Geometry: Cameras

Course web page:

http://goo.gl/EB3aA

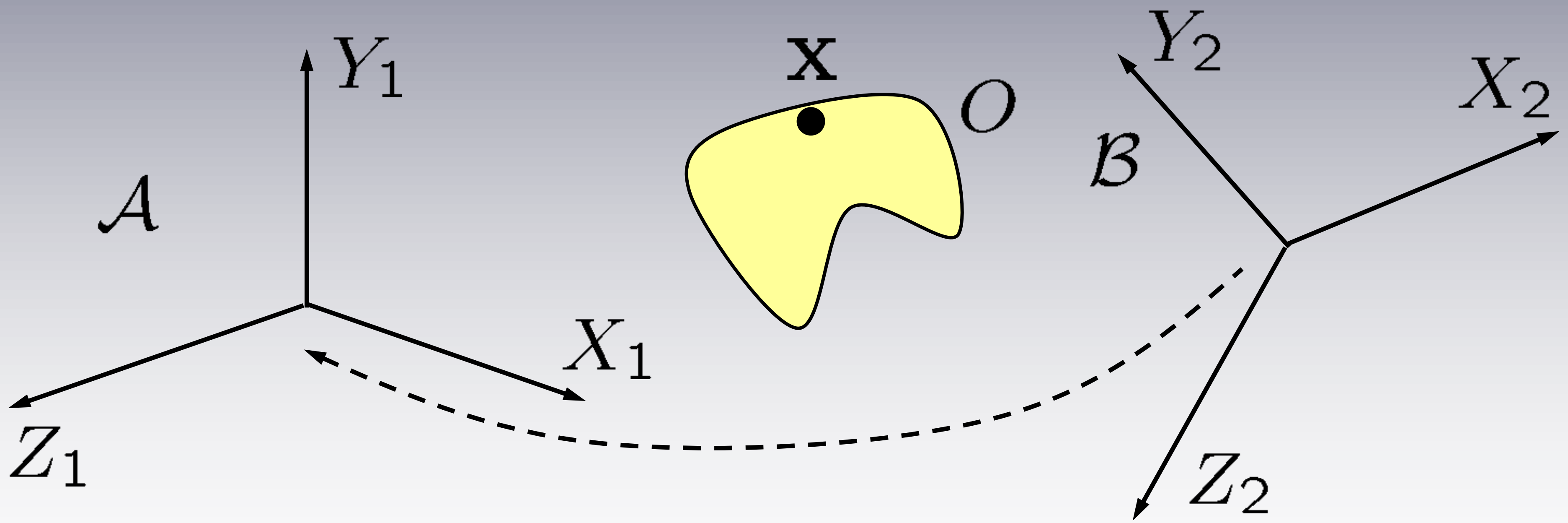
UNIVERSITY OF DELAWARE

Outline

- More 3-D transformations
  - Setting up the camera
- Projections
  - Orthographic
  - Perspective

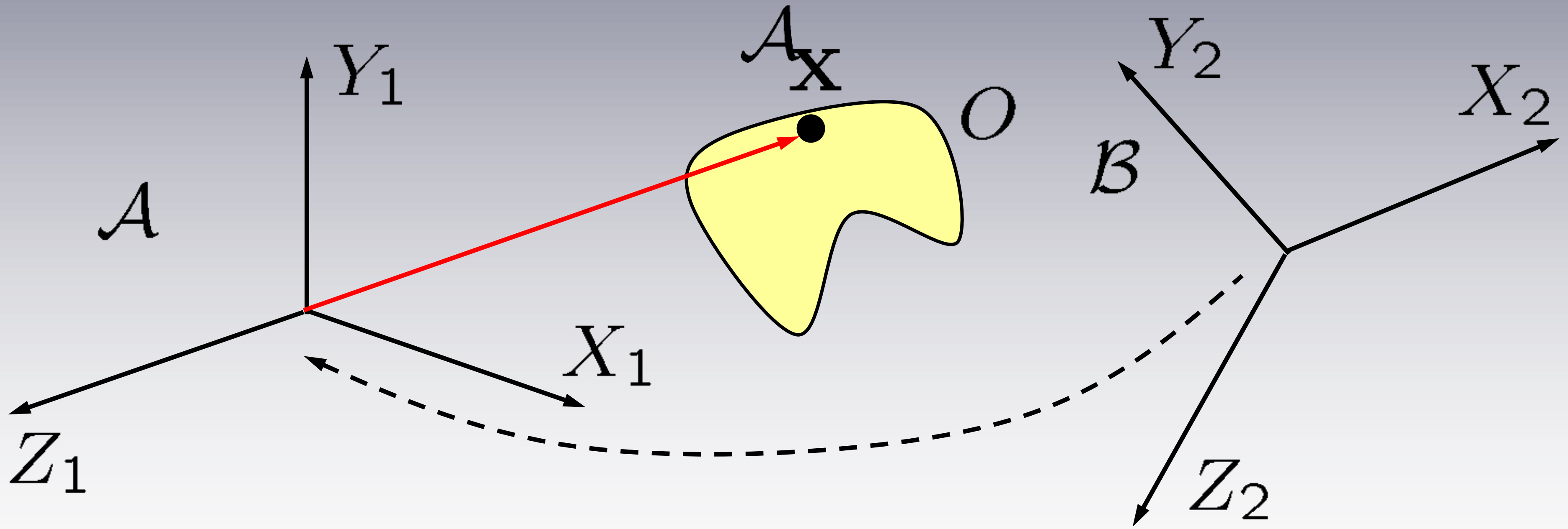
3-D Transformations: Arbitrary Change of Coordinates

- A **rigid transformation** is used to represent a change in the coordinate system that "expresses" a point's location



