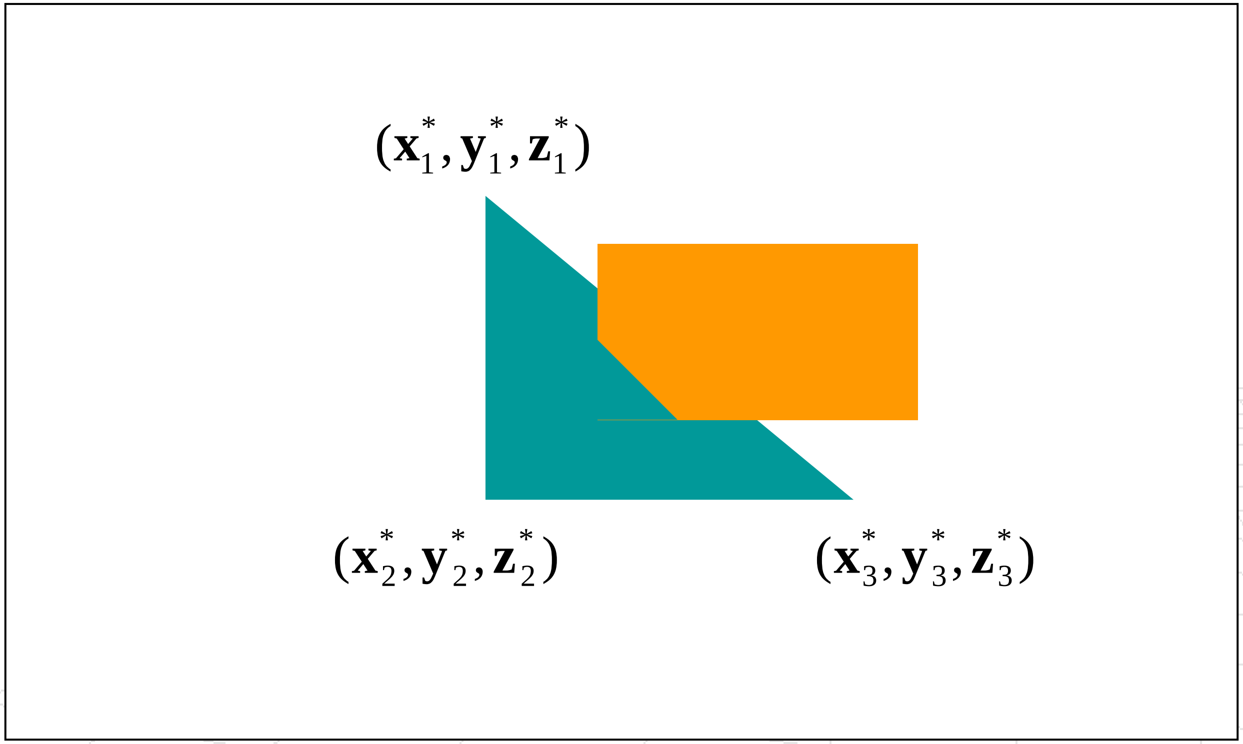
Painter's Algorithm

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



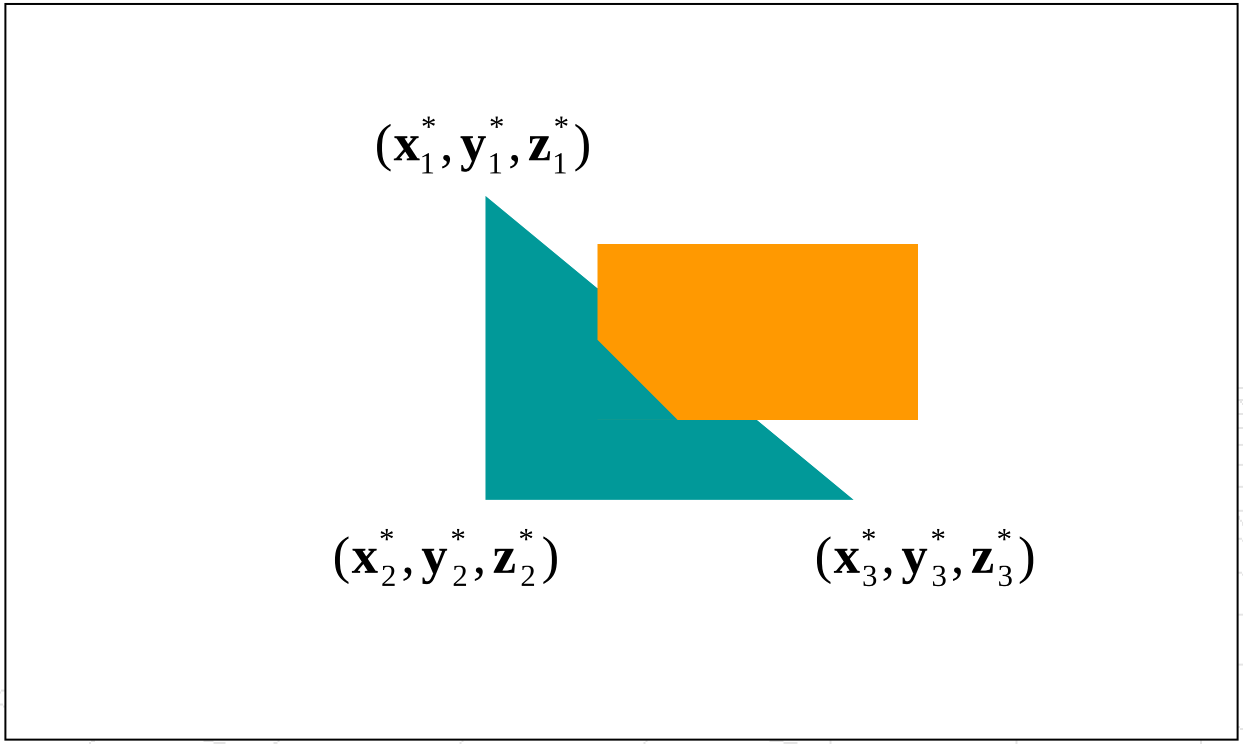
Painter's Algorithm

- 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

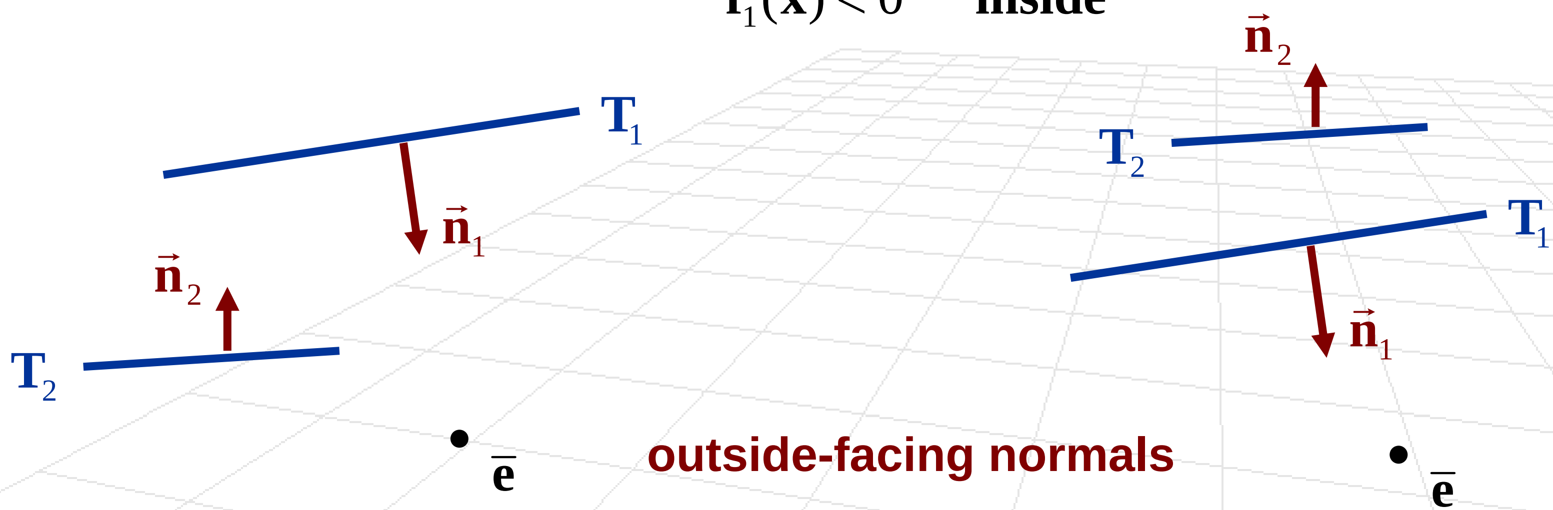
- If $\bar{\mathbf{e}}$ and $\mathbf{T_2}$ on the same side of $\mathbf{T_1}$ (left) then draw $\mathbf{T_1}$ first then $\mathbf{T_2}$
- If $\bar{\mathbf{e}}$ and $\mathbf{T_2}$ are on different sides of $\mathbf{T_1}$ (right) then draw $\mathbf{T_2}$ first then $\mathbf{T_1}$
- How do we know if points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

