3D Rendering Geometry

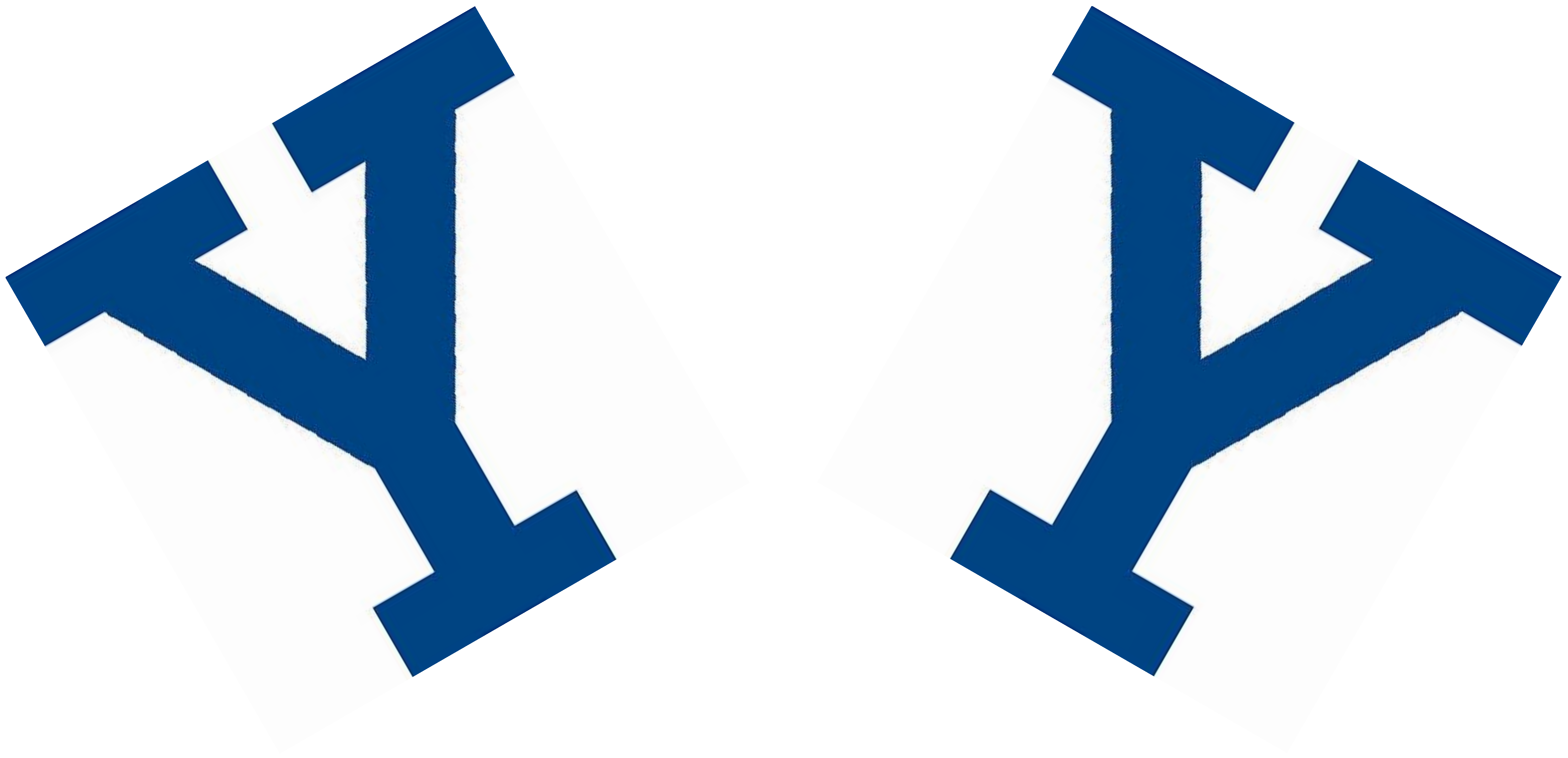
CS 355: Introduction to Graphics and Image Processing