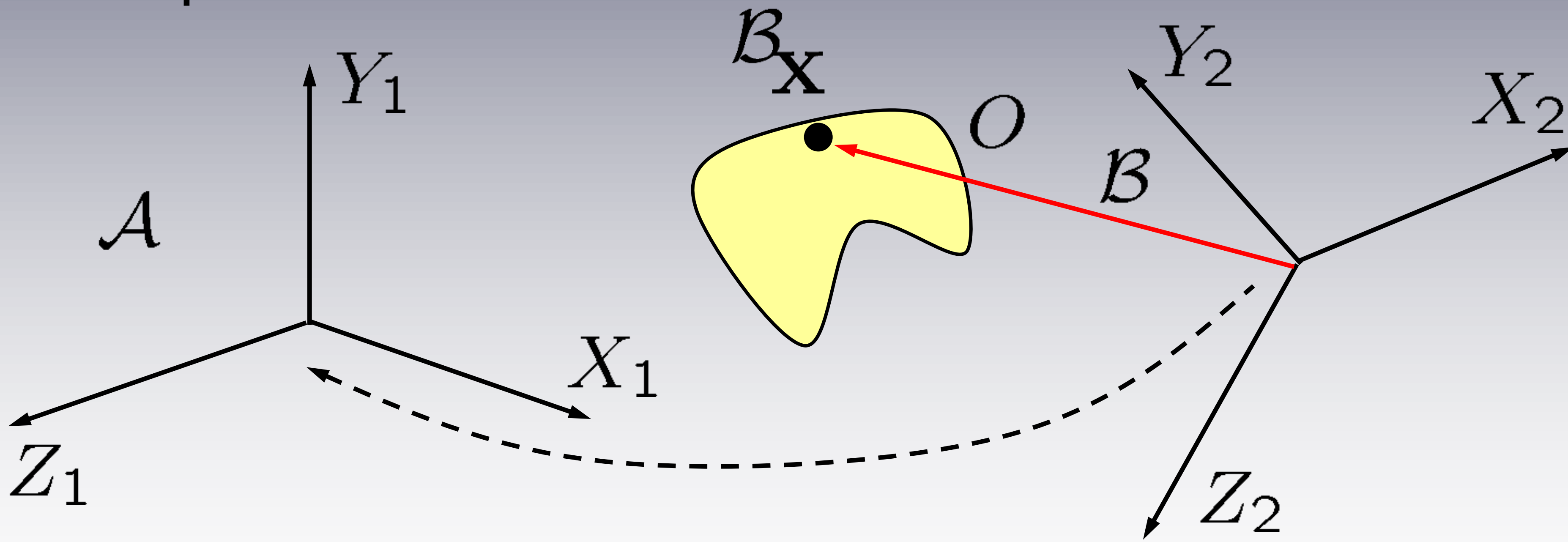
3-D Transformations: Arbitrary Change of Coordinates

- A **rigid transformation** is used to represent a change in the coordinate system that "expresses" a point's location



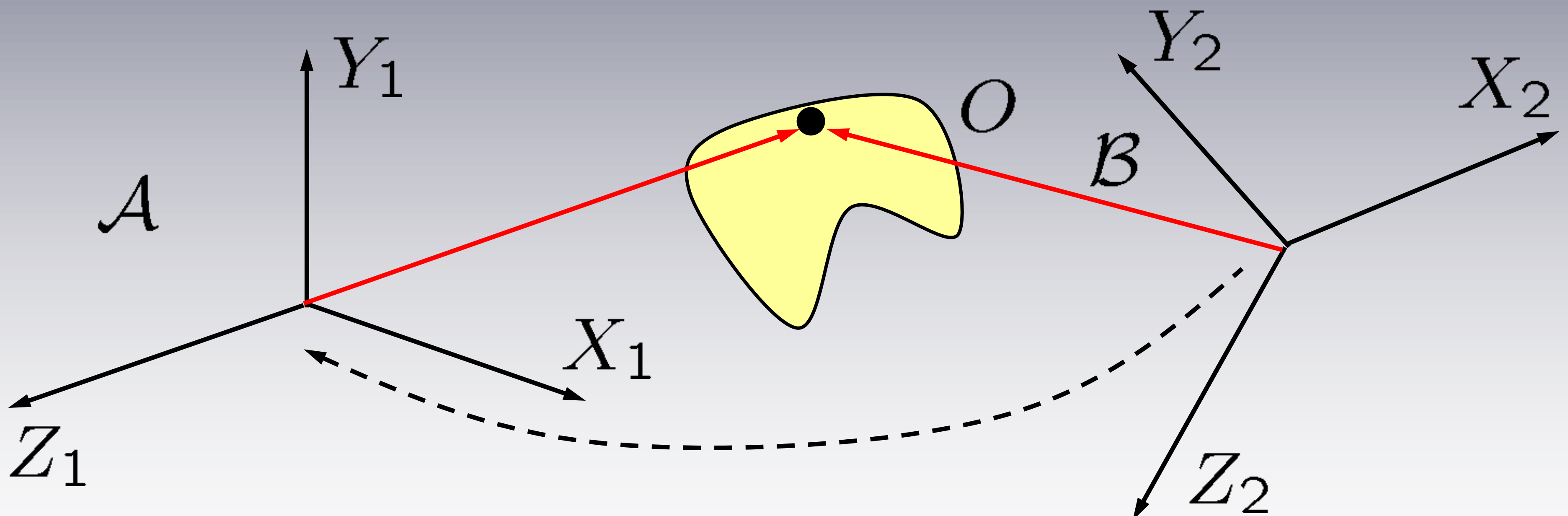
3-D Transformations: Arbitrary Change of Coordinates

- A **rigid transformation** is used to represent a change in the coordinate system that "expresses" a point's location