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

$\mathbf{f}_1(\bar{\mathbf{x}}) = 0$ **on the plane**

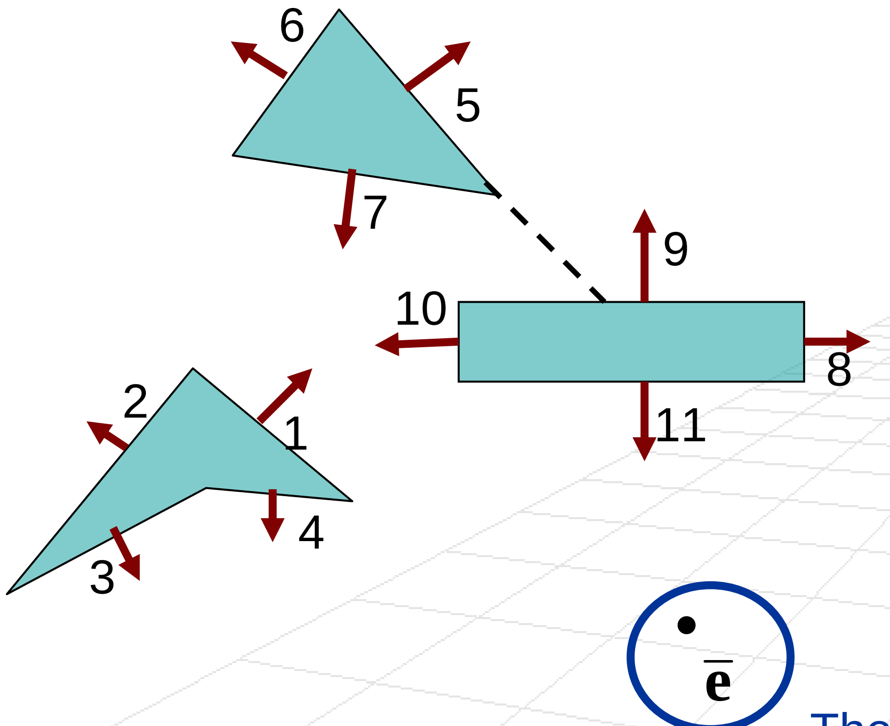
$\mathbf{f}_1(\bar{\mathbf{x}}) > 0$ "**outside**"

$\mathbf{f}_1(\bar{\mathbf{x}}) < 0$ "**inside**"