First, a detour on object modeling…

Objects

Objects

Objects

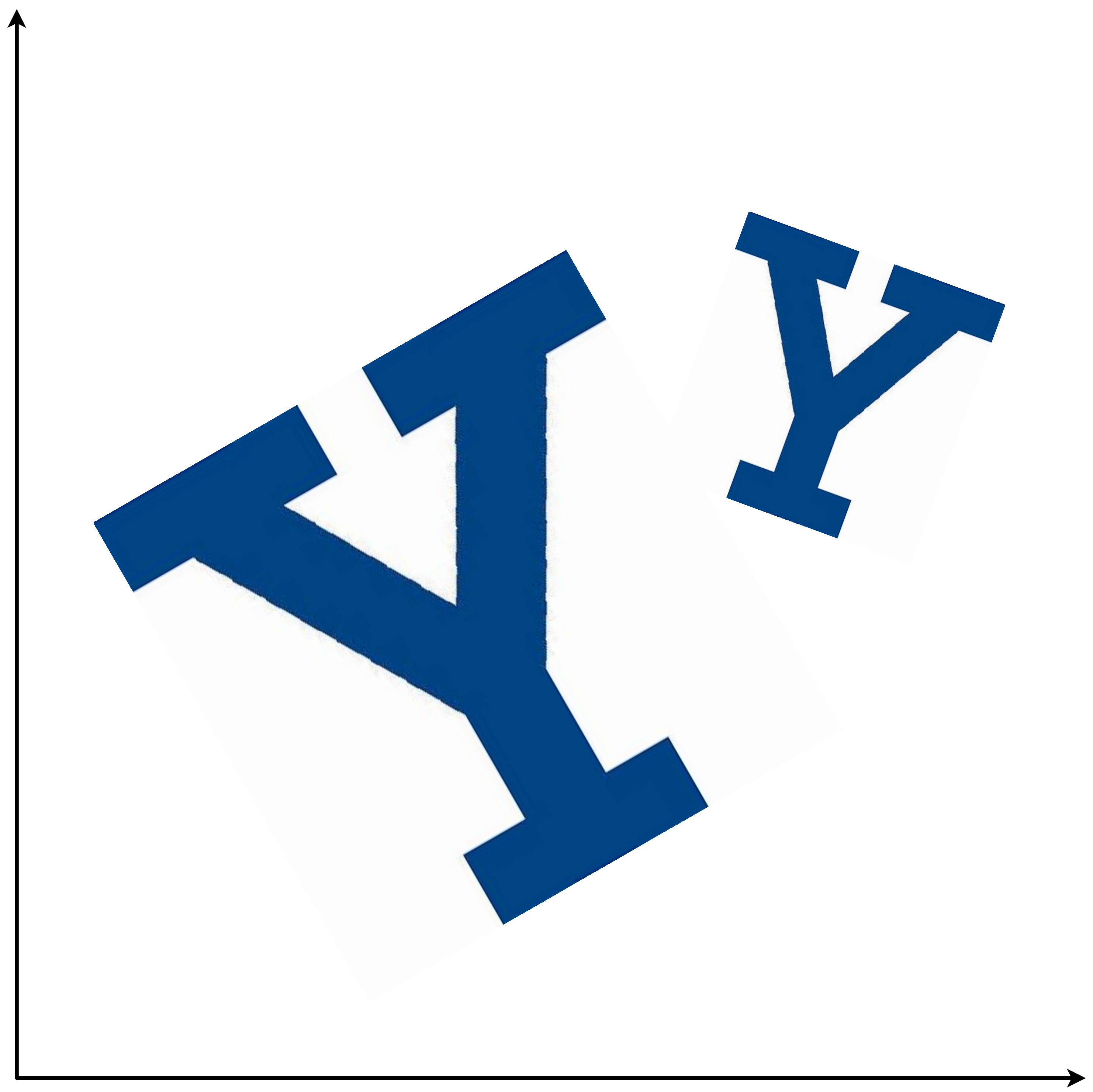


Objects

Objects

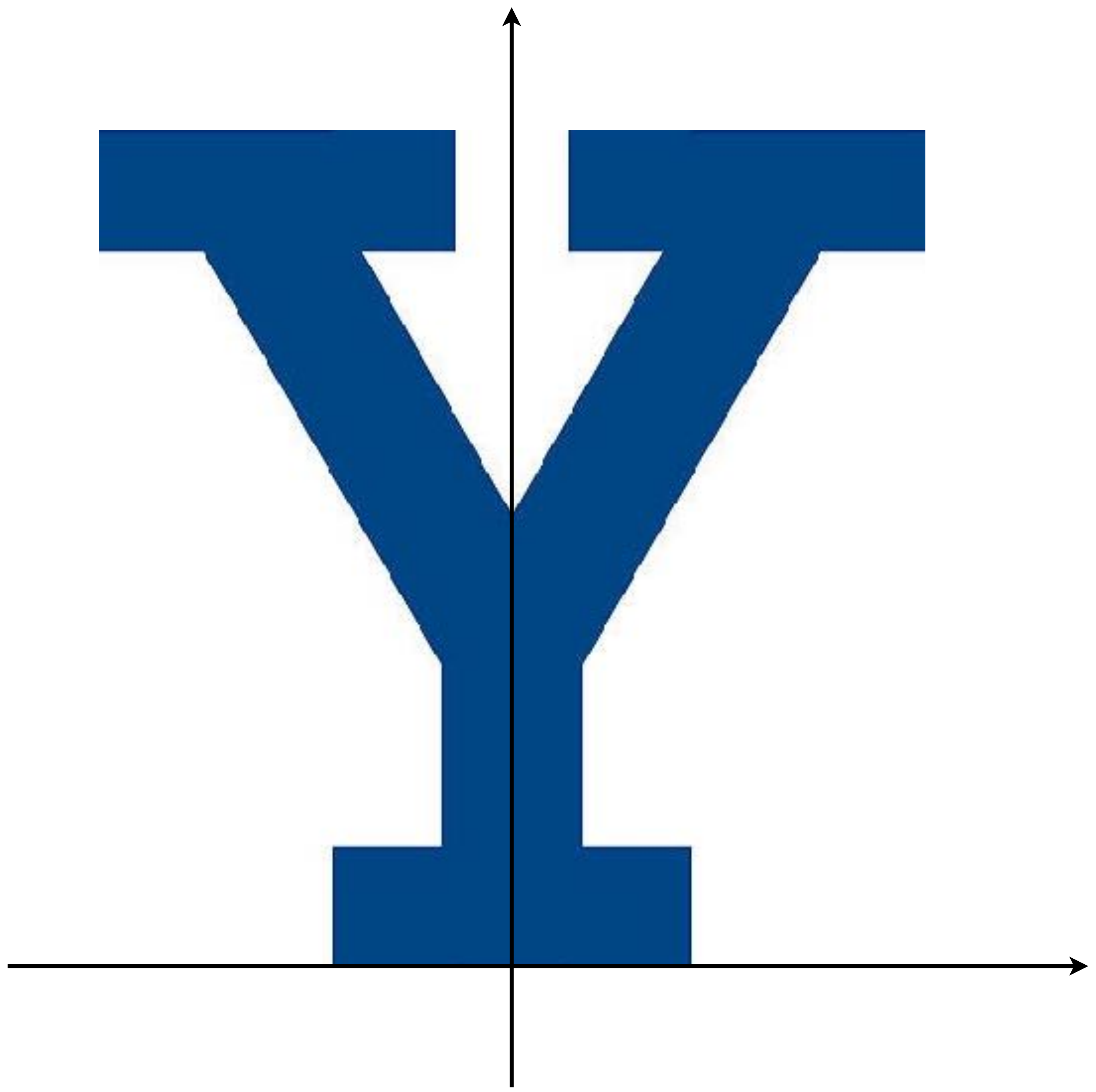
World Space

- The "world space" defines the space in which objects can live
- Choice of origin and coordinate system is arbitrary