3-D Transformations: Arbitrary Change of Coordinates

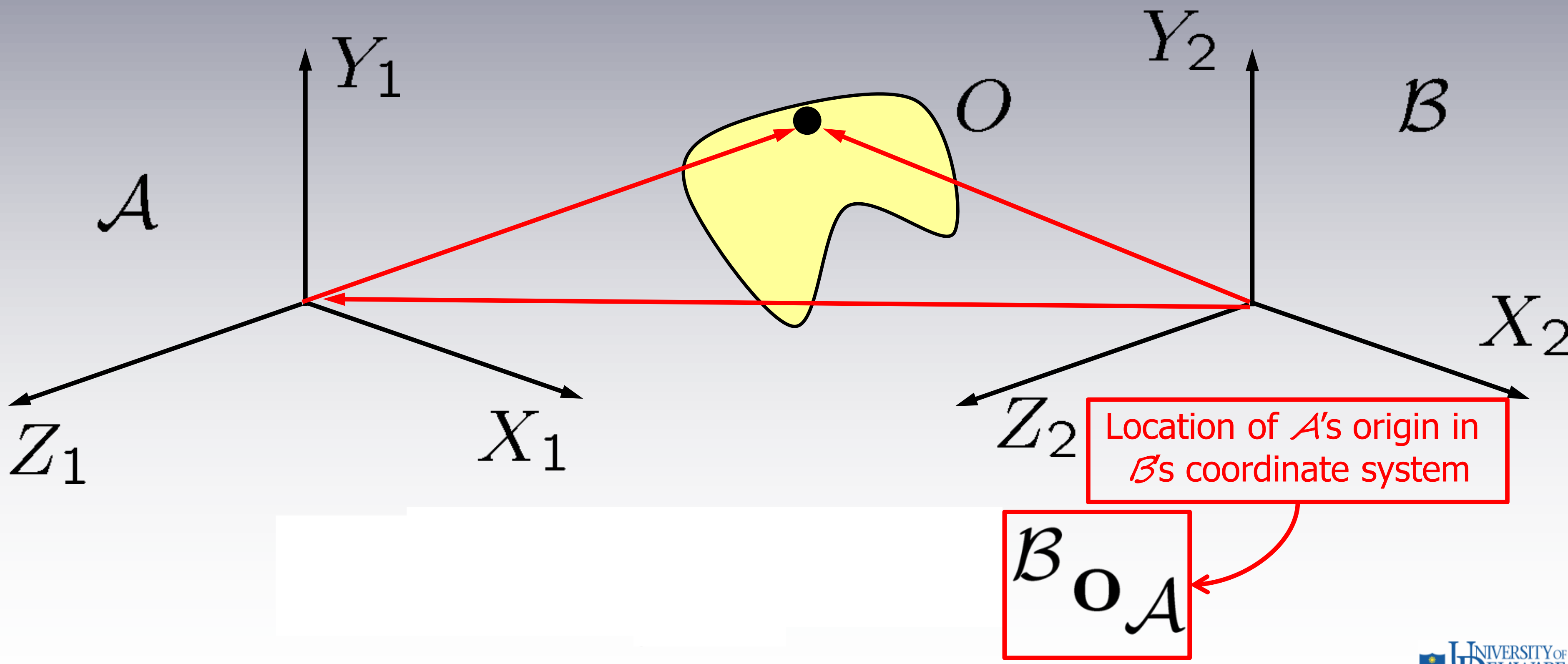
- A **rigid transformation** is used to represent a change in the coordinate system that "expresses" a point's location



$$^{\mathcal{B}}\mathbf{x} = {}^{\mathcal{B}}_{\mathcal{A}}\mathrm{M}\,{}^{\mathcal{A}}\mathbf{x}$$

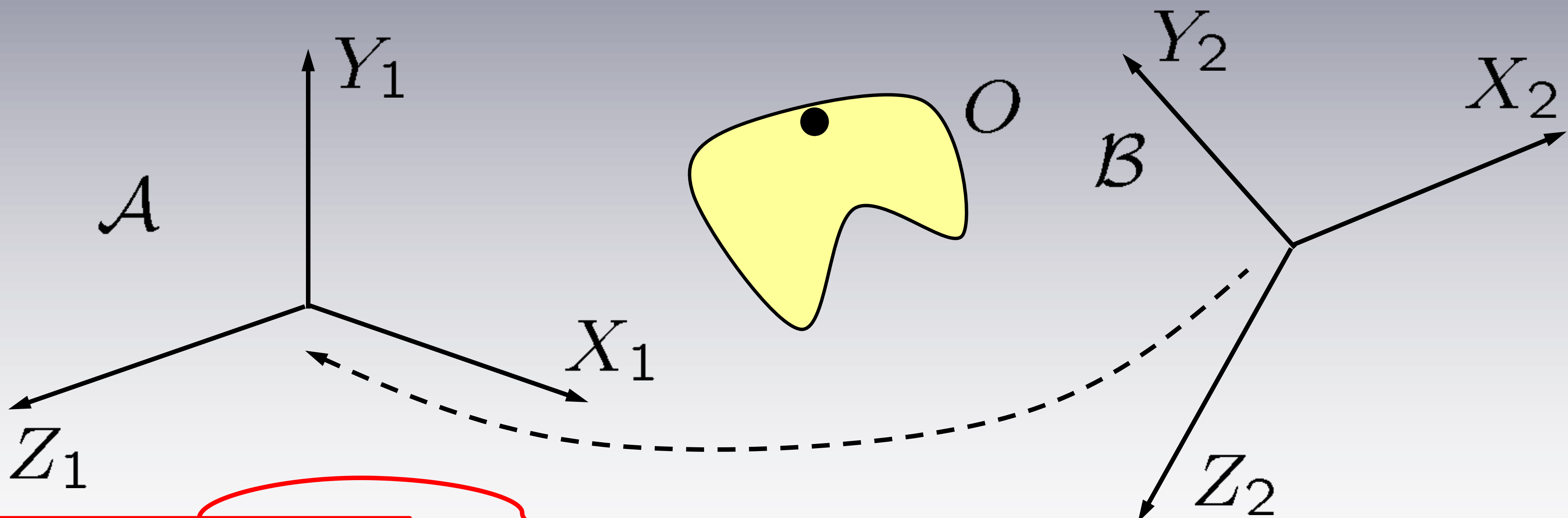
3-D Transformations: Translation-only Change of Coordinates

- A **rigid transformation** is used to represent a change in the coordinate system that "expresses" a point's location



3-D Transformations: Arbitrary Change of Coordinates

- A **rigid transformation** is used to represent a change in the coordinate system that "expresses" a point's location