BSP Tree Example

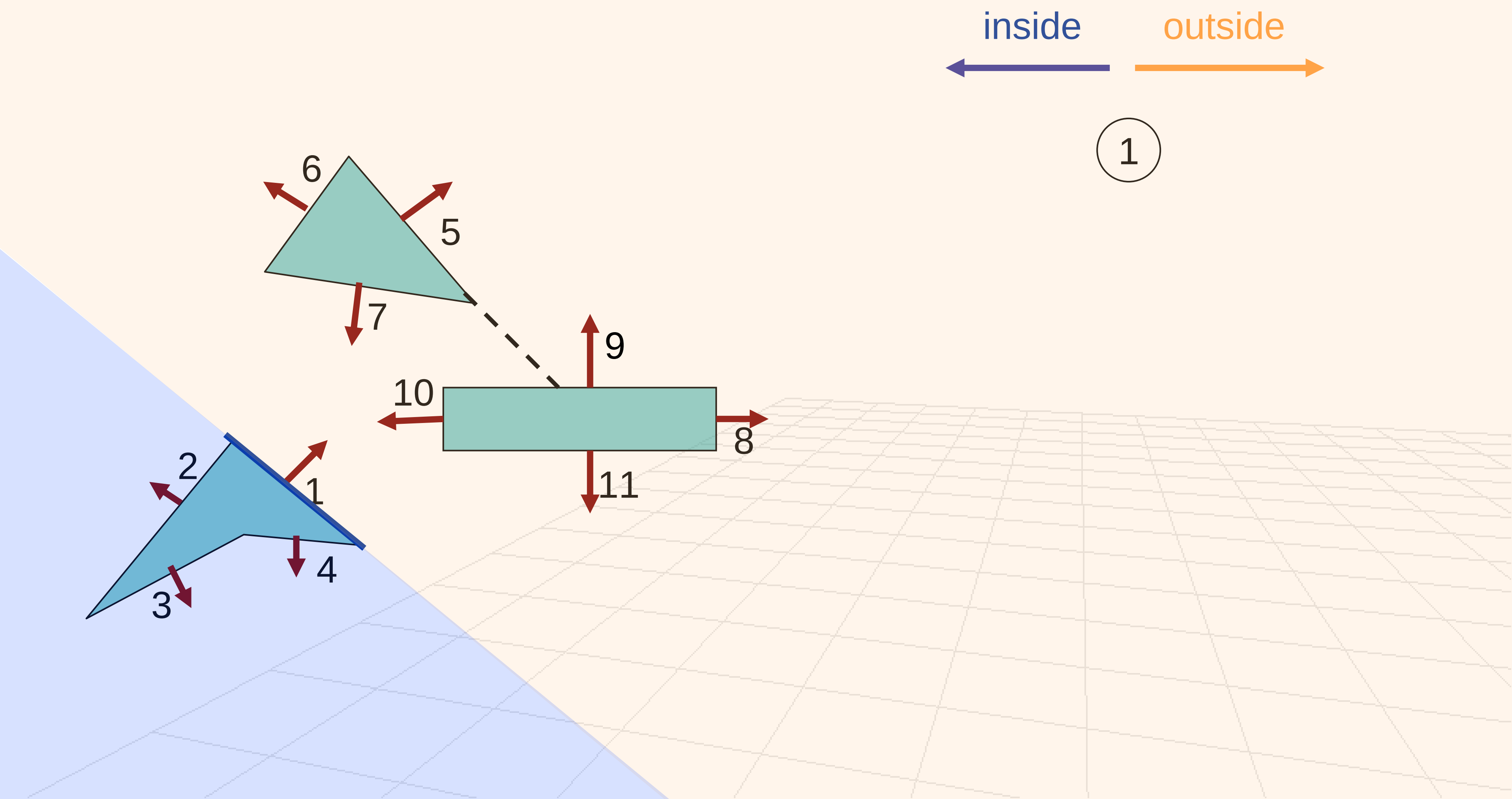
- Let’s try building a BSP tree for this scene