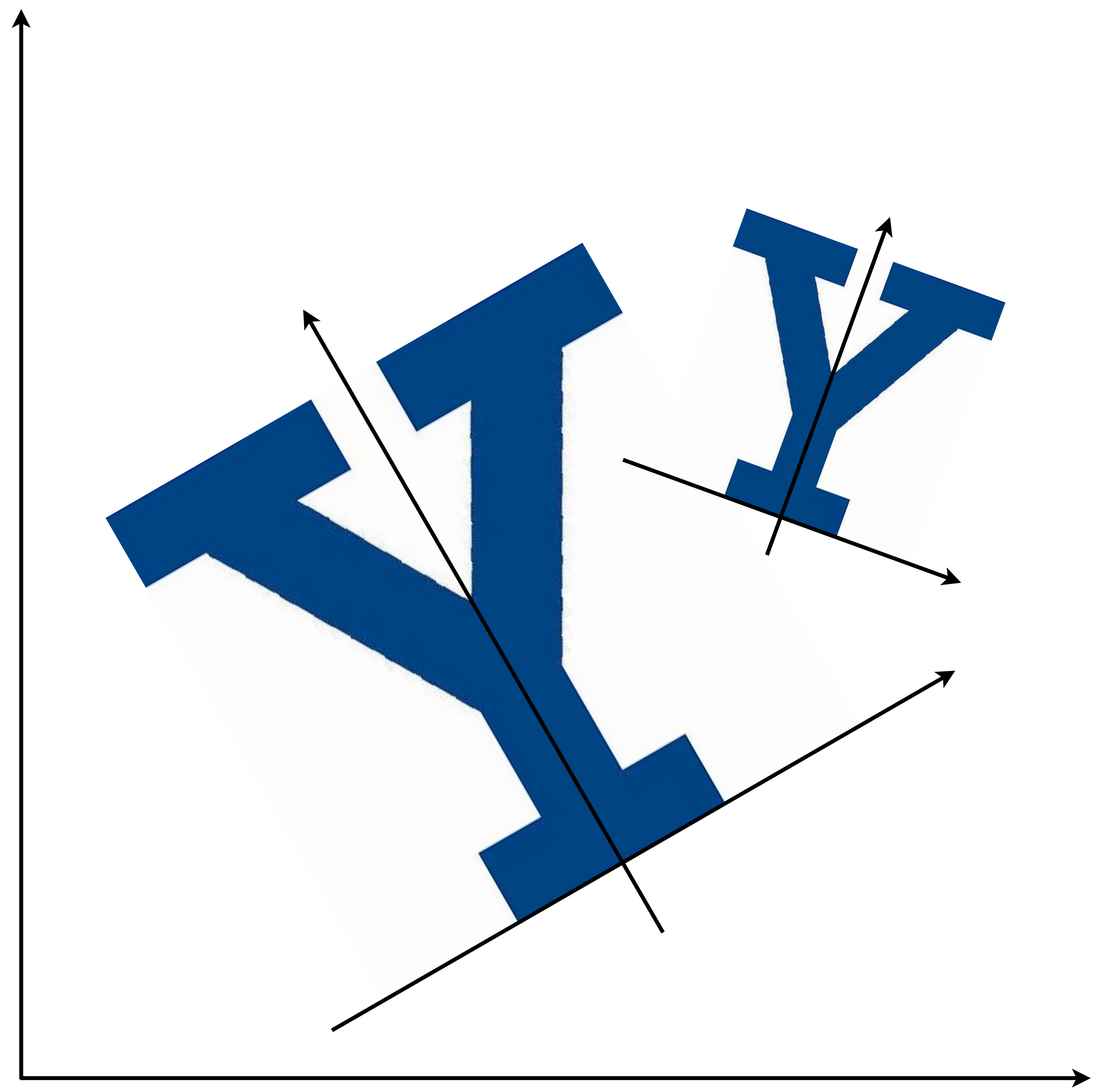
Object Space

- The coordinate system used to define an object
- Choice or origin and coordinate axes also arbitrary
- But usually chosen to make object definition the simplest



Objects in the World

- Placing an object in the world:
  - Size
  - Orientation
  - Location
- These define an *object-to-world transformation*



Object to World

- An object has a *position*, a *size*, and an *orientation*
  - First: **scale** in object space if needed (easiest if the coordinate axes are the natural directions for scaling)
  - Second: **rotate** in object space to desired world-space orientation
  - Third: **translate** (move) to the position in world space

Order matters!

Now back to rendering…

Rendering Geometry Pipeline

- Transform from object to world coordinates
- Transform from world to camera coordinates
- Preprocess to more efficiently handle things outside the field of view (we’re going to skip this for the moment)
- Perspective projection (to coordinates on the imaging plane)
- View transformation (to pixel coordinates on the screen)

Rendering Geometry Pipeline

- ✓ Transform from object to world coordinates
- Transform from world to camera coordinates
- Preprocess to more efficiently handle things outside the field of view (we’re going to skip this for the moment)
- ✓ Perspective projection (to coordinates on the imaging plane)
- View transformation (to pixel coordinates on the screen)