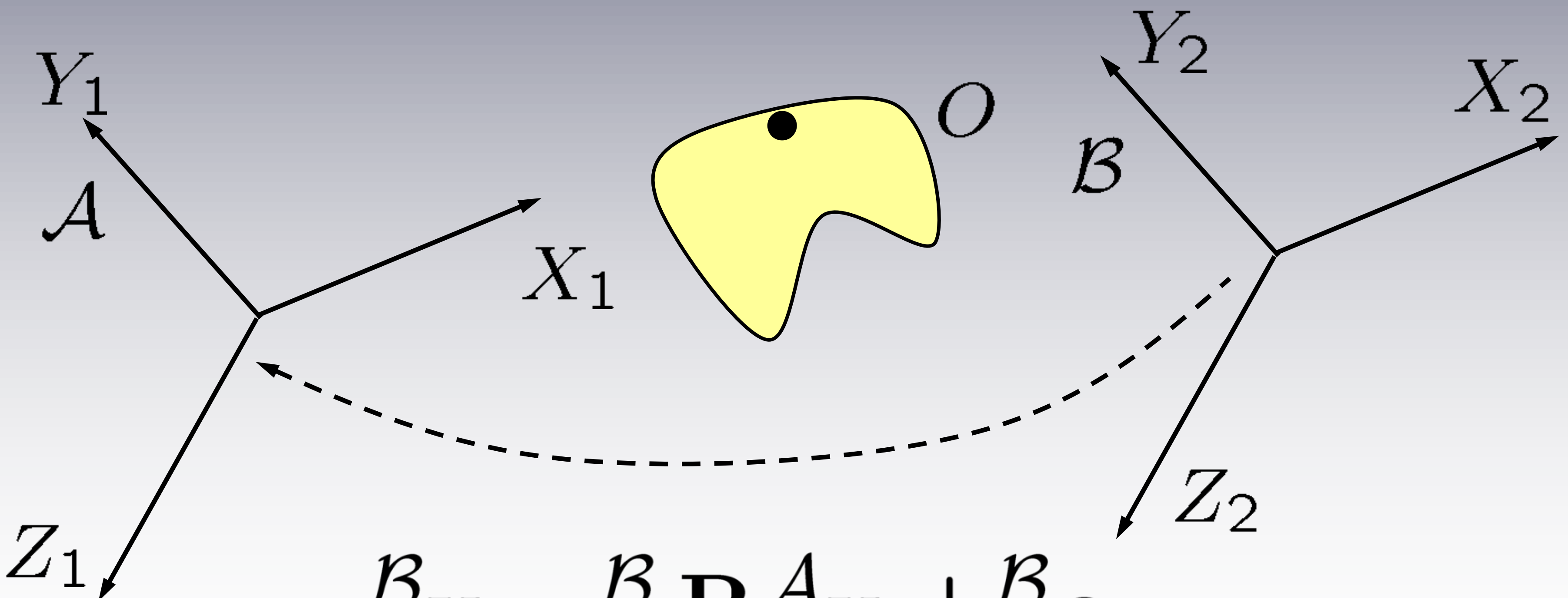


Rotation that makes $\mathcal{A}$'s axes parallel to $\mathcal{B}$s, creating translation-only case

$$ {}^{\mathcal{B}}\mathbf{x} = \boxed{{}^{\mathcal{B}}_{\mathcal{A}}\mathbf{R}}\,{}^{\mathcal{A}}\mathbf{x} + {}^{\mathcal{B}}\mathbf{o}_{\mathcal{A}} $$

3-D Transformations: Arbitrary Change of Coordinates

- A **rigid transformation** is used to represent a change in the coordinate system that “expresses” a point’s location



$$ {}^{\mathcal{B}}\mathbf{x} = {}^{\mathcal{B}}_{\mathcal{A}}\mathbf{R}\,{}^{\mathcal{A}}\mathbf{x} + {}^{\mathcal{B}}\mathbf{o}_{\mathcal{A}} $$

3-D Rigid Transformations

- Combination of rotation followed by translation, without scaling, etc.
- "Moves" an object from one 3-D pose to another

$$\begin{pmatrix} 1 & 0 & 0 & \Delta x \\ 0 & 1 & 0 & \Delta y \\ 0 & 0 & 1 & \Delta z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & \Delta x \\ r_{21} & r_{22} & r_{23} & \Delta y \\ r_{31} & r_{32} & r_{33} & \Delta z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

T **R** **M**

Rigid Transformations: Homogeneous Coordinates

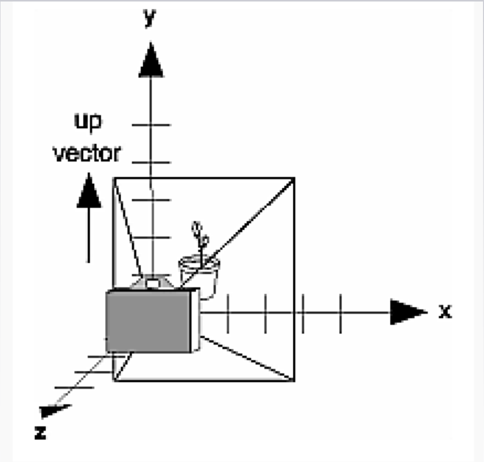
- Points in one coordinate system are transformed to the other as follows:

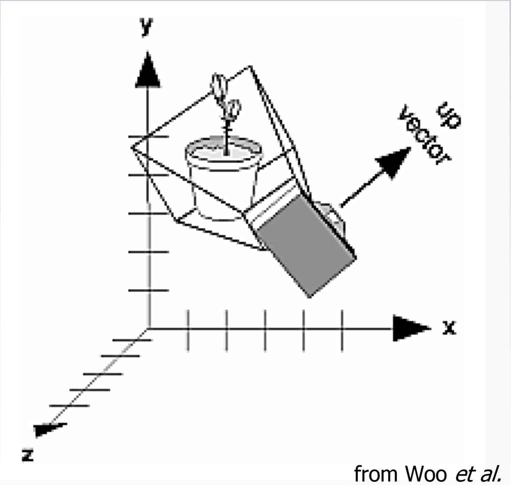
$$^{\mathcal{B}}\mathbf{x} = {}^{\mathcal{B}}_{\mathcal{A}}\mathrm{M}\,{}^{\mathcal{A}}\mathbf{x} = \begin{pmatrix} {}^{\mathcal{B}}_{\mathcal{A}}\mathbf{R} & {}^{\mathcal{B}}\mathbf{o}_{\mathcal{A}} \\ \mathbf{0}^T & 1 \end{pmatrix} {}^{\mathcal{A}}\mathbf{x}$$