The tree will be the same regardless of the camera placement

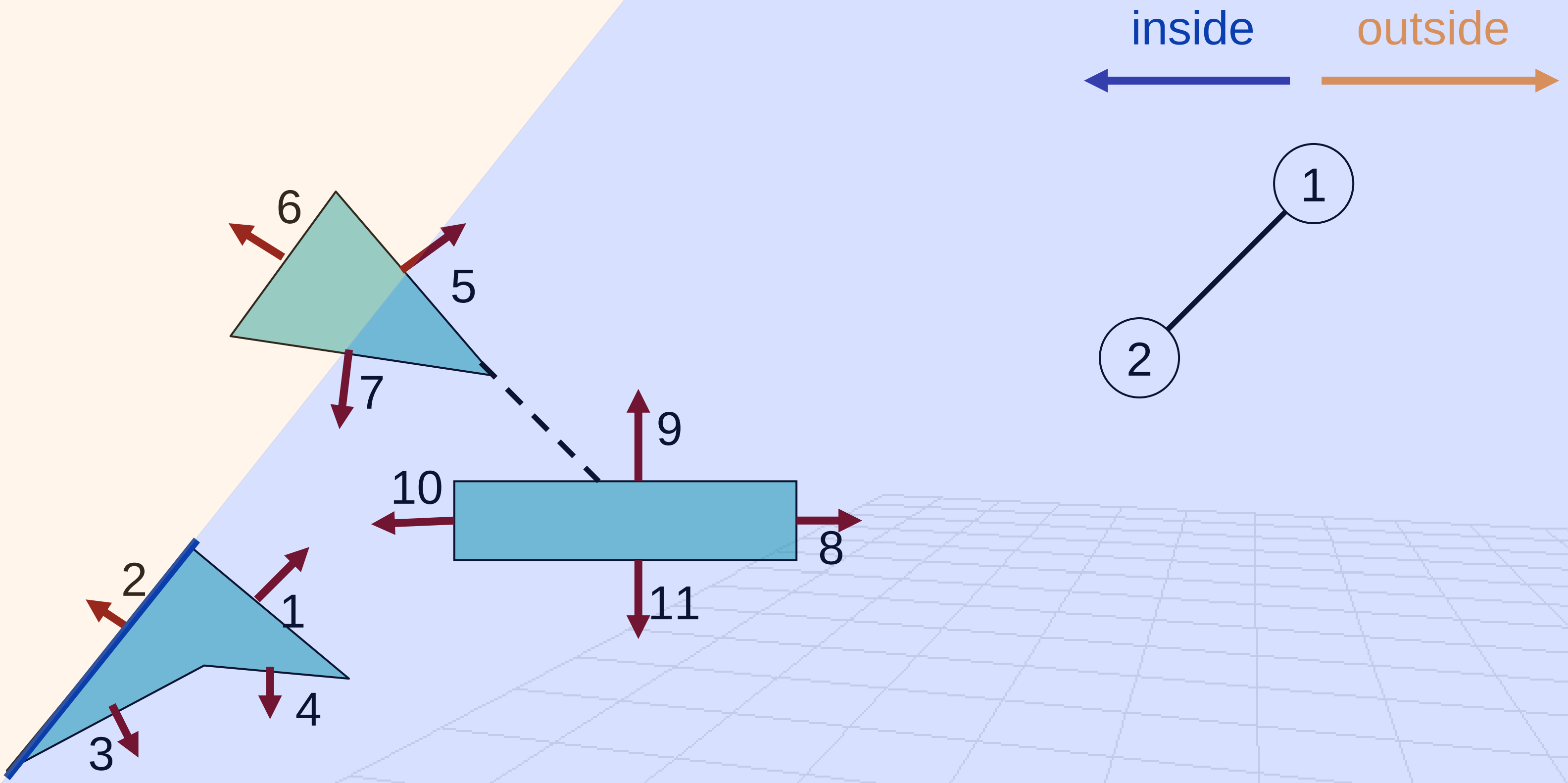
BSP Tree Example

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