World to Camera

- Suppose that you know
  - Position of camera in world coordinates

$$\mathbf{c} = (c_x, c_y, c_z)$$

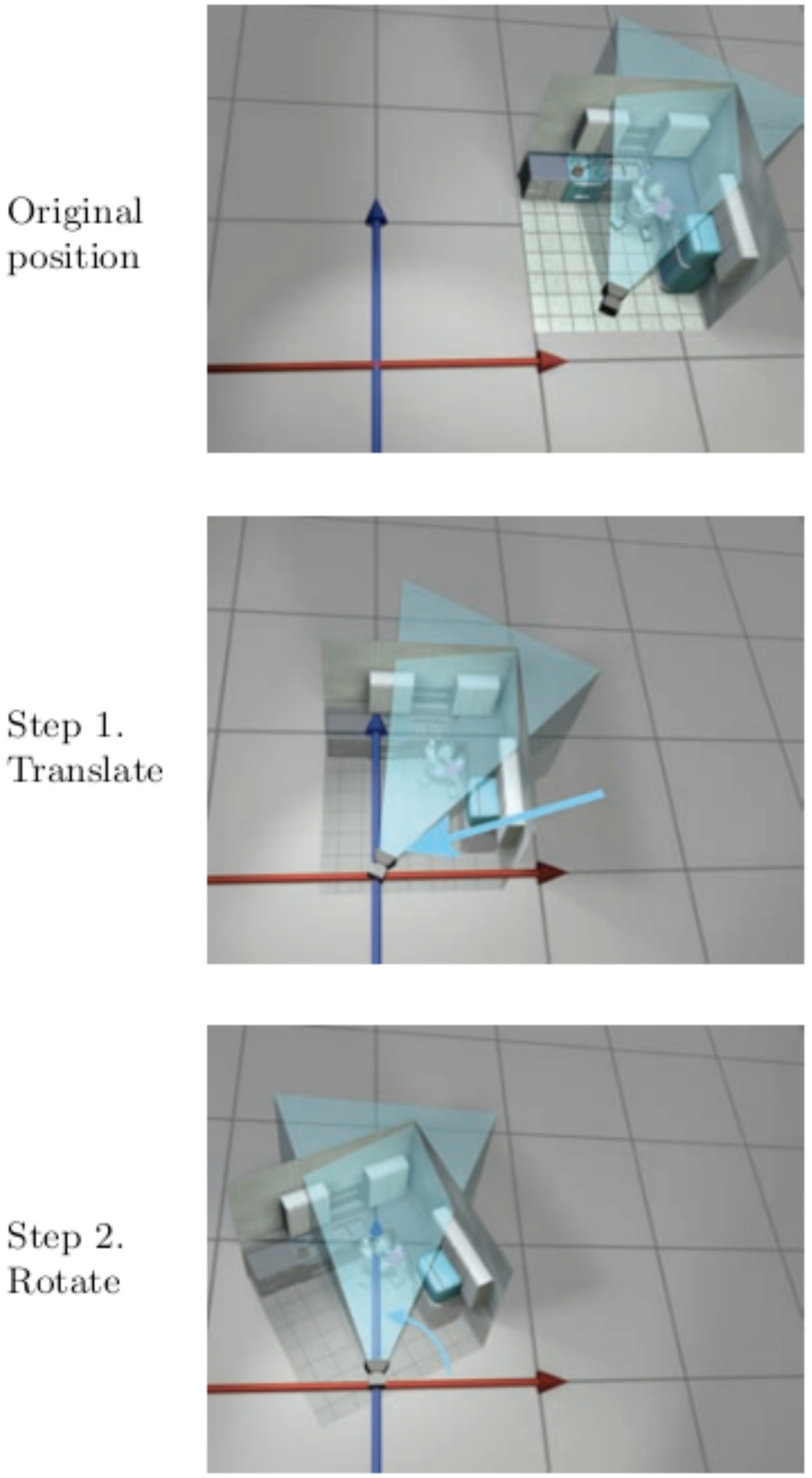
  - Orientation of camera as given by
    a set of basic vectors in world coordinates

$$\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$$

Camera's x Camera's y Camera's z

World to Camera

- Two steps:
  - **Translate**
    everything to be relative to the camera position
  - **Rotate**
    into the camera's viewing orientation



World to Camera

- Two steps:
  - **Translate**
    everything to be relative to the camera position
  - **Rotate**
    into the camera's viewing orientation

$$\begin{bmatrix} 1 & 0 & 0 & \boxed{\begin{matrix} -c_x \\ -c_y \\ -c_z \end{matrix}} \\ 0 & 1 & 0 & \\ 0 & 0 & 1 & \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{array}{l} \begin{bmatrix} e_{11} & e_{12} & e_{13} & 0 \\ e_{21} & e_{22} & e_{23} & 0 \\ e_{31} & e_{32} & e_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{array} \begin{array}{l} \text{Camera's x} \\ \text{Camera's y} \\ \text{Camera's z} \\ \end{array}$$

Putting It Together With Projection

World-to-camera transformation

$$\begin{bmatrix} \boxed{\begin{matrix} x \\ y \end{matrix}} \\ f \\ 1 \end{bmatrix} \sim \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ Z_c/f \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/f & 0 \end{bmatrix} \begin{bmatrix} e_{11} & e_{12} & e_{13} & 0 \\ e_{21} & e_{22} & e_{23} & 0 \\ e_{31} & e_{32} & e_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 1 & -c_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boxed{\begin{matrix} X_w \\ Y_w \\ Z_w \end{matrix}} \\ 1 \end{bmatrix}$$

Normalize Project Rotate Translate

Rendering Geometry

- ✓ Transform from object to world coordinates
- ✓ Transform from world to camera coordinates
- Preprocess to more efficiently handle things outside the field of view (we're going to skip this for the moment)
- ✓ Perspective projection (to coordinates on the imaging plane)
- View transformation (to pixel coordinates on the screen)

To Screen Space

- Perspective projection gives you projected coordinates on the imaging plane
  - real-world units
  - centered at the focal point (intersection of the optical axis)
- We want actual on-screen pixel coordinates
- Simple transformation:
  - Multiply by the sampling density (pixels per real-world unit)
    Need to specify the resolution of the image we're trying to render
  - Translate the origin to the upper left corner