- Rows of rotation matrix are *B*'s axes "in" *A*'s coordinate system
- ${}^{\mathcal{C}}_{\mathcal{W}}\mathrm{M}$ takes the camera to the world origin, transforming points expressed in **world coordinates** into points expressed in **camera coordinates**
  - Info needed: Camera axes in world coordinates, world origin in camera coordinates

Controlling the camera position

- Standard OpenGL position: At $(0, 0, 0)^T$ in world coordinates looking in -Z direction (0, 0, -1) with **up vector** $(0, 1, 0)^T$
  - Up vector controls camera roll (rotation around Z axis)
- Changing position: **`gluLookAt()`**
  - **eye** = (**`eyeX`**, **`eyeY`**, **`eyeZ`**)^T: Desired camera position
  - **center** = (**`centerX`**, **`centerY`**, **`centerZ`**)^T: Point at which camera is aimed (defining "gaze direction")
  - **up** = (**`upX`**, **`upY`**, **`upZ`**)^T: Camera's "up" vector
- Robins' `projection` tutor



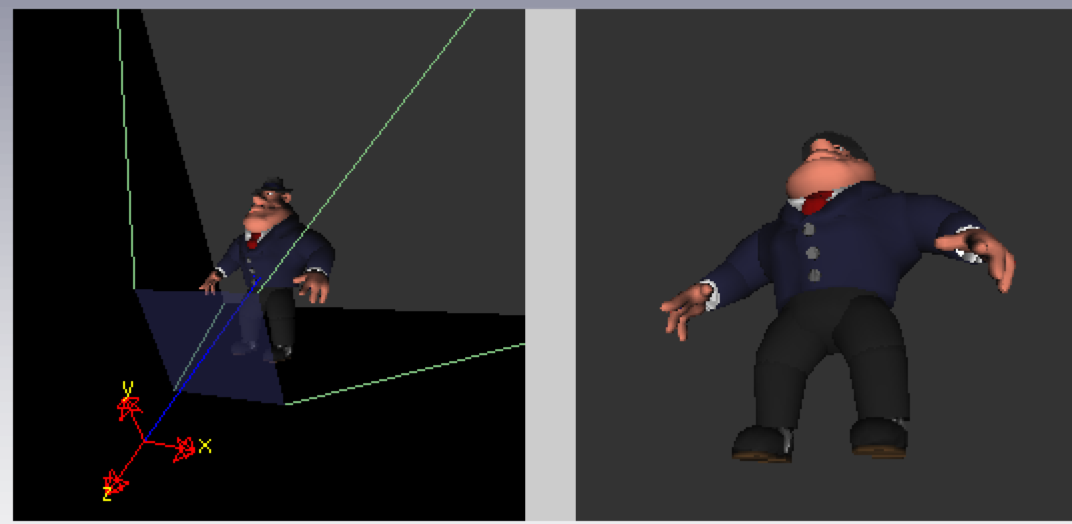


from Woo *et al.*

University of Delaware

The Viewing Volume

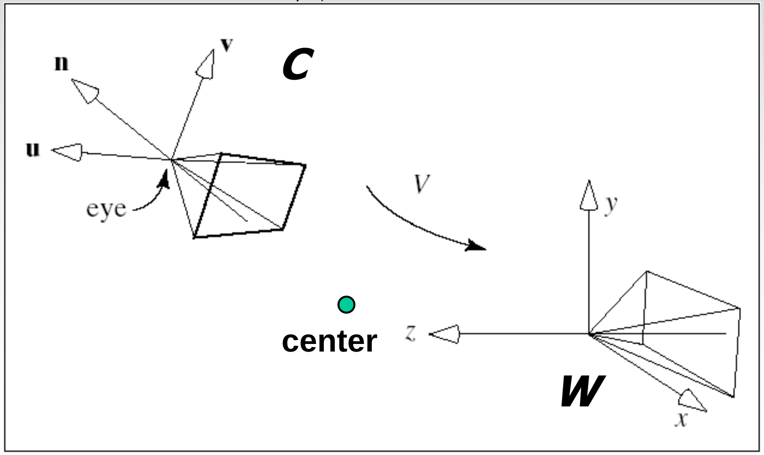
- Definition: The region of 3-D space visible in the image
- Depends on:
  - Camera position, orientation
  - Field of view, image size
  - Projection type
    - Orthographic
    - Perspective



courtesy of N. Robins

`gluLookAt()`: Details (7.1.3 in Shirley)

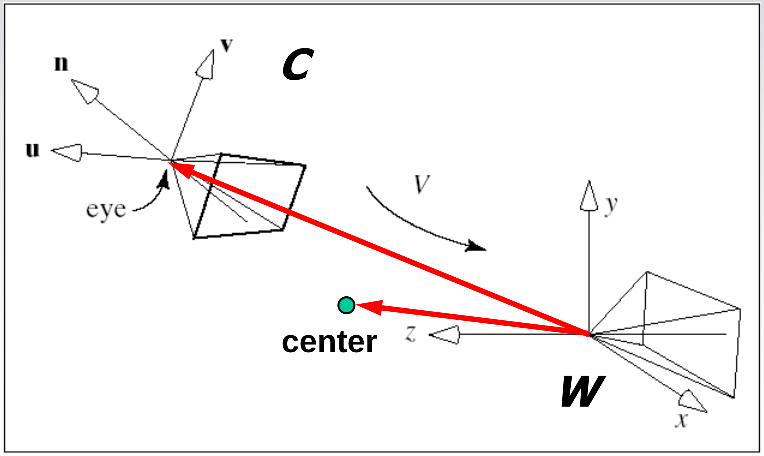
- Moves scene points so that camera is at origin, “look at” point is on -Z axis, and camera +Y axis is aligned with **up** vector
  - Create and execute rigid transformation ${}^{\mathcal{C}}_{\mathcal{W}}\mathrm{M}$ making a change from world to camera coordinates
- Steps
  1. Compute vectors **u**, **v**, **n** defining new **camera axes** in **world coordinates (**Shirley textbook uses **w** instead of **n)**
     - “Old” axes are **u’** = $(1, 0, 0)^T$, **v’** = $(0, 1, 0)^T$, **n’ =** $(0, 0, 1)^T$
  2. Compute location ${}^{\mathcal{C}}\mathbf{o}_{\mathcal{W}}$ of old camera position in terms of new location’s coordinate system
  3. Fill in rigid transform matrix ${}^{\mathcal{C}}_{\mathcal{W}}\mathrm{M}$



from Hill

`gluLookAt()`: Camera axes in world coords.