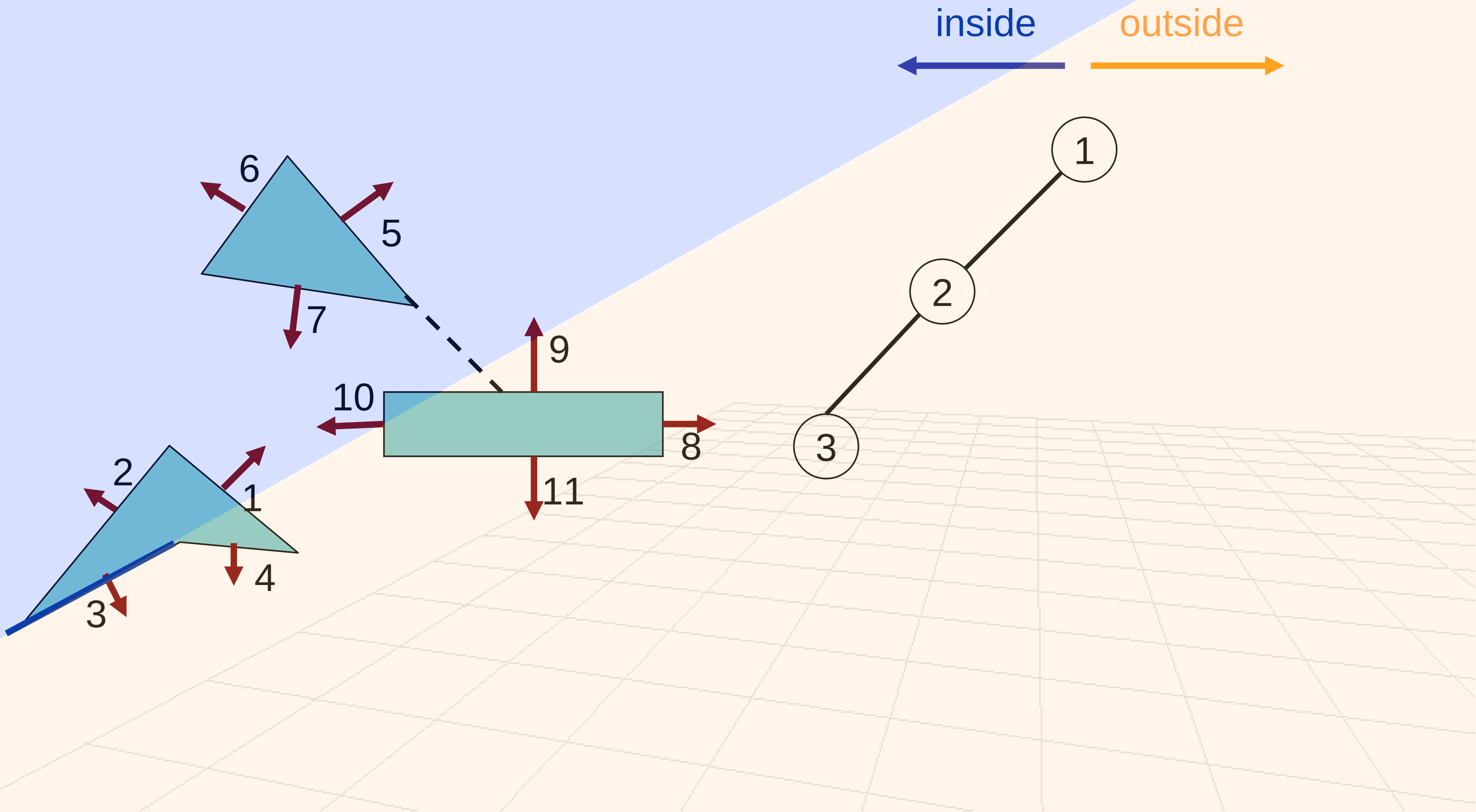
BSP Tree Example

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



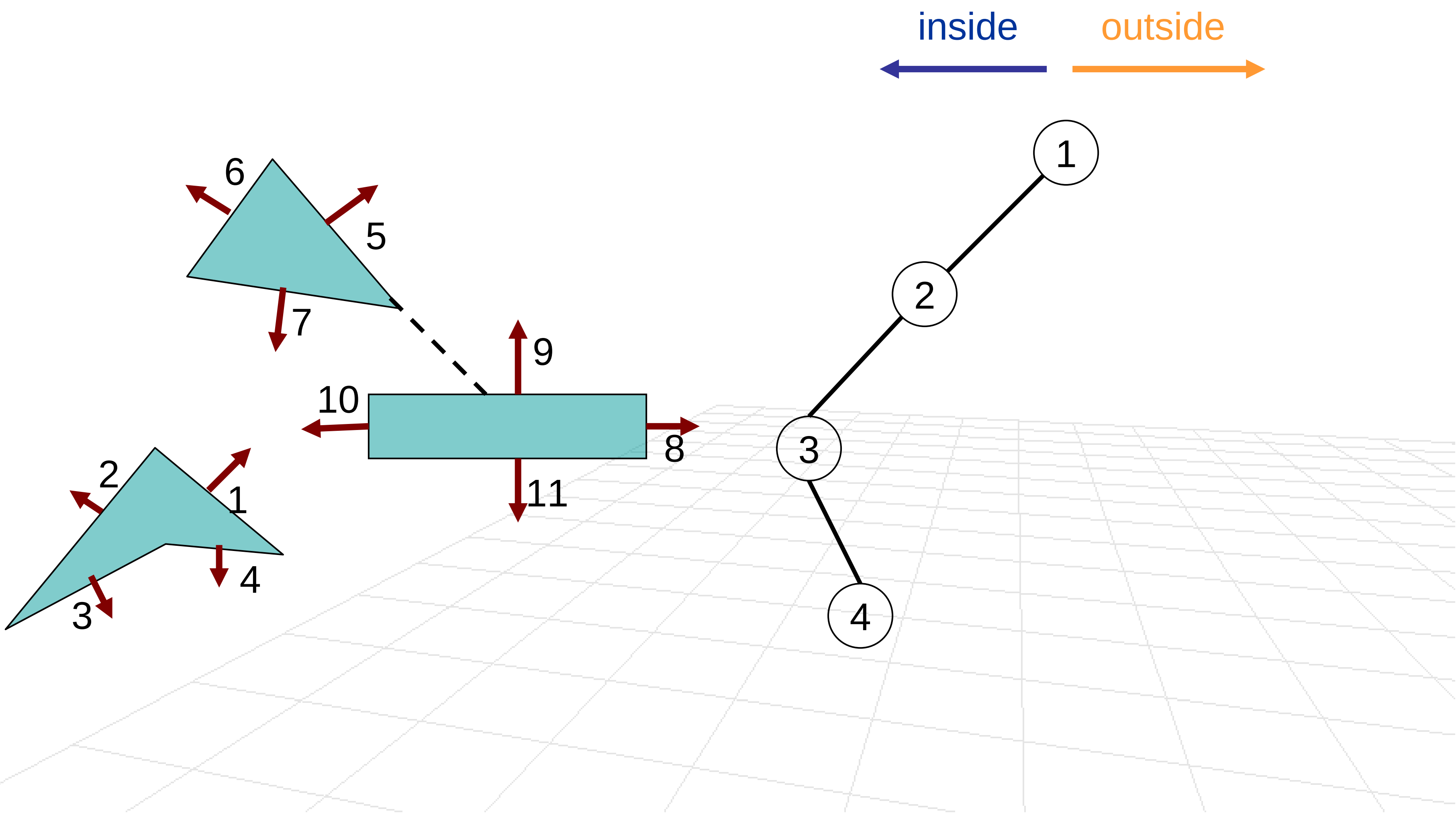
BSP Tree Example