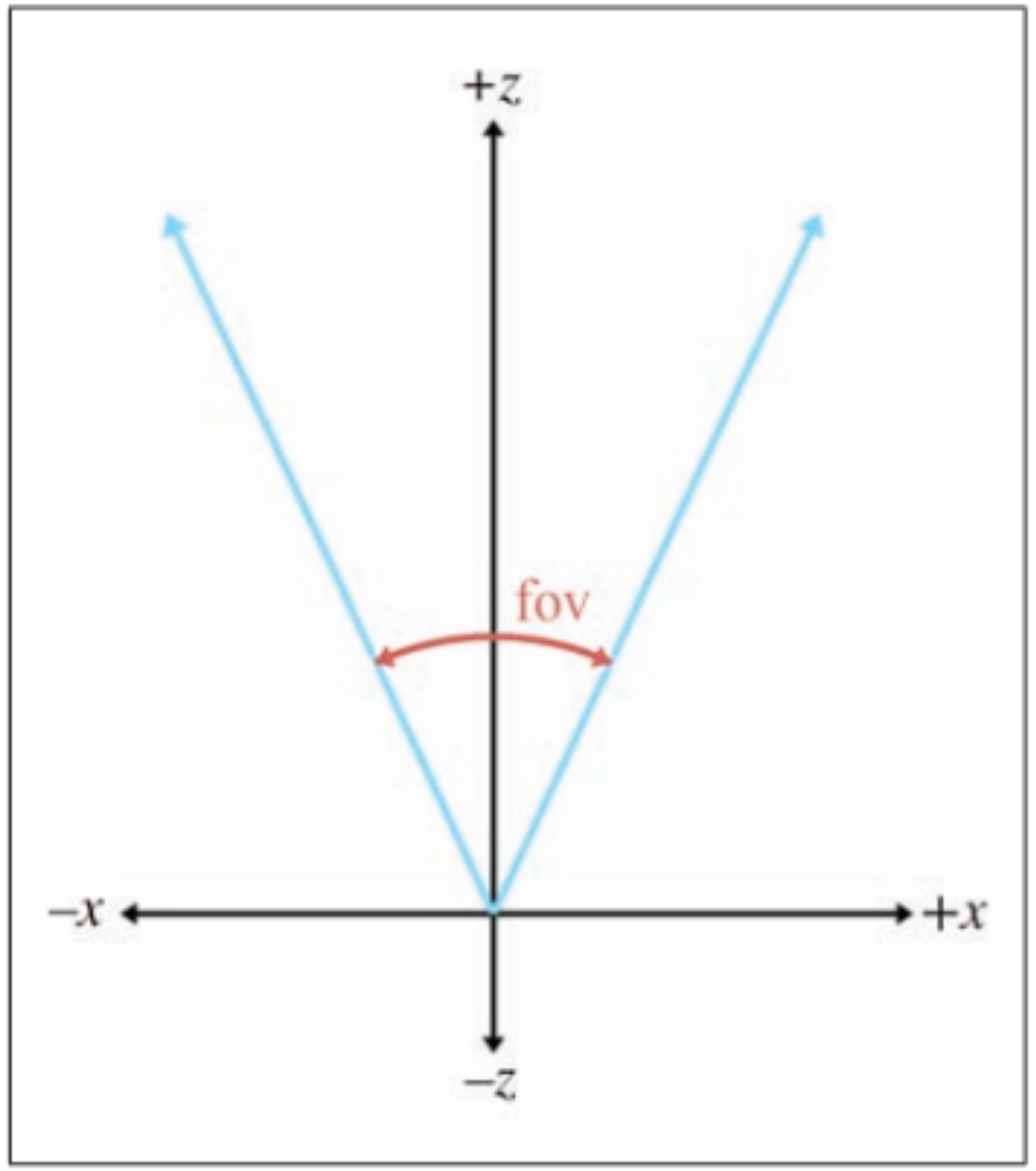
In efficient practice, there's a bit more — we'll come back to this in more detail later…

Rendering Geometry

- ✓ Transform from object to world coordinates
- ✓ Transform from world to camera coordinates
- Preprocess to more efficiently handle things outside the field of view (we're going to skip this for the moment)
- ✓ Perspective projection (to coordinates on the imaging plane)
- ✓ View transformation (to pixel coordinates on the screen)

Field of View

- All cameras have a limited field of view
- Field of view depends on the focal length
  - Zoomed in - smaller
  - Zoomed out - larger



Idea: spend as little time as possible on things that are outside the field of view!