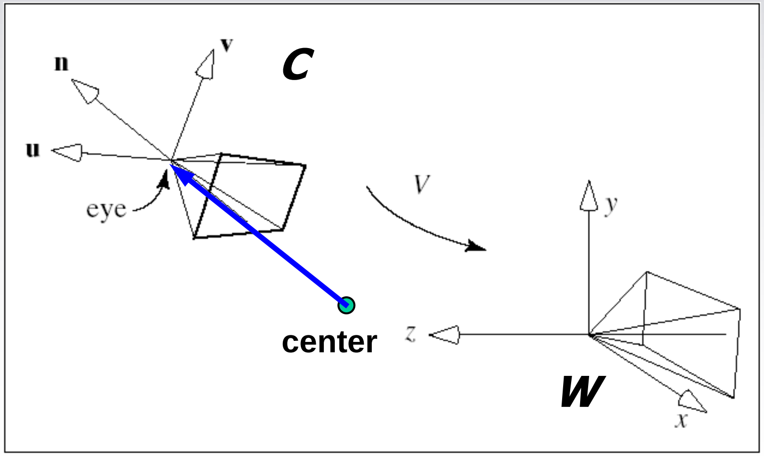
- Form basis vectors
  - New camera Z axis: **n** = **eye** - **center**
  - New camera X axis: **u** = **up** x **n**
  - New camera Y axis: **v** = **n** x **u** (not necessarily same as **up**)
- Normalize so that these are unit vectors



from Hill

`gluLookAt()`: Camera axes in world coords.

- Form basis vectors
  - New camera Z axis: **n** = **eye** - **center**
  - New camera X axis: **u** = **up** x **n**
  - New camera Y axis: **v** = **n** x **u** (not necessarily same as **up**)
- Normalize so that these are unit vectors

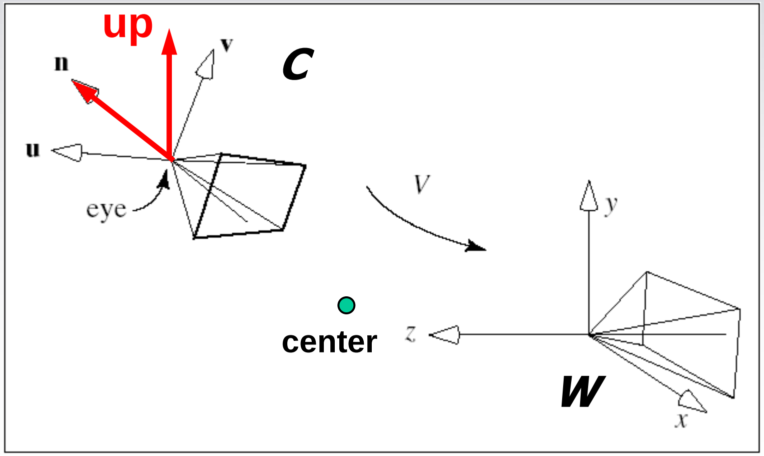


from Hill

gluLookAt(): Camera axes in world coords.

- Form basis vectors
  - New camera Z axis: **n** = **eye** - **center**
  - New camera X axis: **u** = **up** x **n**
  - New camera Y axis: **v** = **n** x **u** (not necessarily same as **up**)
- Normalize so that these are unit vectors

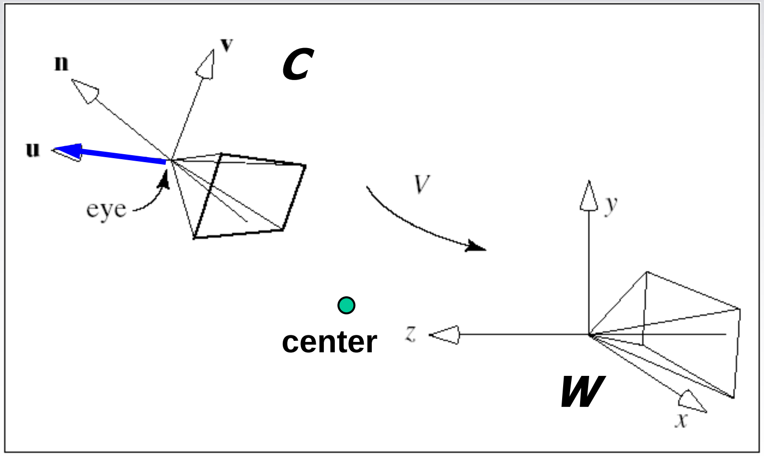
up and **n** define a plane which **u** is normal to—they don't have to be orthogonal



from Hill

`gluLookAt()`: Camera axes in world coords.

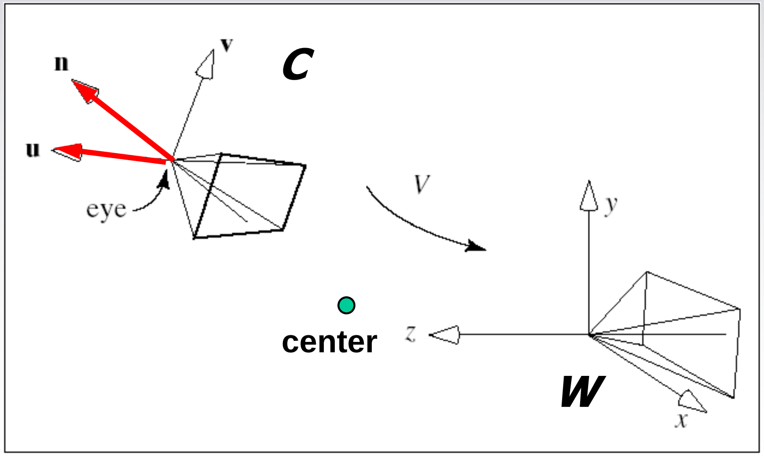
- Form basis vectors
  - New camera Z axis: **n** = **eye** - **center**
  - New camera X axis: **u** = **up** x **n**
  - New camera Y axis: **v** = **n** x **u** (not necessarily same as **up**)
- Normalize so that these are unit vectors



from Hill

`gluLookAt()`: Camera axes in world coords.