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



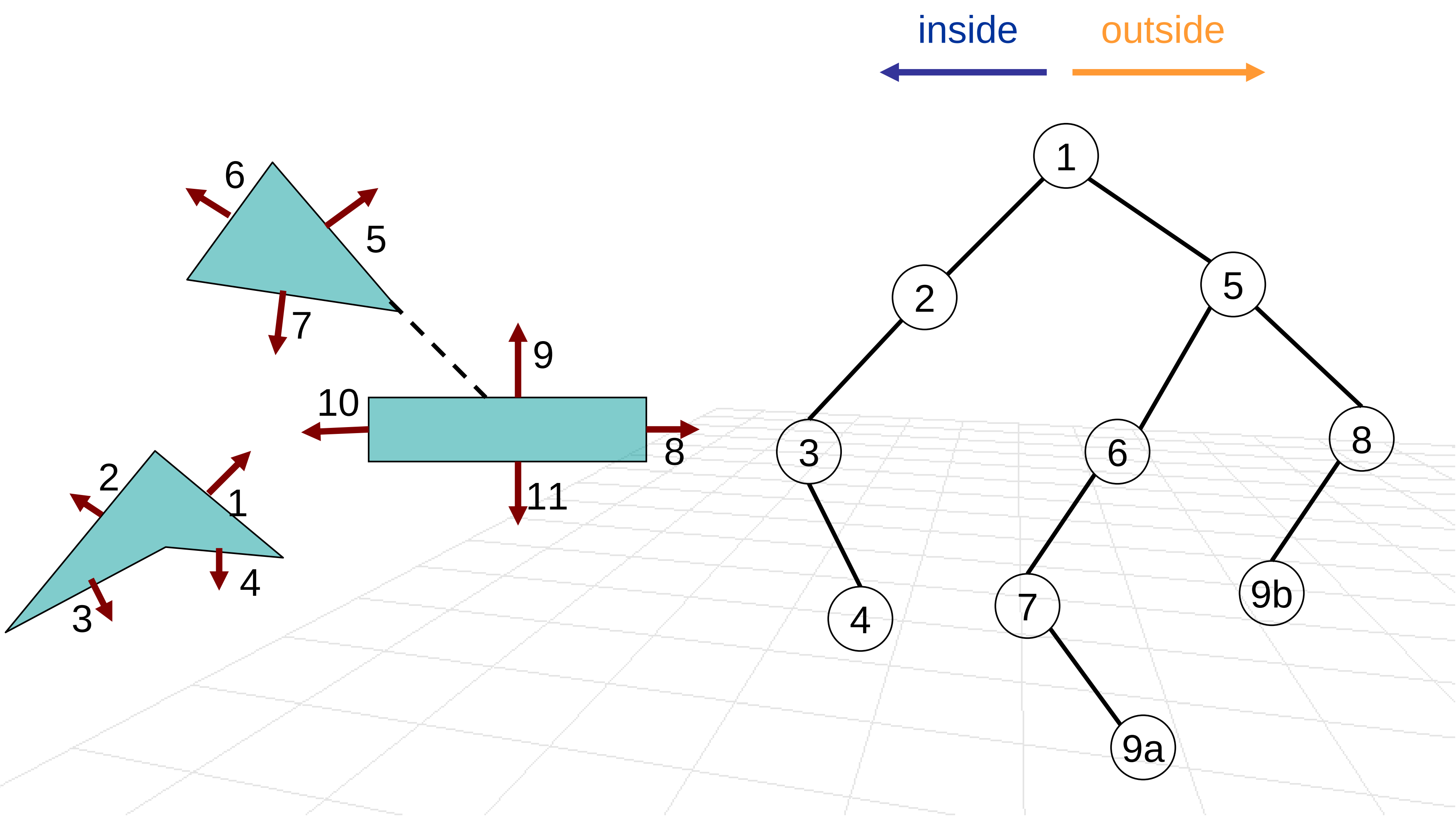
BSP Tree Example

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



BSP Tree Example