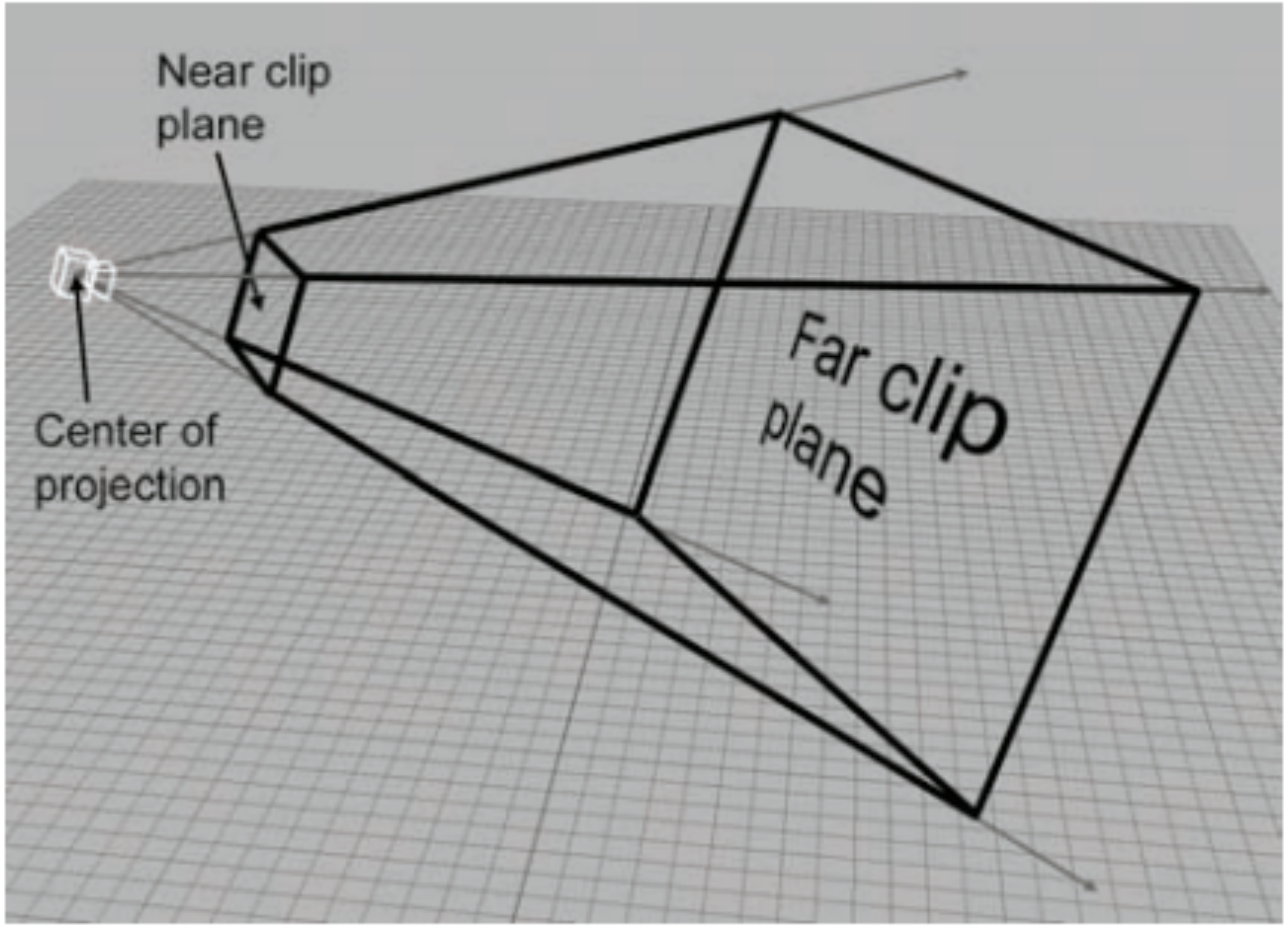
Near and Far Planes

- We don’t want to render things behind us, or perhaps even just barely in front of us
- We don’t care about things too far away to see well

“Near plane”

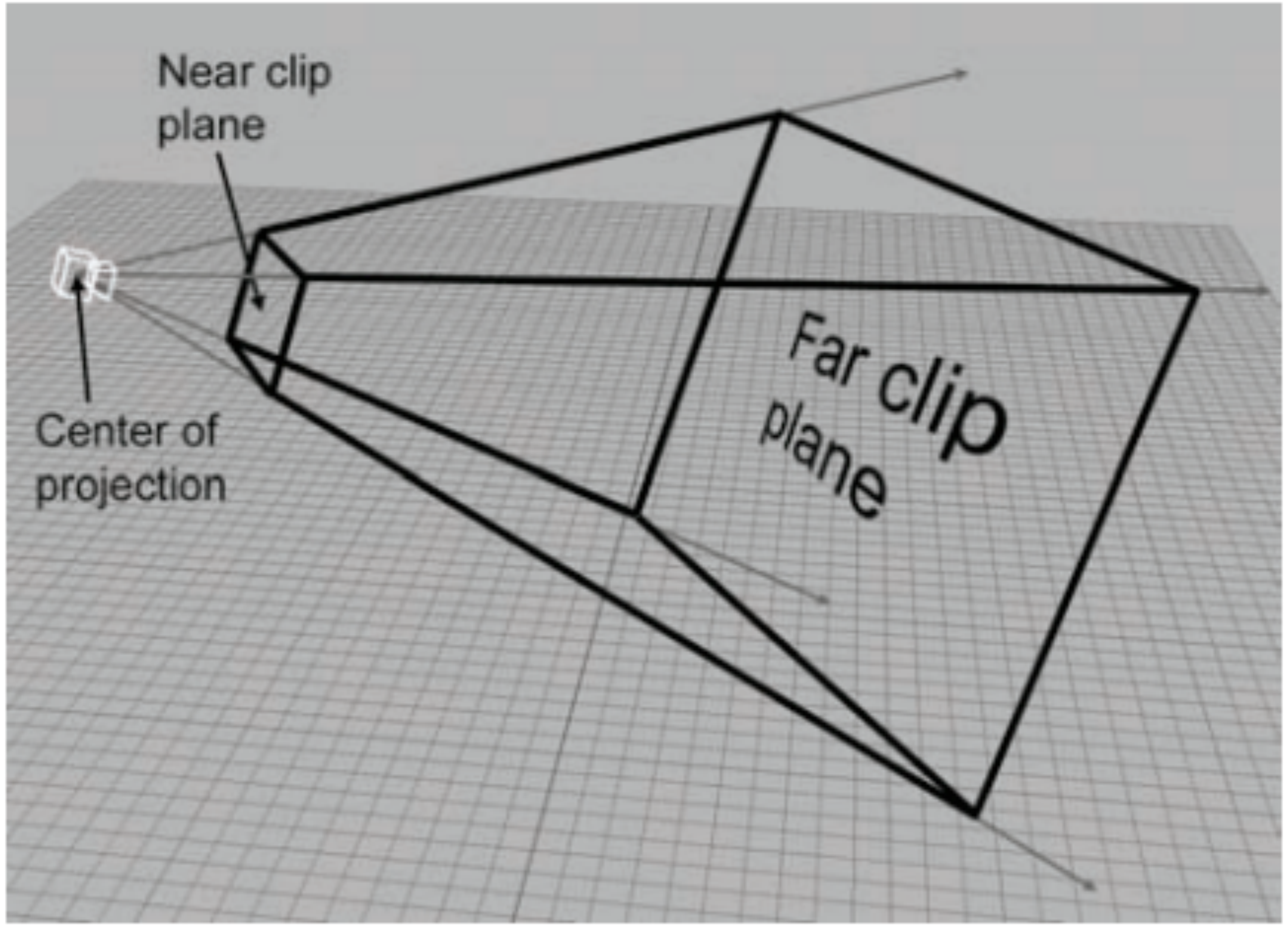
“Far plane”