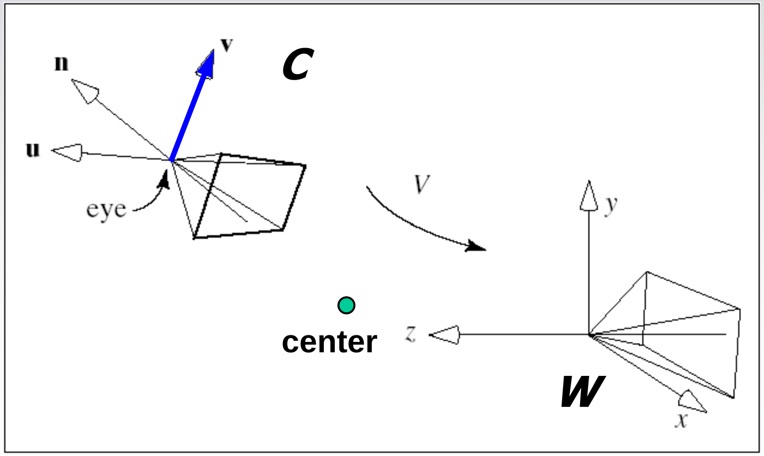
- Form basis vectors
  - New camera Z axis: **n** = **eye** - **center**
  - New camera X axis: **u** = **up** x **n**
  - New camera Y axis: **v** = **n** x **u** (not necessarily same as **up**)
- Normalize so that these are unit vectors



from Hill

`gluLookAt()`: Camera axes in world coords.

- Form basis vectors
  - New camera Z axis: **n** = **eye** - **center**
  - New camera X axis: **u** = **up** x **n**
  - New camera Y axis: **v** = **n** x **u** (not necessarily same as **up**)
- Normalize so that these are unit vectors



from Hill

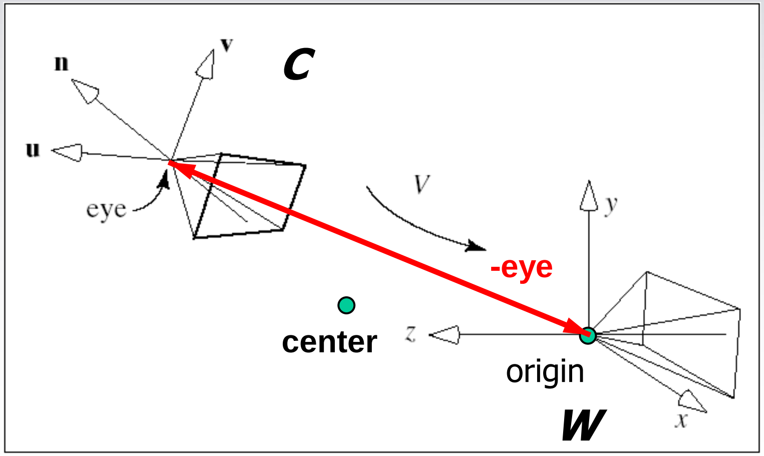
`gluLookAt()`: Camera axes in world coords.

- Now make 3 x 3 rotation matrix from formula on rigid transform slide:

$$ {}^{\mathcal{C}}_{\mathcal{W}}\mathbf{R} = \begin{pmatrix} \mathbf{u}^T \\ \mathbf{v}^T \\ \mathbf{n}^T \end{pmatrix} $$

`gluLookAt()`: Location

- ${}^{\mathcal{C}}\mathbf{o}_{\mathcal{W}}$: World origin in camera coordinates

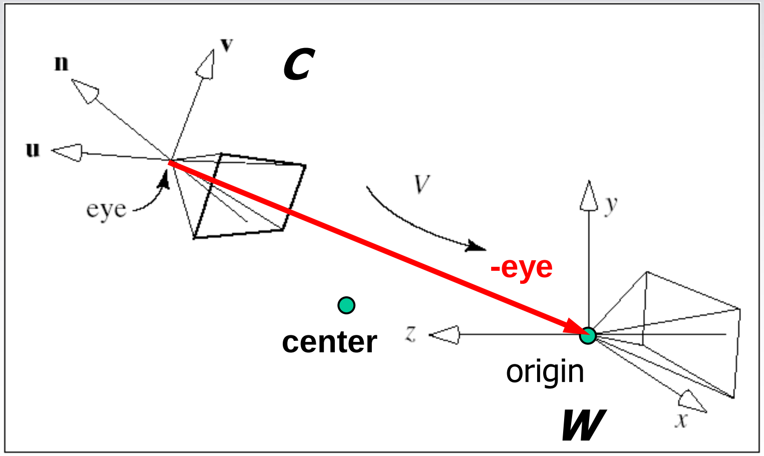


from Hill

`gluLookAt()`: Location

- ${}^{\mathcal{C}}\mathbf{o}_{\mathcal{W}}$: World origin in camera coordinates
- **-eye** is in world coordinates, so project onto camera axes (and don't normalize):

$$\boxed{{}^{\mathcal{C}}\mathbf{o}_{\mathcal{W}} = (-\mathbf{eye}\cdot\mathbf{u}, -\mathbf{eye}\cdot\mathbf{v}, -\mathbf{eye}\cdot\mathbf{n})^T}$$



from Hill

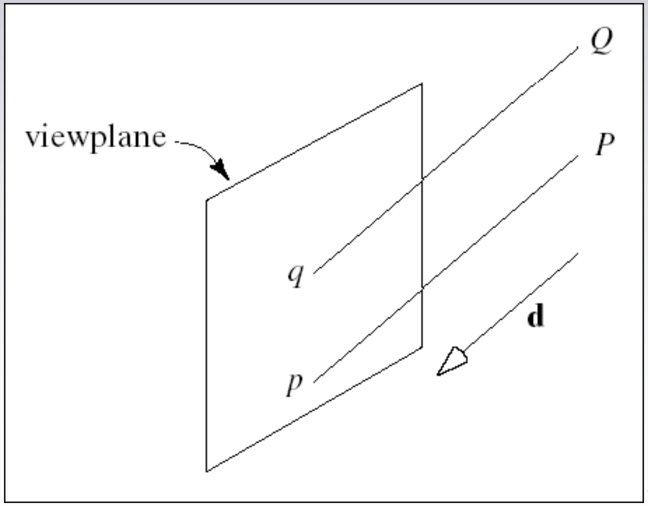
`gluLookAt()`: Matrix

- Letting $\mathrm{t} = {}^{\mathcal{C}}\mathbf{o}_{\mathcal{W}}$ and writing the vector components as **u** = $(u_x, u_y, u_z)^T$, etc., the final transformation matrix is given by:

$$ {}^{\mathcal{C}}_{\mathcal{W}}\mathrm{M} = \begin{pmatrix} u_x & u_y & u_z & t_x \\ v_x & v_y & v_z & t_y \\ n_x & n_y & n_z & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} $$