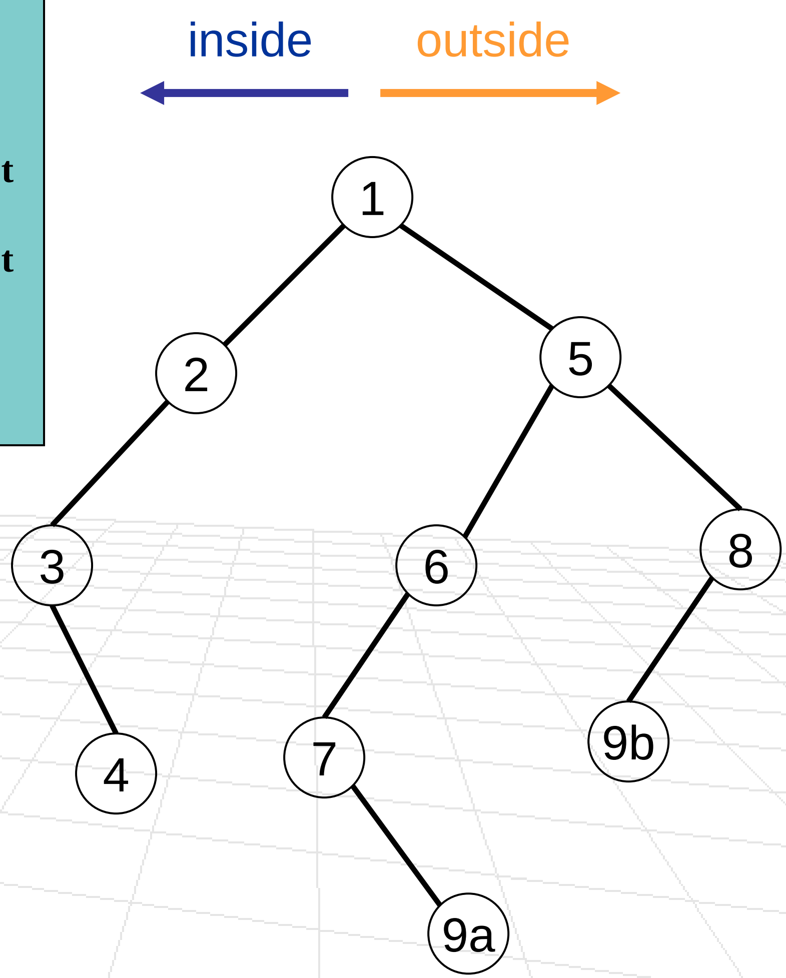
- Let's try building a BSP tree for this scene



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?

This is what is typically done in games, because it's fast