View Frustum

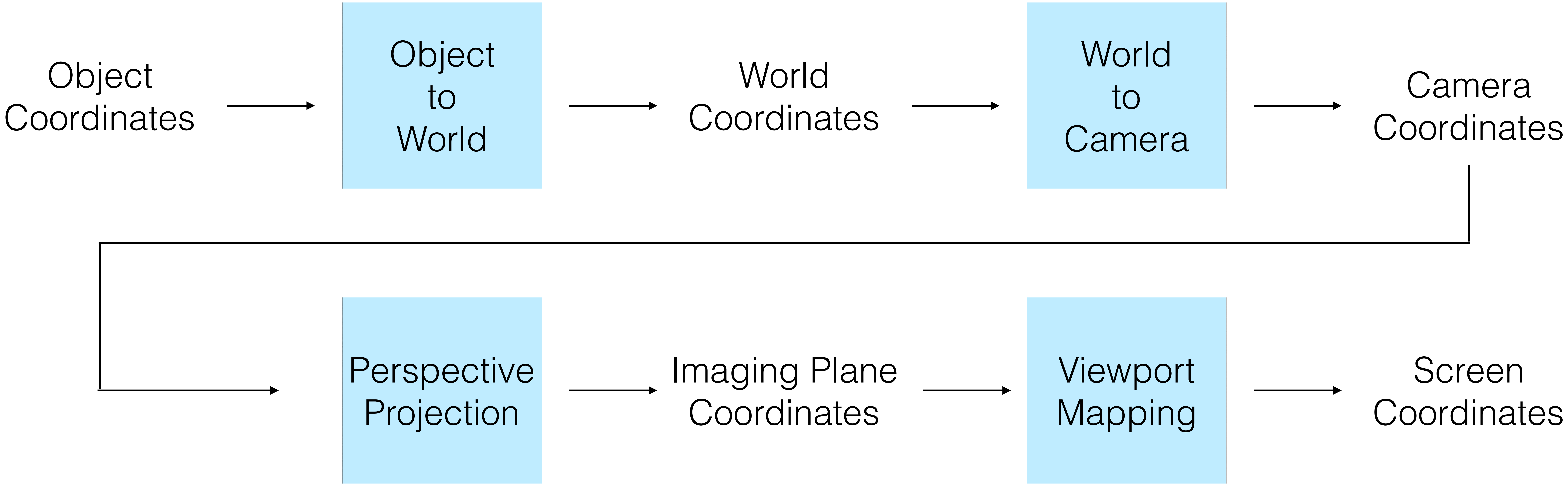


View Frustrum Clipping

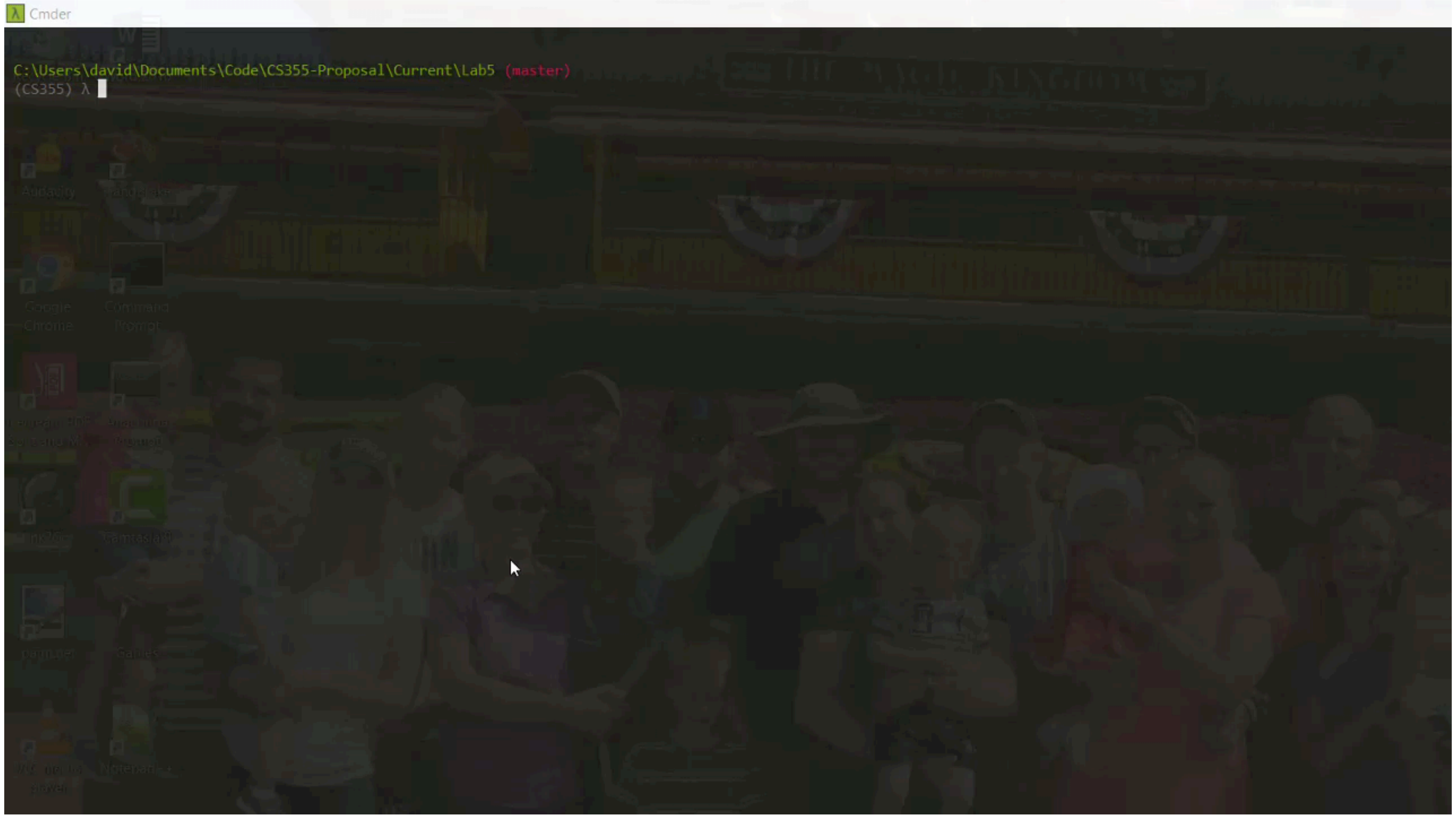
- Goal: Clip out as early as possible things that are outside the view frustum
- Idea: let's tweak our projection matrix to scale/shift things so that clipping tests are more efficient
- OpenGL will do this for you in Labs 5 and 6
- We'll go into more detail before you have to implement it yourself in Lab 7