Transformations vs. Projections

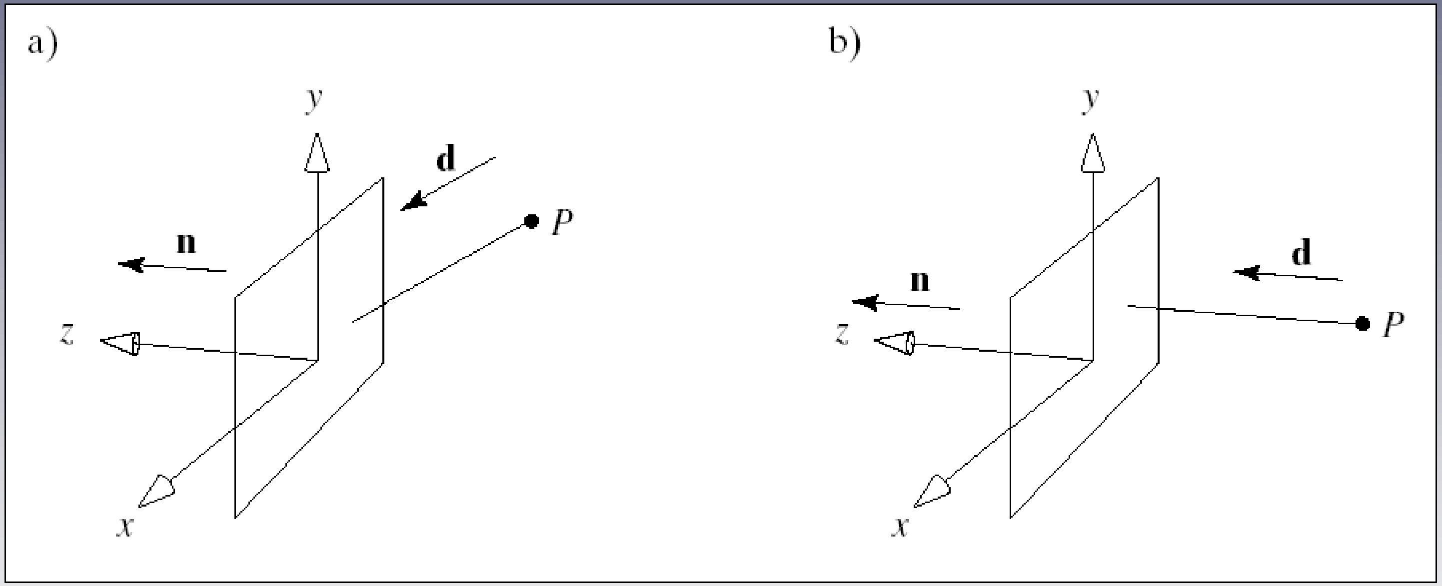
- **Transformation**: Mapping within *n*-D space that moves points around
  - Linear transformations (e.g. matrix multiplication) preserve straight lines
  - Some nonlinear transformations in *n*-D can be expressed by linear ones in (n + 1)-D – the idea behind homogeneous coordinates
- **Projection**: Mapping from *n*-D space down to lower-dimensional subspace
  - E.g., point in 3-D space to point on plane (a 2-D entity) in that space
  - We will be interested in such 3-D to 2-D projections where the plane is the **image**
  - Things to know:
    - Where are the points?
    - Where is the plane?
    - What kind of projection?



from Hill

Parallel projection along direction **d** onto a plane

Parallel Projections



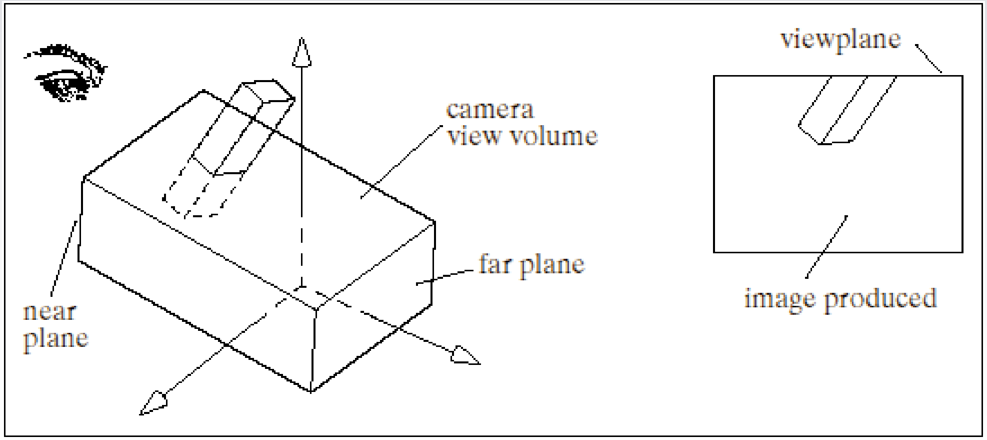
from Hill

Oblique: **d** in general position relative to plane normal **n**

Orthographic: **d** parallel to **n**

Orthographic Projection

- Projection direction **d** is aligned with Z axis
- Viewing volume is "brick"-shaped region in space
  - Not the same as image size
- No perspective effects—distant objects look same as near ones, so camera (x, y, z) ⇒ image (x, y)



from Hill

Simple Orthographic Projection Matrix

$$Pv = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \\ 1 \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ 0 \\ 1 \end{bmatrix}$$