3D Geometry Pipeline



Lab 5


Cmder
C:\Users\david\Documents\Code\CS355-Proposal\Current\Lab5 (master)
(CS355) λ

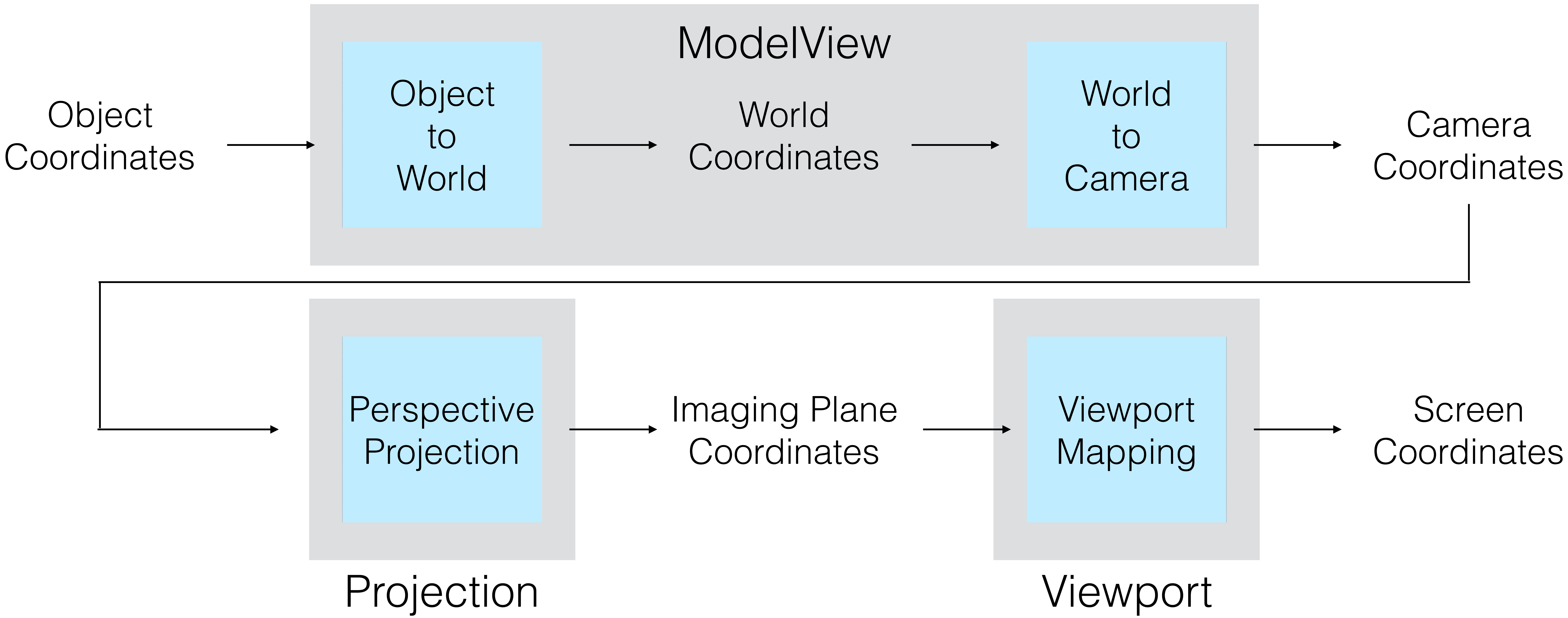
OpenGL

- OpenGL = Open Graphics Library
- Commonly used in many graphics applications
- Does the rendering for you
  - You set up the pipeline
  - You “draw” in 3D
  - It renders to the screen

OpenGL

- Key matrices in OpenGL:
  - ModelView (converts from model coordinates to camera ones)
    - Object to world
    - World to view
  - Projection
    - Orthographic
    - Perspective
  - Viewport

3D Geometry Pipeline



Lab 5: Key Functions

- glMatrixMode - changes which matrix you're manipulating
- glLoadIdentity - loads the identity matrix as the current one
- glRotated - concatenates a rotation matrix to the current one
- glTranslated - concatenates a translation matrix to the current one
- glOrtho - loads an orthographic projection matrix
- gluPerspective - loads a perspective projection matrix

Concatenating Matrices

- OpenGL concatenates new rotation/ translation operations to the **right** of the current one
- Read **right-to-left** in order of **application**
- Build **left-to-right** in **construction**

$$\mathbf{M} = \mathbf{R}\mathbf{T}$$

$$\mathbf{M} \leftarrow \mathbf{I}$$

$$\mathbf{M} \leftarrow \mathbf{M}\mathbf{R}$$

$$\mathbf{M} \leftarrow \mathbf{M}\mathbf{T}$$

Coming up...

- Hierarchical transformations
- 3D rendering geometry
  - More details
  - Efficient implementation
- Visibility
- Lighting